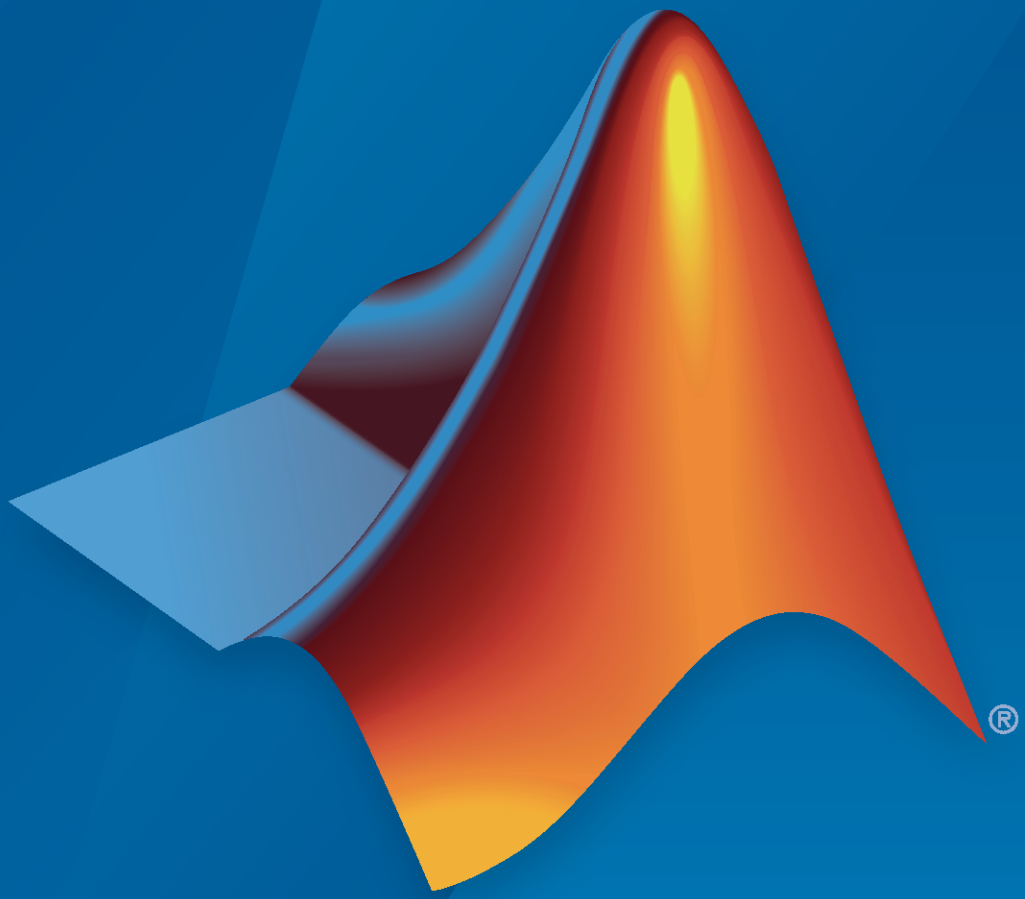
5G Toolbox™

Reference



MATLAB®

R2020b

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

5G Toolbox™ Reference

Revision History

September 2018	Online only	New for Version 1.0 (Release 2018b)
March 2019	Online only	Revised for Version 1.1 (Release 2019a)
September 2019	Online only	Revised for Version 1.2 (Release 2019b)
March 2020	Online only	Revised for Version 2.0 (Release 2020a)
September 2020	Online only	Revised for Version 2.1 (Release 2020b)

Contents

Functions

displayChannel

Visualize and explore CDL channel model characteristics

Syntax

```
fig = displayChannel(cdl)
fig = displayChannel(cdl,Name,Value)
```

Description

`fig = displayChannel(cdl)` displays geometric and electromagnetic characteristics of the specified clustered delay line (CDL) channel model at the transmitter and the receiver ends. The visualization includes the position, polarization, and directivity radiation pattern of the antenna elements, cluster paths directions, and average path gains. Because all antenna elements are equal, the visualization shows the radiation pattern of the first antenna element only and displays the cluster paths directions centered also at the first antenna element. By adding customized data tips to the visualization windows, you can explore antenna element, element pattern, and cluster paths characteristics. The function also returns an array of `figure` objects that correspond to the displayed visualization windows.

`fig = displayChannel(cdl,Name,Value)` specifies visualization options of the displayed channel characteristics by using one or more name-value pair arguments. For example, `'LinkEnd','Tx'` specifies visualization for the transmitter end only. Unspecified options take default values.

Examples

Visualize CDL Channel Model Characteristics

This example shows how to visualize CDL channel characteristics and explore channel information about the antenna element, element pattern, and cluster paths.

Define the channel configuration structure by using an `nrCDLChannel` System object. Specify the delay profile as CDL-D.

```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
```

Configure the transmit array size as a vector of the form $\left[M\,N\,P\,M_g\,N_g\right]$ = [4 3 2 1 2], which specifies two rectangular panels (M_g = 1and N_g = 2) of a 4-by-3 antenna array (M = 4 and N = 3) and two polarizations (P = 2). The total number of polarized elements in the array is $M \times N \times P \times M_g \times N_g = 48$.

```
txSize = [4 3 2 1 2];
cdl.TransmitAntennaArray.Size = txSize;
```

Configure the vertical and horizontal element spacing and the vertical and horizontal panel spacing, in wavelength, as a vector of the form $[\lambda_v\,\lambda_h\,dg_v\,dg_h]$. Because panel spacing is measured from the

center of the panels, to avoid panel overlapping, set dg_h to a value greater than 1 wavelength. To ensure uniform antenna element spacing across vertically and horizontally separated panels, configure panel spacings as $dg_v = \lambda_v \times M$ and $dg_h = \lambda_h \times N$, respectively.

```
lambda_v = 0.5;
lambda_h = 0.5;
dg_v = lambda_v*txSize(1); % lambda_v * M
dg_h = lambda_h*txSize(2); % lambda_h * N
cdl.TransmitAntennaArray.ElementSpacing = [lambda_v lambda_h dg_v dg_h];
```

Configure the mechanical orientation of the array as $\left[\alpha\ \beta\ \gamma\right]^T = [0\ 15\ 0]^T$, which specifies 0 degrees bearing, 15 degrees downtilt, and 0 degrees slant.

```
cdl.TransmitAntennaArray.Orientation = [0 15 0]';
```

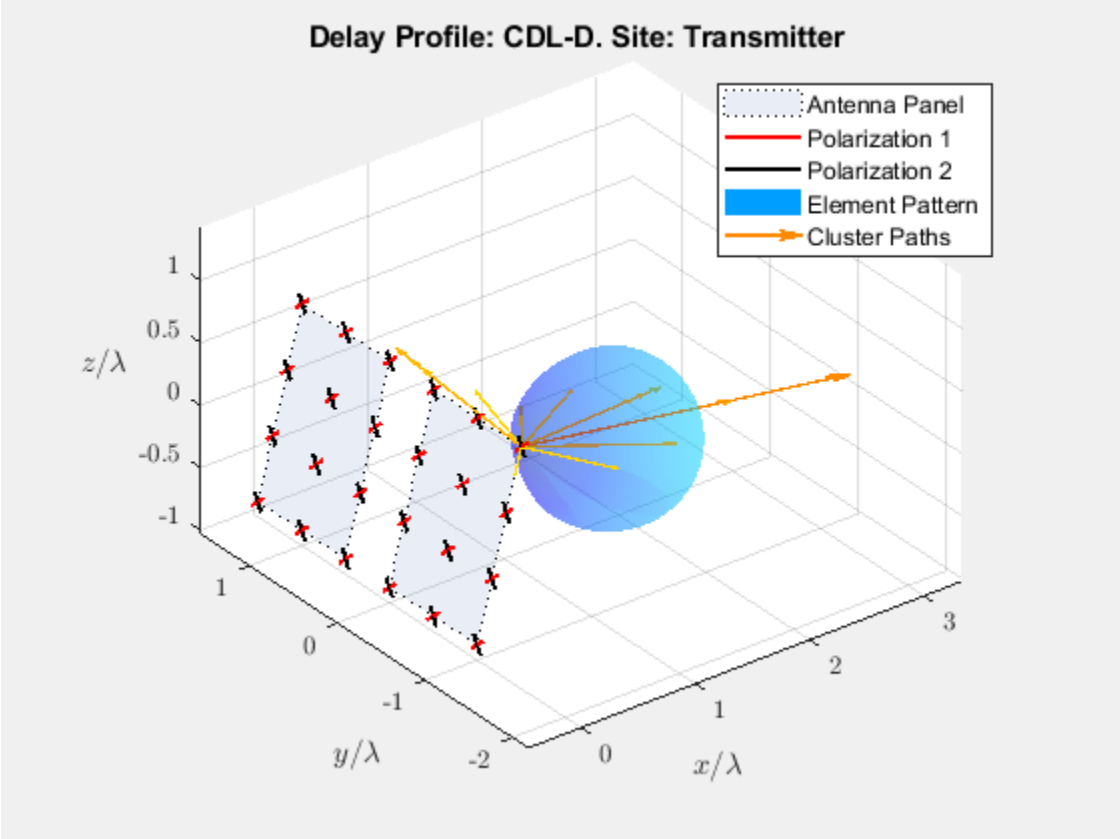
For an overview of all transmit antenna array properties, see the “TransmitAntennaArray” on page 2-0 property of the `nrCDLChannel` System object.

Display the channel characteristics at the transmitter end.

```
figTx = displayChannel(cdl,'LinkEnd','Tx');
```

The generated figure supports customized data tips. Add data tips in the current figure by enabling the data cursor mode.

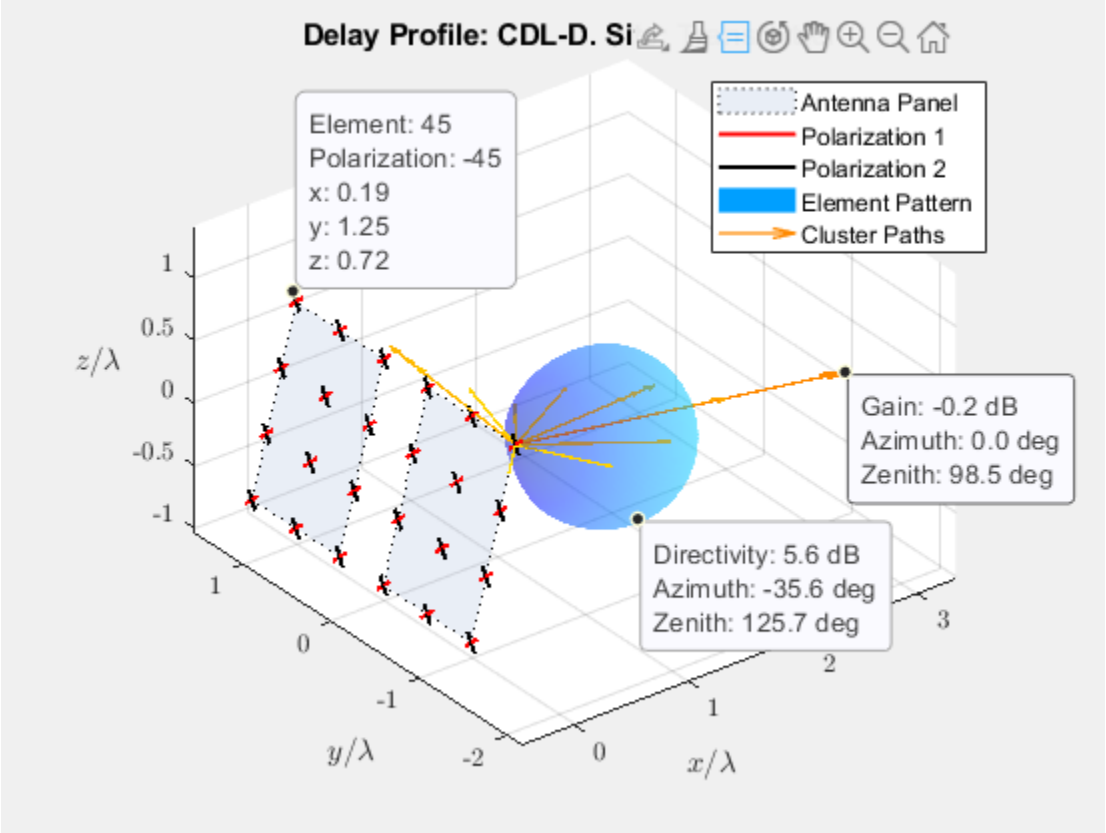
```
datacursormode on;
```



With data cursor mode enabled, explore channel characteristics by adding data tips. To create a data tip, click a data point. To create multiple data tips, press the **Shift** key while clicking the data points.

For example, this figure shows data tips added to the antenna element, element pattern, and cluster paths at the transmitter end.

- Antenna element data tips include information about the position, polarization angle, and element number of each antenna element. The element numbers indicate the order in which the channel model maps input signals column-wise to antenna elements. For more details, see the `TransmitAntennaArray.size` property of the `nrCDLChannel` System object.
- Element pattern data tips include the directivity corresponding to any azimuth and zenith angles.
- Cluster path data tips include the average path gain and azimuth and zenith angles of the cluster path.

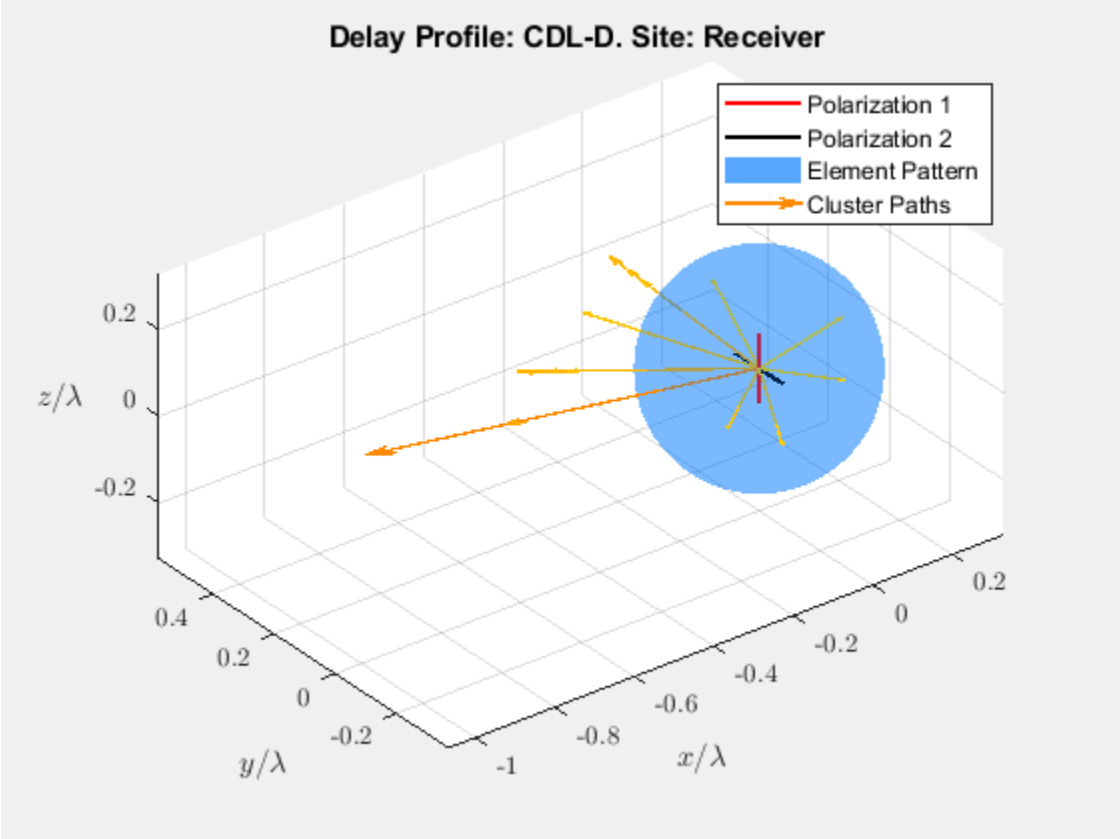


Visualize and explore channel characteristics at the receiver end. To customize the receive antenna array, use the “ReceiveAntennaArray” on page 2-0 property of the `nrCDLChannel` System object. Then, display the channel characteristics at the receiver end by calling the `displayChannel` function with the `'LinkEnd','Rx'` name-value pair argument.

```
figRx = displayChannel(cdl,'LinkEnd','Rx');
```

Explore channel information about the antenna element, element pattern, and cluster paths at the receiver end by enabling data cursor mode for the current figure.

```
datacursormode on;
```



Input Arguments

`cdl` — CDL channel model

`nrCDLChannel` System object™

CDL channel model, specified as an `nrCDLChannel` System object. The `nrCDLChannel` object implements the multi-input multi-output (MIMO) link-level fading channel specified in TR 38.901 Section 7.7.1 [1].

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'LinkEnd','Tx'` specifies visualization for the transmitter end only.

`LinkEnd` — Link-level channel end

`'Both'` (default) | `'Tx'` | `'Rx'`

Link-level channel end, specified as the comma-separated pair consisting of `'LinkEnd'` and one of these values:

- `'Both'` — The function displays channel characteristics at both ends: the transmitter and receiver ends.

- `'Tx'` — The function displays channel characteristics only at the transmitter end.
- `'Rx'` — The function displays channel characteristics only at the receiver end.

Data Types: `char` | `string`

`Polarization` — Polarization angle of antenna elements
`'on'` (default) | `'off'`

Polarization angle of antenna elements, specified as the comma-separated pair consisting of `'Polarization'` and `'on'` or `'off'`. When set to `'on'`, the function displays the polarization angle of the antenna elements.

Data Types: `char` | `string`

`ElementPattern` — Directivity radiation pattern of antenna elements
`'on'` (default) | `'off'`

Directivity radiation pattern of antenna, specified as the comma-separated pair consisting of `'ElementPattern'` and `'on'` or `'off'`. When set to `'on'`, the function displays the directivity radiation pattern of the antenna elements.

Note In the specified CDL channel model, `cdl`, the antenna element pattern is the same for all antenna elements. To orient the array with respect to the cluster paths, the function displays the element pattern centered at the first element of the array.

Data Types: `char` | `string`

`ClusterPaths` — Direction and average gain of cluster paths
`'on'` (default) | `'off'`

Direction and average gain of cluster paths, specified as the comma-separated pair consisting of `'ClusterPaths'` and `'on'` or `'off'`. When set to `'on'`, the function displays the direction and average gain of the cluster paths.

Note In the specified CDL channel model, `cdl`, the cluster path directions are the same for all antenna elements. To orient the array with respect to the cluster paths, the function displays the path directions centered at the first element of the array.

Data Types: `char` | `string`

Output Arguments

`fig` — Visualization windows
1-by-2 array of `figure` objects

Visualization windows, returned as a 1-by-2 array of `figure` objects.

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`info`

Objects
`nrCDLChannel`

Introduced in R2020b

generate

Generate next FTP, On-Off, or VoIP application traffic packet

Syntax

```
[dt,packetSize] = generate(cfg)
[dt,packetSize] = generate(cfg,elapsedTime)
[___,packet] = generate(___)
```

Description

`[dt,packetSize] = generate(cfg)` generates the next FTP, On-Off, or VoIP application traffic pattern based on the specified configuration object, `cfg`. The object function returns the time remaining to generate the next packet, `dt`, and the size of the current packet, `packetSize`.

`[dt,packetSize] = generate(cfg,elapsedTime)` specifies the time elapsed, `elapsedTime`, since the previous call of this object function.

`[___,packet] = generate(___)` returns the FTP, On-Off, or VoIP application traffic packet. Specify an argument combination from any of the previous syntaxes.

Examples

Generate VoIP Application Traffic Pattern Without Jitter

Create a VoIP application traffic pattern object, disabling modeling of jitter.

```
cfgVoIP = networkTrafficVoIP('HasJitter',false);
```

Generate a VoIP application traffic pattern.

```
[dt,packetSize] = generate(cfgVoIP);
```

Input Arguments

`cfg` — Configuration object to generate FTP, On-Off, or VoIP application traffic pattern

`networkTrafficFTP` object | `networkTrafficOnOff` object | `networkTrafficVoIP` object

Configuration object to generate FTP, On-Off, or VoIP application traffic pattern, specified as a `networkTrafficFTP`, `networkTrafficOnOff`, or `networkTrafficVoIP` object.

`elapsedTime` — Time elapsed since previous call of this object function

nonnegative scalar

Time elapsed since the previous call of this object function, specified as a nonnegative scalar.

Data Types: `double`

Output Arguments

`dt` — Time remaining to generate next packet

nonnegative scalar

Time remaining to generate the next packet, returned as a nonnegative scalar.

Data Types: `double`

`packetSize` — Size of current packet

positive scalar

Size of the current packet, returned as a positive scalar. The units of this argument are in bytes.

Data Types: `double`

`packet` — Application data packet

column vector of integers in the range [0, 255]

Application data packet, returned as a column vector of integers in the range [0, 255]. This value contains the application data specified by the `ApplicationData` property of the input `cfg`. If the `ApplicationData` property is not specified, `packet` is a column vector of 1s.

Dependencies

To enable this output argument, set the `GeneratePacket` property of the input `cfg` to `1` (`true`).

Data Types: `double`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`networkTrafficFTP` | `networkTrafficOnOff` | `networkTrafficVoIP`

Introduced in R2020b

getPathFilters

Get path filter impulse response for link-level MIMO fading channel

Syntax

```
pathFilters = getPathFilters(channel)
```

Description

`pathFilters = getPathFilters(channel)` returns path filter impulse responses for the link-level multi-input multi-output (MIMO) fading channel `channel`. Use `pathFilters` together with the `pathGains` output argument returned by the channel object to reconstruct a perfect channel estimate.

Examples

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 UE velocity of 15 km/h:

```
v = 15.0;                    % UE 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;     % UE 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.

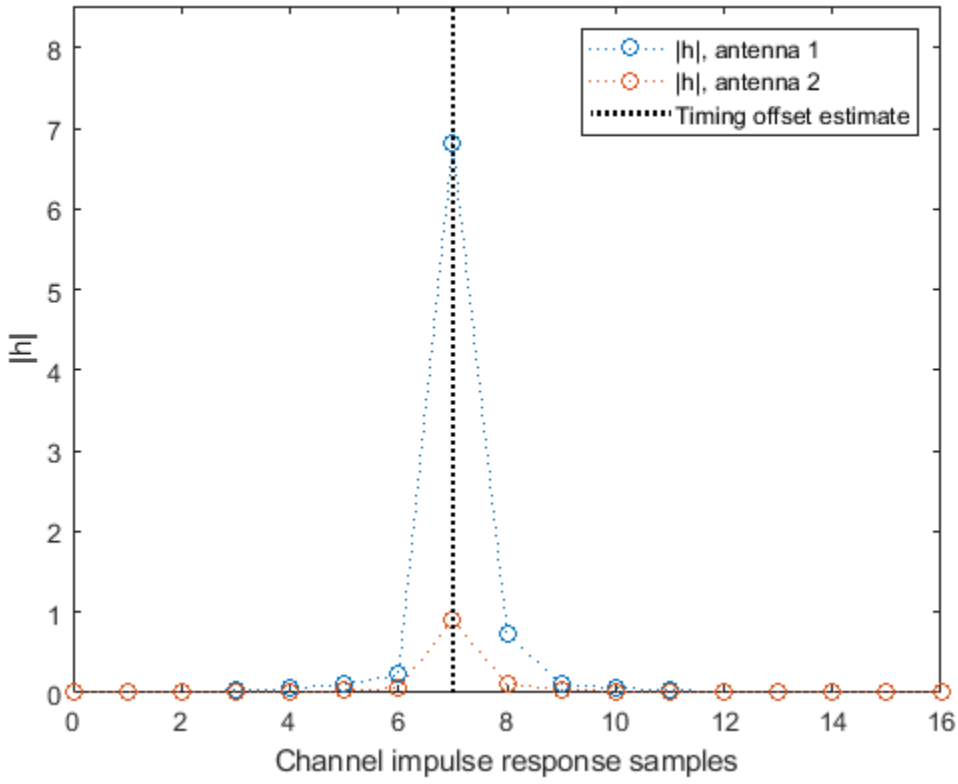
```
pathFilters = getPathFilters(cdl);
```

Perform timing offset estimation using `nrPerfectTimingEstimate`.

```
[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');
```



Input Arguments

`channel` — MIMO fading channel

`nrCDLChannel` | `nrTDLChannel`

MIMO fading channel, specified as an `nrCDLChannel` or `nrTDLChannel` System object. The objects implement the link-level MIMO fading channels specified in TR 38.901 Section 7.7.1 and Section 7.7.2, respectively.

Output Arguments

`pathFilters` — Path filter impulse response

N_h-by-N_p real matrix

Path filter impulse response, returned as an N_h-by-N_p real matrix, where:

- N_h is the number of impulse response samples.
- N_p is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`nrCDLChannel` | `nrTDLChannel`

Introduced in R2018b

getTransportBlock

Get transport block from UL-SCH or DL-SCH encoder

Syntax

```
trblk = getTransportBlock(encUL)
trblk = getTransportBlock(encDL,trblkID)
trblk = getTransportBlock( ___ ,harqID)
```

Description

`trblk = getTransportBlock(encUL)` returns the transport block from the specified uplink shared channel (UL-SCH) encoder System object. The function assumes that a transport block was previously loaded into the specified UL-SCH encoder by using the `setTransportBlock` function.

`trblk = getTransportBlock(encDL,trblkID)` returns the transport block from the specified downlink shared channel (DL-SCH) encoder System object `encDL` for the specified transport block number `trblkID`. The function assumes that a transport block was previously loaded into the specified DL-SCH encoder by using the `setTransportBlock` function.

`trblk = getTransportBlock( ___ ,harqID)` returns the transport block loaded for the specified hybrid automatic repeat-request (HARQ) process number `harqID`. Specify `harqID` in addition to the input arguments in any of the previous syntaxes.

Examples

Retrieve Transport Block from UL-SCH Encoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen1 = 5120;
trBlk1 = randi([0 1],trBlkLen1,1,'int8');
```

Create and configure an UL-SCH encoder System object with multiple HARQ processes and the specified target code rate.

```
targetCodeRate = 567/1024;
encUL = nrULSCH('MultipleHARQProcesses',true);
encUL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the UL-SCH encoder for HARQ process number 1.

```
setTransportBlock(encUL,trBlk1,1);
```

Call the encoder with QPSK modulation scheme, 1 transmission layer, an output length of 10,240 bits, redundancy version 0, and HARQ process number 1. The encoder applies the UL-SCH processing chain to the transport block loaded into the object using HARQ process number 1.

```
encUL('QPSK',1,10240,0,1);
```

Retrieve the transport block from the encoder for HARQ process number 1. Verify that the retrieved block is identical to the block originally loaded into the encoder for this HARQ process.

```
tmp = getTransportBlock(encUL,1);
isequal(tmp,trBlk1)
```

```
ans = logical
   1
```

Repeat the encoding operation for a new transport block of length 4400 and HARQ process number 2.

```
trBlkLen2 = 4400;
trBlk2 = randi([0 1],trBlkLen2,1,'int8');
setTransportBlock(encUL,trBlk2,2);
encUL('QPSK',1,8800,0,2);
```

Retrieve the first transport block again. Verify that the first transport block is still unchanged.

```
tmp = getTransportBlock(encUL,1);
isequal(tmp,trBlk1)
```

```
ans = logical
   1
```

Retrieve Transport Block from DL-SCH Encoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object with multiple HARQ processes and the specified target code rate.

```
targetCodeRate = 567/1024;
encDL = nrDLSCH('MultipleHARQProcesses',true);
encDL.TargetCodeRate = targetCodeRate;
```

Load transport block `trBlk` for transport block number 0 into the DL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
trBlkID = 0;
setTransportBlock(encDL,trBlk,trBlkID,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
```

```
rv = 3;
codedTrBlock = encDL(mod,nLayers,outlen,rv,harqID);
```

Retrieve the transport block for transport block number 0 from the encoder, specifying HARQ process number 2. Verify that the retrieved block is identical to the block originally loaded into the encoder for this HARQ process.

```
tmp = getTransportBlock(encDL,trBlkID,harqID);
isequal(tmp,trBlk)
```

```
ans = logical
   1
```

Input Arguments

`encUL` — UL-SCH encoder

`nrULSCH` System object

UL-SCH encoder, specified as an `nrULSCH` System object. The objects implements the UL-SCH processing chain specified in TS 38.212 Section 6.2.

`encDL` — DL-SCH encoder

`nrDLSCH` System object

DL-SCH encoder, specified as an `nrDLSCH` System object. The object implements the DL-SCH processing chain specified in TS 38.212 Section 7.2.

`trblkID` — Transport block number in DL-SCH processing

`0` (default) | `1`

Transport block number in DL-SCH processing, specified as `0` or `1`.

Data Types: `double`

`harqID` — HARQ process number

integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`trblk` — Transport block

binary column vector

Transport block, returned as a binary column vector.

Data Types: `int8`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`setTransportBlock`

Objects

`nrDLSCH` | `nrULSCH`

Introduced in R2019a

info

Get characteristic information about link-level MIMO fading channel

Syntax

```
channelInfo = info(channel)
```

Description

`channelInfo = info(channel)` returns characteristic information about the link-level multi-input multi-output (MIMO) fading channel `channel`.

Examples

Get Characteristic Information About TDL Fading Channel

Create an `nrTDLChannel` System object.

```
tdl = nrTDLChannel;
```

To get characteristic information about the channel, call the info function on the object.

```
channelInfo = info(tdl)
```

```
channelInfo = struct with fields:
          ChannelFilterDelay: 7
                  PathDelays: [1x23 double]
            AveragePathGains: [1x23 double]
             KFactorFirstTap: -Inf
          NumTransmitAntennas: 1
           NumReceiveAntennas: 2
    SpatialCorrelationMatrix: [2x2 double]
```

Input Arguments

`channel` — MIMO fading channel

`nrCDLChannel` | `nrTDLChannel`

MIMO fading channel, specified as an `nrCDLChannel` or `nrTDLChannel` System object. The objects implement the link-level MIMO fading channels specified in TR 38.901 Section 7.7.1 and Section 7.7.2, respectively.

Output Arguments

`channelInfo` — Characteristic information of channel model

structure

Characteristic information of channel model, returned as a structure. The fields of the structure depend on the input `channel`.

- If `channel` is an `nrCDLChannel` System object, the `channelInfo` structure has these fields.

Parameter Field	Value	Description
`PathDelays`	Numeric row vector	Delays of discrete channel paths for each cluster in seconds, returned as a numeric row vector. These values include the effects of `DelaySpread` scaling and `KFactorScaling` (when enabled).
`ClusterTypes`	Cell array of character vectors	Type of each cluster in the delay profile, returned as a cell array of character vectors. Cluster types can be `'LOS'`, `'SubclusteredNLOS'`, or `'NLOS'`.
`AveragePathGains`	Numeric row vector	Average path gains of the discrete path or clusters in dB, returned as a numeric row vector. These values include the effect of `KFactorScaling` scaling (when enabled).
`AnglesAoD`	Numeric row vector	Azimuth of departure angles of the clusters in degrees, returned as a numeric row vector. These values include the effect of angle scaling if enabled, see the `AngleSpreads` property.
`AnglesAoA`	Numeric row vector	Azimuth of arrival angles of the clusters in degrees, returned as a numeric row vector. These values include the effect of angle scaling if enabled, see the `AngleSpreads` property.
`AnglesZoD`	Numeric row vector	Zenith of departure angles of the clusters in degrees, returned as a numeric row vector. These values include the effect of angle scaling if enabled, see the `AngleSpreads` property.
`AnglesZoA`	Numeric row vector	Zenith of arrival angles of the clusters in degrees, returned as a numeric row vector. These values include the effect of angle scaling if enabled, see the `AngleSpreads` property.

Parameter Field	Value	Description
KFactorFirstCluster	Numeric scalar	K-factor of first cluster of delay profile in dB, returned as a numeric scalar. If the first cluster of the delay profile follows a Laplacian instead of a Rician distribution, `KFactorFirstCluster` is `-Inf`.
NumTransmitAntennas	Numeric scalar	Number of transmit antennas, returned as a numeric scalar.
NumReceiveAntennas	Numeric scalar	Number of receive antennas, returned as a numeric scalar.
ChannelFilterDelay	Numeric scalar	Channel filter delay in samples, returned as a numeric scalar.

Note

- The step of splitting the strongest clusters into subclusters, described in TR 38.901 Section 7.5, requires sorting of the clusters by their average power. If the `NumStrongestClusters` property is nonzero (applies only when `DelayProfile` is set to `'Custom'`), the fields of the information structure are sorted by average power. That is, the `AveragePathGains`, `ClusterTypes`, `PathDelays`, `AnglesAoD`, `AnglesAoA`, `AnglesZoD`, and `AnglesZoA` fields are presented in descending order of the average gain.
- If the `HasLOSCluster` property is set to `true`, the NLOS (Laplacian) part of that cluster and the LOS cluster are not necessarily next to each other. However, the `KFactorFirstCluster` field still indicates the appropriate K-factor.

- If `channel` is an `nrTDLChannel` System object, the `channelInfo` structure has the following fields.

Parameter Field	Value	Description
ChannelFilterDelay	Numeric scalar	Channel filter delay in samples, returned as a numeric scalar.
AveragePathGains	Numeric row vector	Average path gains of the discrete paths in dB, returned as a numeric row vector. These values include the effect of `KFactorScaling` (when enabled).
PathDelays	Numeric row vector	Delays of discrete channel paths in seconds, returned as a numeric row vector. These values include the effects of `DelaySpread` scaling and `KFactorScaling` (when enabled).

Parameter Field	Value	Description
KFactorFirstTap	Numeric scalar	K-factor of first tap of delay profile in dB, returned as a numeric scalar. If the first tap of the delay profile follows a Rayleigh instead of a Rician distribution, `KFactorFirstTap` is `-Inf`.
NumTransmitAntennas	Numeric scalar	Number of transmit antennas, returned as a numeric scalar.
NumReceiveAntennas	Numeric scalar	Number of receive antennas, returned as a numeric scalar.
SpacialCorrelationMatrix	Numeric matrix	Combined correlation matrix or 3-D array, returned as a numeric matrix.

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`nrCDLChannel` | `nrTDLChannel`

Introduced in R2018b

nrBCH

Broadcast channel (BCH) encoding

Syntax

```
cdblk = nrBCH(trblk,sfn,hrf,lssb,idxoffset,ncellid)
```

Description

`cdblk = nrBCH(trblk,sfn,hrf,lssb,idxoffset,ncellid)` encodes BCH transport block `trblk`, as defined in TS 38.212, Section 7.1 [1], and returns the encoded BCH transport block. The function takes these additional input arguments:

- `sfn`, the system frame number
- `hrf`, the half frame bit in synchronization signal / physical broadcast channel (SS/PBCH) block transmissions
- `lssb`, the number of candidate SS/PBCH blocks in a half frame
- `idxoffset`, the subcarrier offset or the SS block index, depending on the input value of `lssb`
- `ncellid`, the physical layer cell identity number

Examples

Encode BCH Transport Block

Generate a random sequence of binary values corresponding to a BCH transport block of 24 bits.

```
trblk = randi([0 1],24,1,'int8');
```

Specify the physical layer cell identity number as 321, the system frame number as 10, and the second half frame.

```
nid = 321;
sfn = 10;
hrf = 1;
```

Specify the number of candidate SS/PBCH blocks as 8. When you specify the number of candidate SS/PBCH blocks as 4 or 8, you can specify the subcarrier offset `kssb` as an input argument to the BCH encoder.

```
lssb = 8;
kssb = 18;
```

Encode the BCH transport block using the specified arguments.

```
cdblk = nrBCH(trblk,sfn,hrf,lssb,kssb,nid);
```

When you specify the number of candidate SS/PBCH blocks as 64, you can specify the SS block index `ssbIdx` as an input argument instead of the subcarrier offset `kssb`.

```
lssb = 64;
ssbIdx = 13;
```

Encode the BCH transport block with the updated input arguments.

```
cdblk2 = nrBCH(trblk,sfn,hrf,lssb,ssbIdx,nid);
```

Input Arguments

`trblk` — BCH transport block

24-by-1 binary column vector

BCH transport block, specified as a 24-by-1 binary column vector. The input `trblk` is the *BCCH-BCH-Message*, as defined in TS 38.331 Section 6.2.1 [2]. The *BCCH-BCH-Message* contains the master information block (MIB), as defined in TS 38.331 Section 6.2.2.

Data Types: `double` | `int8`

`sfn` — System frame number

nonnegative integer

System frame number, specified as a nonnegative integer.

Data Types: `double`

`hrf` — Half frame bit in SS/PBCH block transmissions

`0` | `1`

Half frame bit in SS/PBCH block transmissions, specified as `0` for the first half of a frame or `1` for the second half of a frame. For more information, see TS 38.214 Section 4.1 [3].

Data Types: `double`

`lssb` — Number of candidate SS/PBCH blocks

`4` | `8` | `64`

Number of candidate SS/PBCH blocks in a half frame, specified as `4`, `8`, or `64`.

Data Types: `double`

`idxoffset` — Subcarrier offset or SS block index

nonnegative integer

Subcarrier offset or SS block index, specified as a nonnegative integer.

- If `lssb` is `4` or `8`, `idxoffset` specifies the subcarrier offset, which must be an integer from 0 to 31.
- If `lssb` is `64`, `idxoffset` specifies the SS block index, which must be an integer from 0 to 63.

Data Types: `double`

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Output Arguments

`cdblk` — Encoded BCH transport block

864-by-1 binary column vector

Encoded BCH transport block, returned as an 864-by-1 binary column vector. `cdblk` inherits the data type of the input `trblk`.

Data Types: `double` | `int8`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrBCHDecode` | `nrPBCH` | `nrPBCHDecode`

Introduced in R2018b

nrBCHDecode

Broadcast channel (BCH) decoding

Syntax

```
scrblk = nrBCHDecode(softbits,L)
[scrblk,errFlag] = nrBCHDecode(softbits,L)
[scrblk,errFlag,trblk,lsbofsfn,hrf,msbidxoffset] = nrBCHDecode(softbits,L,
lssb,ncellid)
```

Description

`scrblk = nrBCHDecode(softbits,L)` decodes the log-likelihood ratios (LLRs) `softbits` in accordance with TS 38.212, Section 7.1 [1]. The function returns the decoded scrambled BCH transport block `scrblk`. The input argument `L` is the list length used for polar decoding.

`[scrblk,errFlag] = nrBCHDecode(softbits,L)` also returns an error flag, `errFlag`, to indicate whether `scrblk` contains an error after decoding.

`[scrblk,errFlag,trblk,lsbofsfn,hrf,msbidxoffset] = nrBCHDecode(softbits,L,lssb,ncellid)` also returns the decoded and unscrambled BCH transport block `trblk`. The additional input arguments are the number of candidate synchronization signal / physical broadcast channel (SS/PBCH) blocks, `lssb`, and the physical layer cell identity number, `ncellid`. The function also returns these information elements:

- `lsbofsfn`, the four least significant bits (LSBs) of the system frame number
- `hrf`, the half frame bit
- `msbidxoffset`, the most significant bits (MSBs) of the index offset

Examples

Decode Scrambled BCH Transport Block

Generate a random sequence of binary values corresponding to a BCH transport block of 24 bits.

```
trblk = randi([0 1],24,1,'int8');
```

Specify the physical layer cell identity number as 321, the system frame number as 10, and the second half frame.

```
nid = 321;
sfn = 10;
hrf = 1;
```

Specify the number of candidate SS/PBCH blocks as 8. When you specify the number of candidate SS/PBCH blocks as 4 or 8, you can specify the subcarrier offset `kssb` as an input argument to the BCH encoder.

```
lssb = 8;
kssb = 18;
```

Encode the BCH transport block using the specified arguments.

```
bch = nrBCH(trblk,sfn,hrf,lssb,kssb,nid);
```

Decode the encoded transport block and recover information by using a polar decoding list length of 8 bits.

```
listLen = 8;
[~,errFlag,rxtrblk,rxSFN4lsb,rxHRF,rxKssb] = nrBCHDecode( ...
   double(1-2*bch),listLen,lssb,nid);
```

Verify that the decoding has no errors.

```
errFlag
```

```
errFlag = uint32
    0
```

```
isequal(trblk,rxtrblk)
```

```
ans = logical
   1
```

```
isequal(bi2de(rxSFN4lsb','left-msb'),mod(sfn,16))
```

```
ans = logical
   1
```

```
[isequal(hrf,rxHRF) isequal(de2bi(floor(kssb/16),1),rxKssb)]
```

```
ans = 1x2 logical array

   1   1
```

Input Arguments

`softbits` — Approximate log-likelihood ratio (LLR) soft bits

864-by-1 real-valued column vector

Approximate log-likelihood ratio (LLR) soft bits, specified as an 864-by-1 real-valued column vector.

Data Types: `single` | `double`

`L` — Polar decoding list length

power of 2

Polar decoding list length, specified as a power of 2.

Data Types: `double`

`lssb` — Number of candidate SS/PBCH blocks in a half frame

4 | 8 | 64

Number of candidate SS/PBCH blocks in a half frame, specified as `4`, `8`, or `64`.

Data Types: `double`

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Output Arguments

`scrblk` — Decoded scrambled BCH transport block
32-by-1 binary column vector

Decoded scrambled BCH transport block, returned as a 32-by-1 binary column vector.

Data Types: `int8`

`errFlag` — Error flag
`0` | `1`

Error flag to indicate whether `scrblk` contains an error, returned as `0` or `1`. If `errFlag` is `1`, then an error has occurred.

Data Types: `uint32`

`trblk` — Decoded and unscrambled BCH transport block
24-by-1 binary column vector

Decoded and unscrambled BCH transport block, returned as a 24-by-1 binary column vector. The output `trblk` is the *BCCH-BCH-Message*, as defined in TS 38.331 Section 6.2.1 [2]. The *BCCH-BCH-Message* contains the master information block (MIB), as defined in TS 38.331 Section 6.2.2.

Data Types: `logical`

`lsbofsfn` — LSBs of the system frame number
4-by-1 column vector

The four LSBs of the system frame number, returned as a 4-by-1 column vector.

Data Types: `logical`

`hrf` — Half frame bit in SS/PBCH block transmissions
`0` | `1`

Half frame bit in SS/PBCH block transmissions, returned as `0` indicating the first half of a frame or `1` indicating the second half of a frame. For more information, see TS 38.214 Section 4.1 [3].

Data Types: `logical`

`msbidxoffset` — MSBs of index offset
scalar | 3-by-1 column vector

MSBs of index offset, returned as a scalar or 3-by-1 column vector.

- If `lssb` is `4` or `8`, `msbidxoffset` is the decoded MSB of the subcarrier index, returned as a scalar.
- If `lssb` is `64`, the entries of `msbidxoffset` are the three decoded MSBs of the SSB index, returned as a 3-by-1 column vector.

Data Types: `logical`

Compatibility Considerations

Polar decoding metric update
Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrBCHDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The input argument `L` must be a compile-time constant. Include `{coder.Constant(L)}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrBCH` | `nrPBCH` | `nrPBCHDecode`

Introduced in R2018b

nrChannelEstimate

Practical channel estimation

Syntax

```
[h,nVar,info] = nrChannelEstimate(rxGrid,refInd,refSym)
[h,nVar,info] = nrChannelEstimate(rxGrid,refGrid)
[h,nVar,info] = nrChannelEstimate(carrier,___)
[h,nVar,info] = nrChannelEstimate(___,Name,Value)
```

Description

`[h,nVar,info] = nrChannelEstimate(rxGrid,refInd,refSym)` performs practical channel estimation on the received resource grid `rxGrid` by using a reference resource grid containing reference symbols `refSym` at locations `refInd`. The function returns the channel estimate `h`, noise variance estimate `nVar`, and additional information `info` .

`[h,nVar,info] = nrChannelEstimate(rxGrid,refGrid)` specifies a predefined reference resource grid `refGrid`.

`[h,nVar,info] = nrChannelEstimate(carrier,___)` specifies carrier configuration parameters for a specific orthogonal frequency-division multiplexing (OFDM) numerology, in addition to the input arguments from any of the previous syntaxes. The function uses only the `CyclicPrefix` property of the `carrier` input.

`[h,nVar,info] = nrChannelEstimate(___,Name,Value)` specifies options by using one or more name-value pair arguments in addition to the input arguments in any of the previous syntaxes.

Examples

Compare Practical and Perfect Channel Estimates

Generate physical broadcast channel (PBCH) demodulation reference signal (DM-RS) symbols for physical layer cell identity number 42. The time-dependent part of the DM-RS scrambling initialization is 0.

```
ncellid = 42;
ibar_SSB = 0;
dmrsSym = nrPBCHDMRS(ncellid,ibar_SSB);
```

Obtain resource element indices for the PBCH DM-RS.

```
dmrsInd = nrPBCHDMRSIndices(ncellid);
```

Create a resource grid containing the generated DM-RS symbols.

```
nrb = 20;
scs = 15;
carrier = nrCarrierConfig('NSizeGrid',nrb,'SubcarrierSpacing',scs);
nTxAnts = 1;
```

```
txGrid = nrResourceGrid(carrier,nTxAnts);
txGrid(dmrsInd) = dmrsSym;
```

Modulate the resource grid using the specified FFT length and cyclic prefix length.

```
ofdmInfo = nrOFDMInfo(carrier);
nulls = [1:136 377:512].';
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Create a TDL-C channel model with the specified properties.

```
channel = nrTDLChannel;
channel.NumReceiveAntennas = 1;
channel.SampleRate = ofdmInfo.SampleRate;
channel.DelayProfile = 'TDL-C';
channel.DelaySpread = 100e-9;
channel.MaximumDopplerShift = 20;
```

Obtain the maximum number of delayed samples from the channel path by using the largest delay and the implementation delay of the channel filter.

```
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;
```

To flush delayed samples from the channel, append zeros at the end of the transmitted waveform corresponding to the maximum number of delayed samples and the number of transmit antennas. Transmit the padded waveform through the TDL-C channel model.

```
[rxWaveform,pathGains] = channel([txWaveform; zeros(maxChDelay,nTxAnts)]);
```

Estimate timing offset for the transmission using the DM-RS symbols as reference symbols. The OFDM modulation of the reference symbols uses an initial slot number of 0.

```
initialSlot = 0;
offset = nrTimingEstimate(carrier,rxWaveform,txGrid);
```

Synchronize the received waveform according to the estimated timing offset.

```
rxWaveform = rxWaveform(1+offset:end,:);
```

Create a received resource grid containing the demodulated and synchronized received waveform.

```
cpFraction = 0.55;
rxGrid = nrOFDMDemodulate(carrier,rxWaveform,'CyclicPrefixFraction',cpFraction);
```

Obtain the practical channel estimate.

```
H = nrChannelEstimate(rxGrid,dmrsInd,dmrsSym);
```

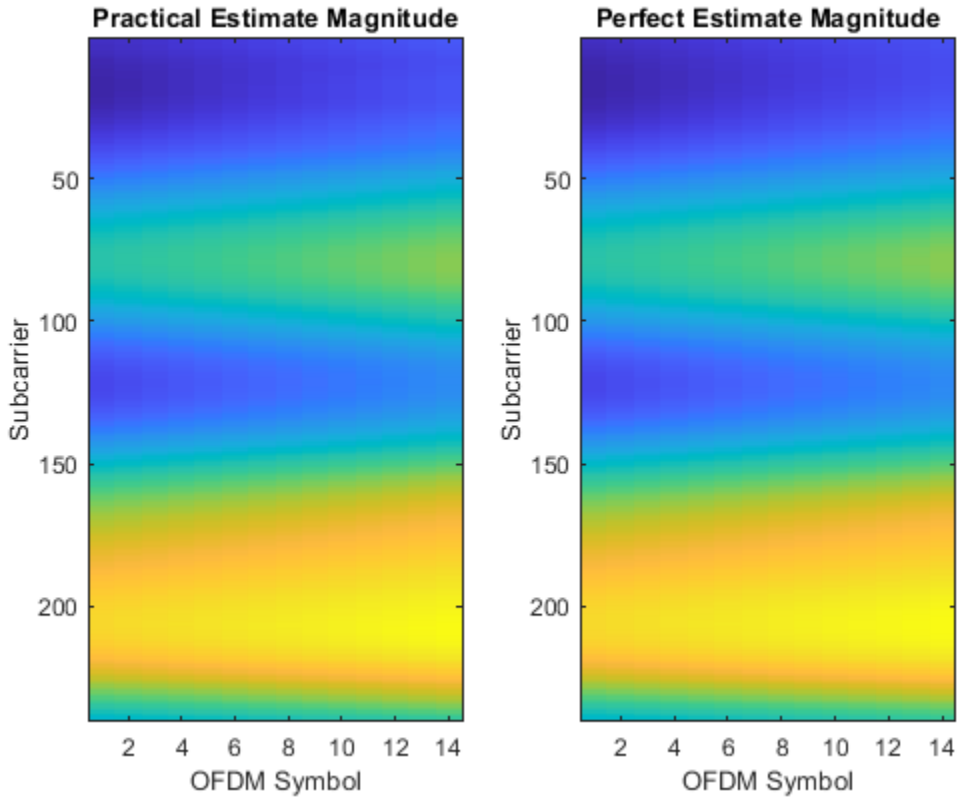
Obtain the perfect channel estimate.

```
pathFilters = getPathFilters(channel);
H_ideal = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset);
```

Compare practical and perfect channel estimates.

```
figure;
subplot(1,2,1);
imagesc(abs(H));
```

```
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title('Practical Estimate Magnitude');
subplot(1,2,2);
imagesc(abs(H_ideal));
xlabel('OFDM Symbol');
ylabel('Subcarrier');
title('Perfect Estimate Magnitude');
```



Input Arguments

`rxGrid` — Received resource grid

K-by-*L*-by-*R* complex array

Received resource grid, specified as a *K*-by-*L*-by-*R* complex array.

- *K* is the number of subcarriers equal to *NRB* × 12, where *NRB* is the number of resource blocks in the range from 1 to 275.
- *L* is the number of OFDM symbols in a slot or in a reference grid.
  - When you call `nrChannelEstimate` with reference symbols `refSym`, *L* is 12 for extended cyclic prefix and 14 for normal cyclic prefix. Set the cyclic prefix length by using the `'CyclicPrefix'` name-value pair argument.
  - When you call `nrChannelEstimate` with reference resource grid `refGrid`, *L* must equal *N*, the number of OFDM symbols in the reference grid.

- *R* is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`refInd` — Reference symbol indices

integer matrix

Reference symbol indices, specified as an integer matrix. The number of rows equals the number of resource elements. You can specify all indices in a single column or spread them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

The elements of `refInd` are one-based linear indices addressing a *K*-by-*L*-by-*P* resource array.

- *K* is the number of subcarriers equal to *NRB* × 12, where *NRB* is the number of resource blocks in the range from 1 to 275. *K* must be equal to the first dimension of `rxGrid`.
- *L* is the number of OFDM symbols in a slot. *L* is 12 for extended cyclic prefix and 14 for normal cyclic prefix. Set the cyclic prefix length by using the `'CyclicPrefix'` name-value pair argument.
- *P* is the number of reference signal ports, inferred from the range of values in `refInd`.

Data Types: `double`

`refSym` — Reference symbols

complex matrix

Reference symbols, specified as a complex matrix. The number of rows equals the number of resource elements. You can specify all symbols in a single column or distribute them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

Data Types: `single` | `double`
Complex Number Support: Yes

`refGrid` — Predefined reference grid

K-by-*N*-by-*P* complex array

Predefined reference grid, specified as a *K*-by-*N*-by-*P* complex array. `refGrid` can span multiple slots.

- *K* is the number of subcarriers equal to *NRB* × 12, where *NRB* is the number of resource blocks in the range from 1 to 275.
- *N* is the number of OFDM symbols in the reference grid.
- *P* is the number of reference signal ports.

Data Types: `single` | `double`
Complex Number Support: Yes

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only the `CyclicPrefix` property of this input.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'CyclicPrefix','extended'` specifies extended cyclic prefix length.

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as the comma-separated pair consisting of `'CyclicPrefix'` and one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

If you specify the `carrier` input and this input, the function uses the value specified in the `CyclicPrefix` property of the `carrier` input as the cyclic prefix length.

Data Types: `char` | `string`

`CDMLengths` — CDM arrangement for reference signals

`[1 1]` (default) | 1-by-2 array of nonnegative integers

Code domain multiplexing (CDM) arrangement for reference signals, specified as the comma-separated pair consisting of `'CDMLengths'` and a 1-by-2 array of nonnegative integers [*FD TD*]. Array elements *FD* and *TD* specify the length of CDM despreading in the frequency domain (FD-CDM) and time domain (TD-CDM), respectively. A value of 1 for an element specifies no CDM.

Example: `'CDMLengths',[2 1]` specifies FD-CDM2 and no TD-CDM.

Example: `'CDMLengths',[1 1]` specifies no orthogonal despreading.

Data Types: `double`

`AveragingWindow` — Pre-interpolation averaging window

`[0 0]` (default) | 1-by-2 array of nonnegative odd integers

Pre-interpolation averaging window, specified as the comma-separated pair consisting of `'AveragingWindow'` and a 1-by-2 array of nonnegative odd integers [*F T*]. Array elements *F* and *T* specify the number of adjacent reference symbols in the frequency domain and time domain, respectively, over which the function performs averaging before interpolation. If *F* or *T* is zero, the function determines the averaging value from the estimated signal-to-noise ratio (SNR) based on the noise variance estimate `nVar`.

Data Types: `double`

Output Arguments

`h` — Practical channel estimate

K-by-*L*-by-*R*-by-*P* complex array

Practical channel estimate, returned as a *K*-by-*L*-by-*R*-by-*P* complex array. *K*-by-*L*-by-*R* is the shape of the received resource grid `rxGrid`. *P* is the number of reference signal ports.

`h` inherits its data type from `rxGrid`.

Data Types: `double` | `single`

`nVar` — Noise variance estimate

nonnegative scalar

Noise variance estimate, returned as a nonnegative scalar. `nVar` is the measured variance of additive white Gaussian noise on the received reference symbols.

Data Types: `double`

`info` — Additional information

structure

Additional information, returned as a structure with the field `AveragingWindow`.

Parameter Field	Value	Description
`AveragingWindow`	1-by-2 array	Pre-interpolation averaging window, returned as a 1-by-2 array [*F T*]. Array elements *F* and *T* indicate the number of adjacent reference symbols in the frequency domain and time domain, respectively, over which the function performed averaging before interpolation.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPerfectChannelEstimate` | `nrPerfectTimingEstimate` | `nrTimingEstimate`

Objects

`nrCarrierConfig`

Introduced in R2019b

nrCodeBlockDesegmentLDPC

LDPC code block desegmentation and CRC decoding

Syntax

```
[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen)
```

Description

`[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen)` concatenates the input code block segments `cbs` into a single output data block `blk` of length `blklen`. The function validates the data dimensions of the input `cbs` based on the specified base graph number `bgn` and output block length `blklen`. The function removes any filler bits and type-24B cyclic redundancy check (CRC) bits present in the input `cbs`. The output `err` is the result of the type-24B CRC decoding (if applicable). This process is the inverse of the low-density parity-check (LDPC) code block segmentation specified in TS 38.212 Section 5.2.2 [1] and implemented in `nrCodeBlockSegmentLDPC`.

Examples

Back-to-Back LDPC Code Block Segmentation and Desegmentation

Perform code block segmentation of a random sequence of binary input data.

```
bgn = 1;
blklen = 10000;
cbs = nrCodeBlockSegmentLDPC(randi([0 1],blklen,1),bgn);
```

When the base graph number is 1, segmentation occurs whenever the input length is greater than 8448. The input data of length 10000 is split into two code block segments of length 5280. The code block segments have filler bits and CRC attached.

```
size(cbs)
```

```
ans = 1×2

        5280           2
```

Concatenate the code block segments.

```
[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen);
```

The concatenated result is of the same size as the original input with CRC and filler bits removed.

```
blkSize = size(blk)
```

```
blkSize = 1×2

       10000           1
```

Verify if the CRC decoding was successful by checking the error vector.

```
err
```

```
err = 1x2 uint32 row vector

   0   0
```

Display Index Mapping of LDPC Code Block Desegmentation

Create a matrix representing two code block segments. Each element contains the linear index of that element within the matrix.

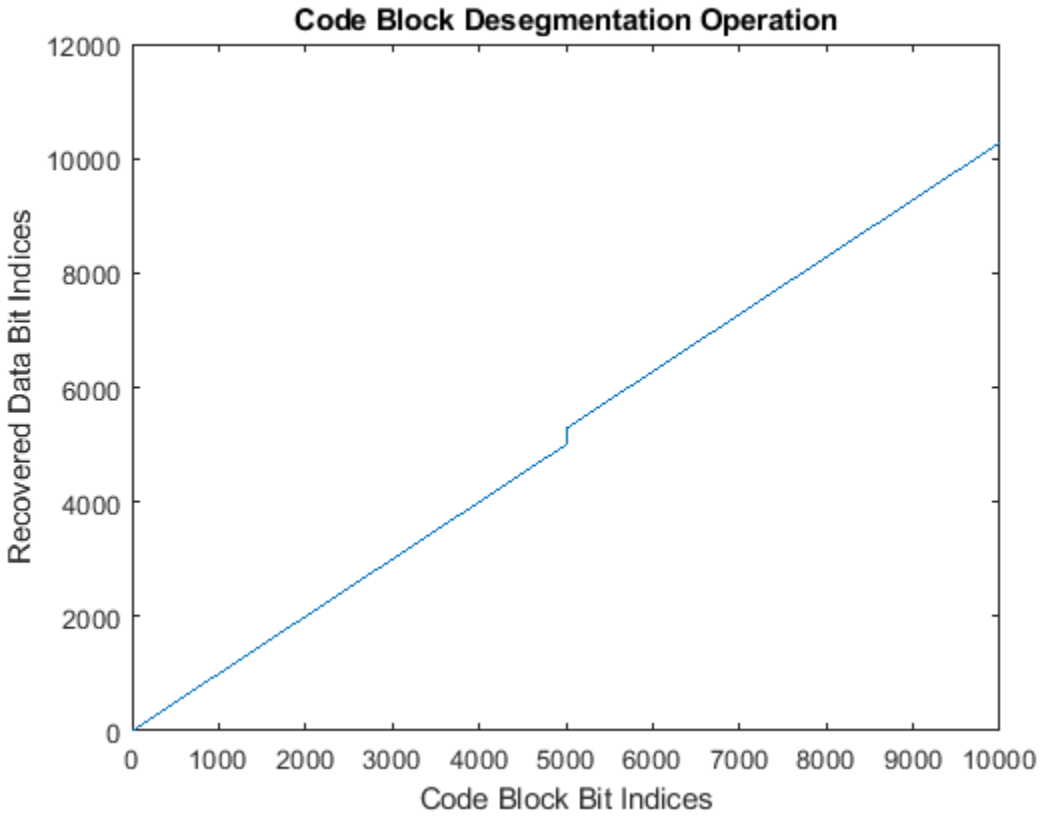
```
cbs = reshape([1:10560]',[],2);
```

Concatenate the code block segments using the specified base graph number and output block length.

```
bgn = 1;
blklen = 10000;
blk = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen);
```

To see how the input maps onto the output, plot code block segment indices relative to the corresponding indices in the concatenated input. In each code block segment, the last 280 bits represent CRC and filler bits. These additional bits are removed from the recovered data.

```
plot(blk);
xlabel('Code Block Bit Indices');
ylabel('Recovered Data Bit Indices');
title('Code Block Desegmentation Operation');
```



Input Arguments

`cbs` — Code block segments

real matrix

Code block segments, specified as a real matrix. A matrix with only one column corresponds to one code block segment without CRC bits appended. If you specify a matrix with more than one column, each column in the matrix corresponds to a separate code block segment with type-24B CRC bits appended. In both cases, the code block segments can contain filler bits.

Data Types: `double` | `int8`

`bgn` — Base graph number

`1` | `2`

Base graph number, specified as `1` or `2`.

Data Types: `double`

`blklen` — Output block length

nonnegative integer

Output block length, specified as a nonnegative integer. If `blklen` is `0`, then both `blk` and `err` are empty. The function uses `blklen` to validate the data dimensions of the input `cbs` and to calculate the number of filler bits to remove.

Data Types: `double`

Output Arguments

`blk` — Concatenated data block

empty vector | real column vector

Concatenated data block, returned as an empty vector (when `blklen` is `0`) or a real column vector. The function removes any filler bits and type-24B CRC bits present in the input `cbs`. The output `blk` inherits its data type from the input `cbs`.

Data Types: `double` | `int8`

`err` — CRC error

empty vector | vector of nonnegative integers

CRC error, returned as one of these values:

- Empty vector — The function returns this value when `blklen` is `0` or if `cbs` has only one column (CRC decoding does not take place).
- Vector of nonnegative integers — If `cbs` has more than one column, `err` contains the CRC error bits obtained from decoding the type-24B CRC bits in each code block segment. The length of `err` is equal to the number of code block segments (number of columns in the input `cbs`).

Data Types: `uint32`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCRCDecode` | `nrCodeBlockSegmentLDPC` | `nrLDPCDecode` | `nrRateRecoverLDPC`

Introduced in R2018b

nrCodeBlockSegmentLDPC

LDPC code block segmentation and CRC attachment

Syntax

```
cbs = nrCodeBlockSegmentLDPC(blk,bgn)
```

Description

`cbs = nrCodeBlockSegmentLDPC(blk,bgn)` splits the input data block `blk` into code block segments based on the base graph number `bgn`, as specified in TS 38.212 Section 5.2.2 [1]. The function appends cyclic redundancy check (CRC) and filler bits to each code block segment in `cbs` (if applicable). `nrCodeBlockSegmentLDPC` provides input to low-density parity-check (LDPC) coders in transport channels, including downlink and uplink shared channels, and paging channels.

Examples

LDPC Code Block Segmentation

Create a random sequence of binary input data.

```
in = randi([0,1],4000,1);
```

Perform LDPC code block segmentation.

```
cbs1 = nrCodeBlockSegmentLDPC(in,1);
cbs2 = nrCodeBlockSegmentLDPC(in,2);
```

When the base graph number is 1, the segmentation results in one code block segment. When the base graph number is 2, the segmentation results in two code block segments. Segmentation occurs only if the input length is greater than the maximum code block size. The maximum code block size is 8448 when the base graph number is 1 and 3840 when the base graph number is 2.

```
size(cbs1)
```

```
ans = 1×2

        4224           1
```

```
size(cbs2)
```

```
ans = 1×2

        2080           2
```

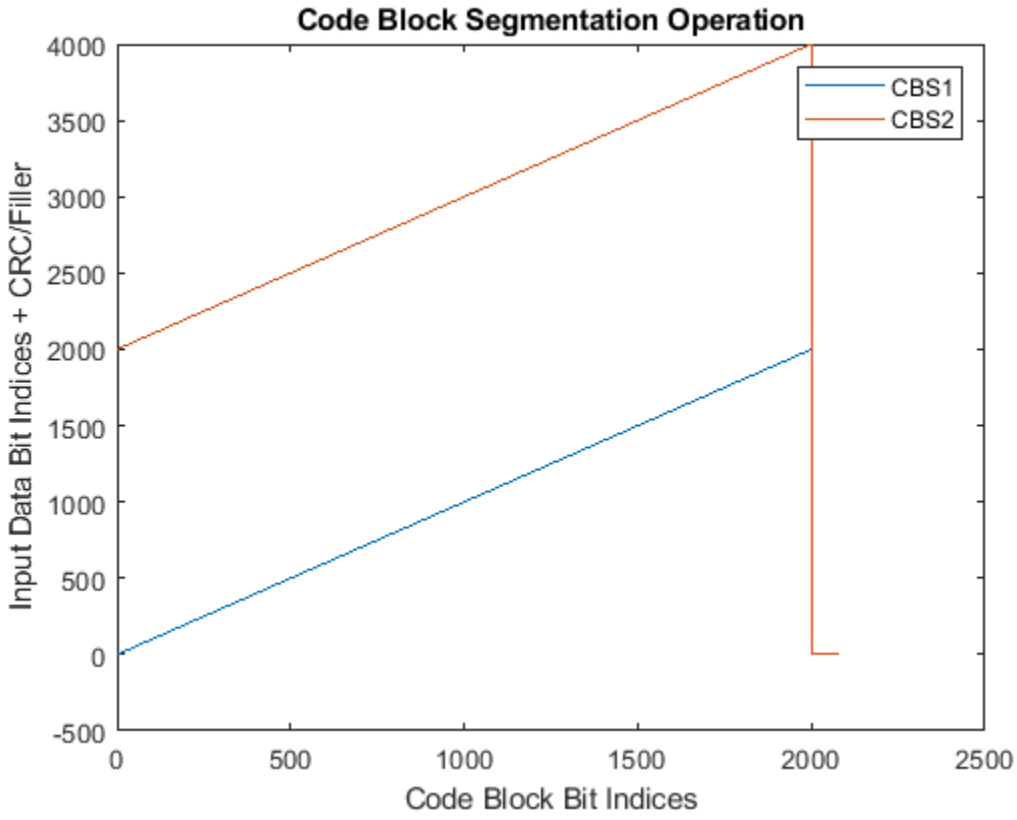
Display Index Mapping of LDPC Code Block Segmentation

Create a ramp data input and perform code block segmentation.

```
cbs = nrCodeBlockSegmentLDPC([1:4000]',2);
```

The input of length 4000 is split into two code block segments of equal size with 24B CRC and filler bits attached. To see how the input maps onto the output, plot the input data indices relative to the corresponding code block segment indices.

```
plot(cbs)
legend('CBS1','CBS2')
xlabel('Code Block Bit Indices');
ylabel('Input Data Bit Indices + CRC/Filler');
title('Code Block Segmentation Operation')
```



Input Arguments

`blk` — Input data block

column vector of real numbers

Input data block, specified as a column vector of real numbers.

Data Types: `double` | `int8` | `logical`

bgn — Base graph number

1 | 2

Base graph number, specified as 1 or 2.

Data Types: double

Output Arguments

cbs — Code block segments

integer or real matrix

Code block segments, returned as an integer or real matrix. Each column corresponds to a separate code block segment. The number of code block segments depends on the maximum code block size of the LDPC coder, Kcb, and the length of the input blk, B. If bgn is set to 1, Kcb = 8448. If bgn is set to 2, Kcb = 3840. If $B \le Kcb$, then the function does not perform segmentation and does not append CRC to the resulting code block. If $B > Kcb$, the segmentation results in several smaller code blocks with a type-24B CRC bits appended.

The function appends filler bits to each code block (with or without CRC) if necessary. The filler bits ensure that the code block segments entering the LDPC coder have a valid length and are a multiple of the LDPC lifting size. To accommodate the filler bits represented by –1, the data type of cbs is cast to int8 when the input blk is logical. Otherwise, cbs inherits the data type of the input blk.

Data Types: double | int8

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

nrCodeBlockDesegmentLDPC | nrLDPCEncode | nrLDPCEncode | nrRateMatchLDPC | nrRateMatchLDPC

Introduced in R2018b

nrCRCDecode

Decode and remove cyclic redundancy check (CRC)

Syntax

```
[blk,err] = nrCRCDecode(blkcrc,poly)
[blk,err] = nrCRCDecode(blkcrc,poly,mask)
```

Description

`[blk,err] = nrCRCDecode(blkcrc,poly)` checks the input data `blkcrc` for a CRC error. The function assumes that the input data comprises the CRC parity bits associated with the polynomial `poly`. The function returns `blk`, which is the data part of the input `blkcrc`. The function also returns `err`, which is the logical difference (XOR) between the CRC comprised in the input and the CRC recalculated across the data part of the input. If `err` is not equal to 0, either an error has occurred or the input CRC has been masked. For details on the associated polynomials, see TS 38.212 Section 5.1 [1].

`[blk,err] = nrCRCDecode(blkcrc,poly,mask)` XOR-masks the CRC difference with `mask` before returning it in `err`. The `mask` value is applied to the CRC difference with the most significant bit (MSB) first to the least significant bit (LSB) last.

Examples

Check Data Block for CRC Error

Check the effect of CRC decoding with and without a mask.

Define a mask corresponding to the radio network temporary identifier (RNTI) equal to 12. Append RNTI-masked CRC parity bits to an all-ones matrix of one data block.

```
rnti = 12;
blkCrc = nrCRCEncode(ones(100,1),'24C',rnti);
```

When you perform CRC decoding without a mask, `err1` is equal to the RNTI because the CRC was masked during coding. The logical difference between the original CRC and the recalculated CRC is the CRC mask.

```
[blk,err1] = nrCRCDecode(blkCrc,'24C');
err1
```

```
err1 = uint32
    12
```

When you perform CRC decoding using the RNTI value as a mask, `err` is equal to 0.

```
[blk,err2] = nrCRCDecode(blkCrc,'24C',err1);
err2
```

```
err2 = uint32
    0
```

Input Arguments

`blkcrc` — CRC encoded data

matrix of real numbers

CRC encoded data, specified as a matrix of real numbers. Each column of the matrix is considered as a separate CRC encoded data block.

Data Types: `double` | `int8` | `logical`

`poly` — CRC polynomial

`'6'` | `'11'` | `'16'` | `'24A'` | `'24B'` | `'24C'`

CRC polynomial, specified as `'6'`, `'11'`, `'16'`, `'24A'`, `'24B'`, or `'24C'`. For details on the associated polynomials, see TS 38.212 Section 5.1.

Data Types: `char` | `string`

`mask` — XOR mask

`0` (default) | nonnegative integer

XOR mask, specified as a nonnegative integer. The mask is typically a radio network temporary identifier (RNTI).

Data Types: `double`

Output Arguments

`blk` — CRC decoded data

matrix of real numbers

CRC decoded data, returned as a matrix of real numbers. `blk` is the data-only part of the input `blkcrc`.

Data Types: `double` | `int8` | `logical`

`err` — Logical CRC difference

integer

Logical CRC difference, returned as an integer. `err` is the logical difference between the CRC comprised in the input `blkcrc` and the CRC recalculated across the data part of the input. If a mask is specified, the function XOR-masks `err` with `mask` before returning it.

Data Types: `uint32`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

nrBCHDecode | nrCRCEncode | nrCodeBlockDesegmentLDPC | nrDCIDecode | nrLDPCDecode | nrPolarDecode | nrRateRecoverLDPC | nrRateRecoverPolar

Introduced in R2018b

nrCRCEncode

Calculate and append cyclic redundancy check (CRC)

Syntax

```
blkcrc = nrCRCEncode(blk,poly)
blkcrc = nrCRCEncode(blk,poly,mask)
```

Description

`blkcrc = nrCRCEncode(blk,poly)` calculates the CRC defined by the polynomial `poly` for the input data `blk`. The function returns the CRC encoded data, which is a copy of the input data with the CRC parity bits appended. For details on the associated polynomials, see TS 38.212 Section 5.1 [1].

`blkcrc = nrCRCEncode(blk,poly,mask)` applies a logical difference (XOR) mask on the appended CRC bits with the integral value of `mask`. The appended CRC bits in `blkcrc` are XOR-masked with the most significant bit (MSB) first to the least significant bit (LSB) last. The masked CRC is of the form (p_0 xor m_0), (p_1 xor m_1),..., (p_{L-1} xor m_{L-1}), where L is the number of parity bits, and p_0 and m_0 are the MSBs in the binary representation of CRC and `mask`, respectively. If the mask value is greater than 2^L - 1, the L LSBs are considered for the mask.

Examples

Calculate and Append CRC

Calculate and append CRC parity bits to an all-zeros matrix of two data blocks. The result is an all-zeros matrix of size 124-by-2.

```
blkcrc = nrCRCEncode(zeros(100,2),'24C');
any(blkcrc(:,1:2));
```

Calculate and Append Masked CRC

Calculate and append masked CRC parity bits to an all-zeros matrix of two data blocks. The appended CRC bits are XOR-masked with the specified `mask`, from the MSB first to the LSB last. The result is an all-zeros matrix apart from the elements in the last position.

```
mask = 1;
blkcrc = nrCRCEncode(zeros(100,2),'24C',mask);
blkcrc(end-5:end,1:2)
```

```
ans = 6×2

     0     0
     0     0
     0     0
     0     0
```

```
     0     0
     1     1
```

Input Arguments

`blk` — Input data

matrix of real numbers

Input data, specified as a matrix of real numbers. Each column of the matrix is treated as a separate data block.

Data Types: `double` | `int8` | `logical`

`poly` — CRC polynomial

`'6'` | `'11'` | `'16'` | `'24A'` | `'24B'` | `'24C'`

CRC polynomial, specified as `'6'`, `'11'`, `'16'`, `'24A'`, `'24B'`, or `'24C'`. For details on the associated polynomials, see TS 38.212 Section 5.1.

Data Types: `char` | `string`

`mask` — XOR mask

`0` (default) | nonnegative integer

XOR mask, specified as a nonnegative integer. The mask is typically a radio network temporary identifier (RNTI).

Data Types: `double`

Output Arguments

`blkcrc` — CRC encoded data

matrix of real numbers

CRC encoded data, returned as a matrix of real numbers. `blkcrc` is a copy of the input `blk` with the CRC parity bits appended. Each column corresponds to a separate CRC encoded code block. `blkcrc` inherits its data type from the input `blk`.

Data Types: `double` | `int8` | `logical`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrBCH | nrCRCDecode | nrCodeBlockSegmentLDPC | nrDCIEncode | nrLDPCEncode | nrPolarEncode | nrRateMatchLDPC | nrRateMatchPolar

Introduced in R2018b

nrCSIRS

Generate CSI-RS symbols

Syntax

```
[sym,info] = nrCSIRS(carrier,csirs)
[sym,info] = nrCSIRS(carrier,csirs,Name,Value)
```

Description

`[sym,info] = nrCSIRS(carrier,csirs)` returns channel state information reference signal (CSI-RS) symbols `sym`, as defined in TS 38.211 Section 7.4.1.5 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `csirs` specifies CSI-RS resource configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources. When configuring both ZP and NZP resources, the returned symbols are in the order ZP followed by NZP, irrespective of the resource order specified by `csirs`. The function also returns the structure `info`, which contains additional information about the CSI-RS locations.

`[sym,info] = nrCSIRS(carrier,csirs,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate CSI-RS Symbols and Indices for 10 MHz Carrier

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object with default properties.

```
csirs = nrCSIRSConfig;
```

Generate CSI-RS symbols of `single` data type.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,'OutputDataType','single');
```

Generate resource element indices for CSI-RS.

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs);
```

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers 3 and 5, symbol locations 13 and 9, and subcarrier locations 6 and 4, respectively. For both resources, set the periodicity to 5, offset to 1, and density to `'one'`.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                 'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}


info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                 'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2
```

```
     2     1
```

Generate and Map CSI-RS Symbols Used for Tracking

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS resource configuration object with CSI-RS parameters set for tracking. Specify four periodic NZP-CSI-RS resources in two consecutive slots. Specify for each slot to contain two periodic NZP-CSI-RS resources with periodicity set to `20`. Set the offset for the first two resources to `0`. Set the offset for the next two resources to `1`. Set the row number to `1` and density to `'three'` for all resources.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp','nzp'};
csirs.CSIRSPeriod = {[20 0],[20 0],[20 1],[20 1]};
csirs.RowNumber = [1 1 1 1];
csirs.Density = {'three','three','three','three'};
csirs.SymbolLocations = {6,10,6,10};
csirs.SubcarrierLocations = {0,0,0,0};
```

Generate CSI-RS symbols and indices for the default slot number of the carrier configuration object (slot number 0).

```
ind0 = nrCSIRSIndices(carrier,csirs);
sym0 = nrCSIRS(carrier,csirs);
```

Map the symbols to a carrier grid of one slot duration.

```
gridSize = [12*carrier.NSizeGrid carrier.SymbolsPerSlot max(csirs.NumCSIRSPorts)];
slotgrid0 = complex(zeros(gridSize));
slotgrid0(ind0) = sym0;
```

Change the absolute slot number in the carrier configuration from `0` to `1`.

```
carrier.NSlot = 1;
```

Generate CSI-RS symbols and indices for slot number 1.

```
ind1 = nrCSIRSIndices(carrier,csirs);
sym1 = nrCSIRS(carrier,csirs);
```

Map the symbols to another carrier grid of one slot duration.

```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

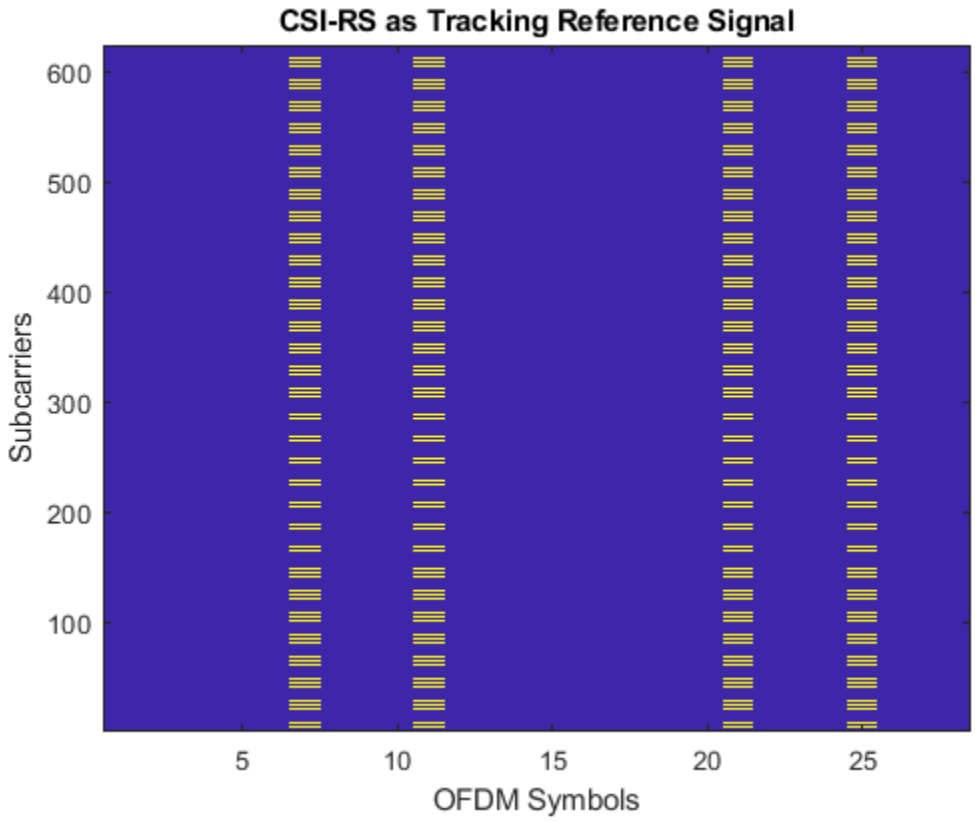
Concatenate the two slots to form the final grid.

```
grid = [slotgrid0 slotgrid1];
```

Plot the grid.

```
imagesc(abs(grid(:,:,1)));
axis xy;
```

```
title('CSI-RS as Tracking Reference Signal');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`csirs` — CSI-RS resource configuration parameters

`nrCSIRSConfig` object

CSI-RS resource configuration parameters, specified as an `nrCSIRSConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'OutputDataType','single'` specifies `single` data type for the output symbols.

`OutputDataType` — Data type of generated CSI-RS symbols
`'double'` (default) | `'single'`

Data type of generated CSI-RS symbols, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`.

Data Types: `char` | `string`

`OutputResourceFormat` — Output format of CSI-RS symbols
`'concatenated'` (default) | `'cell'`

Output format of CSI-RS symbols, specified as the comma-separated pair consisting of `'OutputResourceFormat'` and one of these values:

- `'concatenated'` — The output `sym` is a single column vector containing all CSI-RS symbols concatenated.
- `'cell'` — The output `sym` is a cell array where each cell corresponds to a single CSI-RS resource.

Data Types: `char` | `string`

Output Arguments

`sym` — CSI-RS symbols
complex column vector | cell array of complex column vectors

CSI-RS symbols, returned as a complex column vector or a cell array of complex column vectors.

Data Types: `single` | `double`

`info` — CSI-RS locations information
structure

CSI-RS locations information, returned as a structure containing these fields:

Fields	Description
`ResourceOrder`	Order of CSI-RS resources in terms of `CSIRSType` indices. `CSIRSType` is a property of the input `csirs` configuration object, specifying all CSI-RS resources for which the function generates the output.
`KBarLBar`	Frequency-domain and time-domain locations of the lowest resource elements corresponding to all code division multiplexing (CDM) groups
`CDMGroupIndices`	CDM group indices
`KPrime`	Frequency-domain indexing within a CDM group
`LPrime`	Time-domain indexing within a CDM group

Each field, apart from `ResourceOrder`, returns the information in the resource order specified by the `CSIRSType` property of the input `csirs` configuration object. These fields represent the frequency-domain and time-domain locations of the CSI-RS within a slot for each resource, as defined in TS 38.211 Table 7.4.1.5.3-1.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrCSIRSIndices`

Objects

`nrCSIRSConfig` | `nrCarrierConfig`

Introduced in R2019b

nrCSIRSIndices

Generate CSI-RS resource element indices

Syntax

```
[ind,info] = nrCSIRSIndices(carrier,csirs)
[ind,info] = nrCSIRSIndices(carrier,csirs,Name,Value)
```

Description

`[ind,info] = nrCSIRSIndices(carrier,csirs)` returns resource element indices `ind` for the channel state information reference signal (CSI-RS), as defined in TS 38.211 Section 7.4.1.5.3 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `csirs` specifies CSI-RS resource configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources. When configuring both ZP and NZP resources, the returned indices are in the order ZP followed by NZP, irrespective of the resource order specified by `csirs`. The function also returns the structure `info`, which contains additional information about the CSI-RS locations.

`[ind,info] = nrCSIRSIndices(carrier,csirs,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate CSI-RS Symbols and Indices for 10 MHz Carrier

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object with default properties.

```
csirs = nrCSIRSConfig;
```

Generate CSI-RS symbols of `single` data type.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,'OutputDataType','single');
```

Generate resource element indices for CSI-RS.

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs);
```

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers 3 and 5, symbol locations 13 and 9, and subcarrier locations 6 and 4, respectively. For both resources, set the periodicity to 5, offset to 1, and density to `'one'`.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                 'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}


info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                 'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2
```

```
     2     1
```

Generate and Map CSI-RS Symbols Used for Tracking

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS resource configuration object with CSI-RS parameters set for tracking. Specify four periodic NZP-CSI-RS resources in two consecutive slots. Specify for each slot to contain two periodic NZP-CSI-RS resources with periodicity set to `20`. Set the offset for the first two resources to `0`. Set the offset for the next two resources to `1`. Set the row number to `1` and density to `'three'` for all resources.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp','nzp'};
csirs.CSIRSPeriod = {[20 0],[20 0],[20 1],[20 1]};
csirs.RowNumber = [1 1 1 1];
csirs.Density = {'three','three','three','three'};
csirs.SymbolLocations = {6,10,6,10};
csirs.SubcarrierLocations = {0,0,0,0};
```

Generate CSI-RS symbols and indices for the default slot number of the carrier configuration object (slot number 0).

```
ind0 = nrCSIRSIndices(carrier,csirs);
sym0 = nrCSIRS(carrier,csirs);
```

Map the symbols to a carrier grid of one slot duration.

```
gridSize = [12*carrier.NSizeGrid carrier.SymbolsPerSlot max(csirs.NumCSIRSPorts)];
slotgrid0 = complex(zeros(gridSize));
slotgrid0(ind0) = sym0;
```

Change the absolute slot number in the carrier configuration from `0` to `1`.

```
carrier.NSlot = 1;
```

Generate CSI-RS symbols and indices for slot number 1.

```
ind1 = nrCSIRSIndices(carrier,csirs);
sym1 = nrCSIRS(carrier,csirs);
```

Map the symbols to another carrier grid of one slot duration.

```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

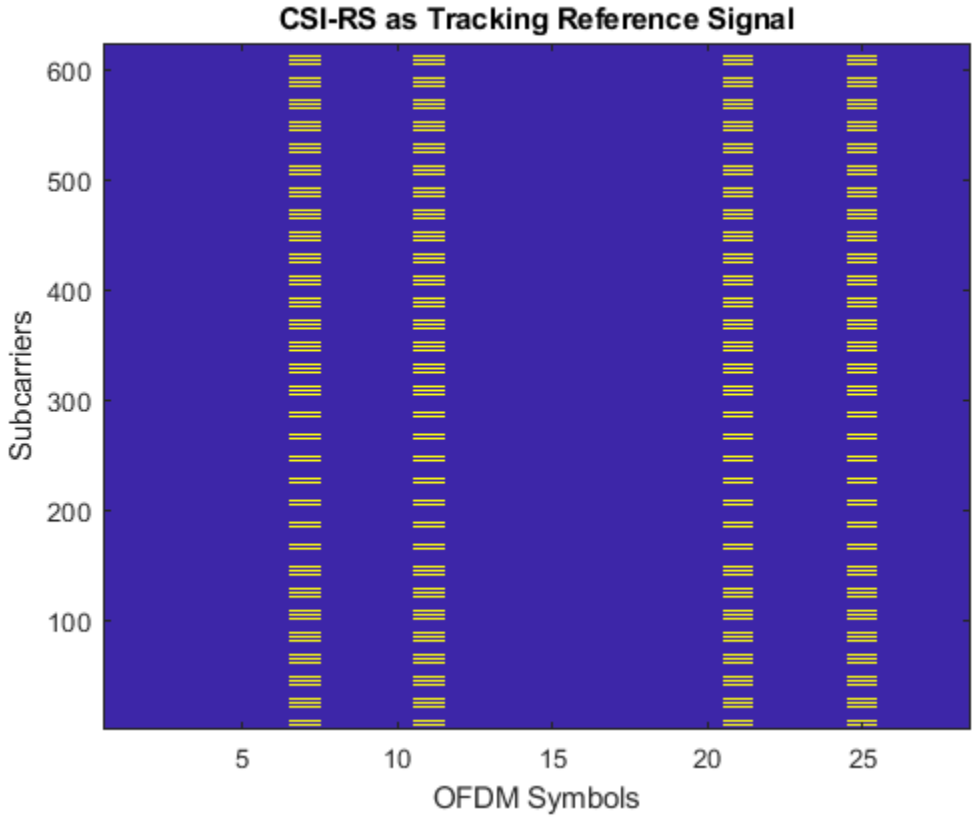
Concatenate the two slots to form the final grid.

```
grid = [slotgrid0 slotgrid1];
```

Plot the grid.

```
imagesc(abs(grid(:,:,1)));
axis xy;
```

```
title('CSI-RS as Tracking Reference Signal');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`csirs` — CSI-RS resource configuration parameters

`nrCSIRSConfig` object

CSI-RS resource configuration parameters, specified as an `nrCSIRSConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the indexing style and indexing base of the output.

`IndexStyle` — Resource element indexing form
`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`OutputResourceFormat` — Output format of CSI-RS resource element indices
`'concatenated'` (default) | `'cell'`

Output format of CSI-RS resource element indices, specified as the comma-separated pair consisting of `'OutputResourceFormat'` and one of these values:

- `'concatenated'` — The output `ind` is a single column vector containing all CSI-RS resource element indices concatenated.
- `'cell'` — The output `ind` is a cell array where each cell corresponds to a single CSI-RS resource.

Data Types: `char` | `string`

Output Arguments

`ind` — CSI-RS resource element indices
column vector | *M*-by-3 matrix | cell array

CSI-RS resource element indices, returned as one of these values:

- Column vector — The function returns this type of value when '`OutputResourceFormat`' is set to `'concatenated'` and '`IndexStyle`' is set to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when '`OutputResourceFormat`' is set to `'concatenated'` and '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.
- Cell array — The function returns this type of value when '`OutputResourceFormat`' is set to `'cell'`. If '`IndexStyle`' is set to `'index'`, each cell is a column vector. If '`IndexStyle`' is set to `'subscript'`, each cell is an *M*-by-3 matrix.

Depending on the value of '`IndexBase`', the function returns either one-based or zero-based indices.

Data Types: `uint32`

`info` — CSI-RS locations information
structure

CSI-RS locations information, returned as a structure containing these fields:

Fields	Description
`ResourceOrder`	Order of CSI-RS resources in terms of `CSIRSType` indices. `CSIRSType` is a property of the input `csirs` configuration object, specifying all CSI-RS resources for which the function generates the output.
`KBarLBar`	Frequency-domain and time-domain locations of the lowest resource elements corresponding to all code division multiplexing (CDM) groups
`CDMGroupIndices`	CDM group indices
`KPrime`	Frequency-domain indexing within a CDM group
`LPrime`	Time-domain indexing within a CDM group

Each field, apart from `ResourceOrder`, returns the information in the resource order specified by the `CSIRSType` property of the input `csirs` configuration object. These fields represent the frequency-domain and time-domain locations of the CSI-RS within a slot for each resource, as defined in TS 38.211 Table 7.4.1.5.3-1.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrCSIRS`

Objects
`nrCSIRSConfig` | `nrCarrierConfig`

Introduced in R2019b

nrDCIDecode

Decode downlink control information (DCI)

Syntax

```
dcibits = nrDCIDecode(softbits,K,L)
[dcibits,mask] = nrDCIDecode(softbits,K,L)
[dcibits,mask] = nrDCIDecode(softbits,K,L,rnti)
```

Description

`dcibits = nrDCIDecode(softbits,K,L)` decodes the input `softbits` and returns the decoded DCI bits of length `K`. The function implements the inverse of the features specified in TS 38.212 Sections 7.3.4, 7.3.3, and 7.3.2 [1], such as rate recovery, polar decoding, and cyclic redundancy check (CRC) decoding. `L` specifies the list length used for polar decoding.

`[dcibits,mask] = nrDCIDecode(softbits,K,L)` also looks for a cyclic redundancy check (CRC) error in the DCI decoding. If `mask` is not equal to 0, either an error has occurred or the input CRC has been masked. When there are no CRC errors, `mask` is the actual value used for masking the CRC bits.

`[dcibits,mask] = nrDCIDecode(softbits,K,L,rnti)` specifies a radio network temporary identifier (RNTI). You can use this syntax when the value of `rnti` masks the CRC parity bits at the transmit end. When you specify `rnti` and there are no CRC errors, `mask` equals to 0.

Examples

Decode DCI Codeword

Create a random sequence of binary values corresponding to a DCI message of 32 bits. Encode the message based on the specified RNTI and rate-matched DCI codeword length. The RNTI masks the CRC parity bits.

```
K = 32;
rnti = 100;
E = 240;
dciBits = randi([0 1],K,1);
dcicw = nrDCIEncode(dciBits,rnti,E);
```

Decode the soft bits representing the DCI codeword `dcicw` by specifying the RNTI used for the CRC masking. Set the length of the polar decoding list to `8`.

```
L = 8;
[recBits,mask] = nrDCIDecode(1-2*dcicw,K,L,rnti)
```

```
recBits = 32x1 int8 column vector

   1
   1
   0
```

```
   1
   1
   0
   0
   1
   1
   1
      ⋮
```

```
mask = uint32
    0
```

Verify that the transmitted and received message bits are identical.

```
isequal(recBits,dciBits)
```

```
ans = logical
   1
```

Verify that the decoding is without error. As the decoding specified the RNTI used for masking, a mask value of 0 indicates no error.

```
mask
```

```
mask = uint32
    0
```

Input Arguments

`softbits` — Coded block of soft bits

column vector of real numbers

Coded block of soft bits, specified as a column vector of real numbers.

Data Types: `double` | `single`

`K` — Length of decoded output in bits

integer

Length of decoded output in bits, specified as an integer from 12 to 140.

Data Types: `double`

`L` — Length of polar decoding list

power of two

Length of polar decoding list, specified as a power of two.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

Output Arguments

`dcibits` — Decoded DCI message bits

K-by-1 column vector of binary values

Decoded DCI message bits, returned as a K-by-1 column vector of binary values. The message bits were transmitted on a single physical downlink control channel (PDCCH).

Data Types: `int8`

`mask` — Result of CRC decoding

nonnegative integer

Result of CRC decoding, returned as a nonnegative integer less than or equal to $2^{16}–1$. If `mask` is not equal to 0, either an error has occurred or the CRC has been masked. When there are no errors, `mask` is the actual value used for masking the CRC bits.

Data Types: `uint32`

Compatibility Considerations

Polar decoding metric update

Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrDCIDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The input argument `L` must be a compile-time constant. Include `{coder.Constant(L)}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions

`nrDCIEncode` | `nrPDCCH` | `nrPDCCHDecode`

Introduced in R2018b

nrDCIEncode

Encode downlink control information (DCI)

Syntax

```
dcicw = nrDCIEncode(dcibits,rnti,E)
```

Description

`dcicw = nrDCIEncode(dcibits,rnti,E)` encodes the input DCI bits and returns the rate-matched DCI codeword of length `E`. The function implements the features described in TS 38.212 Section 7.3.2, 7.3.3, and 7.3.4 [1], such as cyclic redundancy check (CRC) attachment, polar encoding, and rate matching. The CRC parity bits are masked with `rnti`, the radio network temporary identifier (RNTI) of the user equipment (UE).

Examples

Encode DCI Message Bits

Create a random sequence of binary values corresponding to a DCI message of 32 bits.

```
dcibits = randi([0 1],32,1);
```

Encode the message for the specified RNTI and rate-matched output length.

```
rnti = 100;
E = 240;
dcicw = nrDCIEncode(dcibits,rnti,E)
```

```
dcicw = 240×1

     0
     1
     0
     1
     0
     0
     0
     0
     0
     0
     ⋮
```

Input Arguments

`dcibits` — DCI message bits

column vector of binary values

DCI message bits, specified as a column vector of binary values. `dcibits` is the input to the DCI processing to be transmitted on a single physical downlink control channel (PDCCH).

Data Types: `double` | `int8`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

E — Length of rate-matched DCI codeword in bits
positive integer

Length of rate-matched DCI codeword in bits, specified as a positive integer. E must be in the range $K + 24 < E \leq 8192$, where K is the length of `dcibits`.

Data Types: `double`

Output Arguments

`dcicw` — Rate-matched DCI codeword
E-by-1 column vector of binary values

Rate-matched DCI codeword, returned as an E-by-1 column vector of binary values. `dcicw` inherits its data type from the input `dcibits`.

Data Types: `double` | `int8`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrDCIDecode` | `nrPDCCH` | `nrPDCCHDecode`

Introduced in R2018b

nrDLSCHInfo

Get downlink shared channel (DL-SCH) information

Syntax

```
info = nrDLSCHInfo(tBlkLen,targetCodeRate)
```

Description

`info = nrDLSCHInfo(tBlkLen,targetCodeRate)` returns a structure containing DL-SCH information for an input transport block size `tBlkLen` and target code rate `targetCodeRate`. The DL-SCH information includes the cyclic redundancy check (CRC) attachment, code block segmentation (CBS), and channel coding.

Examples

Get DL-SCH Information

Show DL-SCH information before rate matching for an input transport block of length 8456 and target code rate 517/1024. The displayed DL-SCH information shows:

- The transport block has 312 <NULL> filler bits per code block.
- The number of bits per code block, after CBS, is 4576.
- The number of bits per code block, after low-density parity-check (LDPC) coding, is 13,728.

```
tBlkLen = 8456;
targetCodeRate = 517/1024;
nrDLSCHInfo(tBlkLen,targetCodeRate)
```

```
ans = struct with fields:
    CRC: '24A'
      L: 24
    BGN: 1
      C: 2
    Lcb: 24
      F: 312
     Zc: 208
      K: 4576
      N: 13728
```

Input Arguments

`tBlkLen` — Transport block size

nonnegative integer

Transport block size, specified as a nonnegative integer.

Data Types: `double`

`targetCodeRate` — Target code rate
real number

Target code rate, specified as a real number in the range (0, 1).

Data Types: `double`

Output Arguments

`info` — DL-SCH information
structure

DL-SCH information, returned as a structure containing these fields.

Fields	Values	Description
`CRC`	`'16'`, `'24A'`	CRC polynomial selection
`L`	`0`, `16`, `24`	Number of CRC bits
`BGN`	`1`, `2`	LDPC base graph selection
`C`	Positive integer	Number of code blocks
`Lcb`	`0`, `24`	Number of parity bits per code block
`F`	Nonnegative integer	Number of <NULL> filler bits per code block
`Zc`	Positive integer	Lifting size selection
`K`	Nonnegative integer	Number of bits per code block after CBS
`N`	Nonnegative integer	Number of bits per code block after LDPC coding

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrDLSCH` | `nrDLSCHDecoder`

Functions
`nrPDSCH` | `nrPDSCHDecode`

Introduced in R2018b

nrEqualizeMMSE

Minimum mean-squared error (MMSE) equalization

Syntax

```
[eqSym,csi] = nrEqualizeMMSE(rxSym,hest,nVar)
```

Description

`[eqSym,csi] = nrEqualizeMMSE(rxSym,hest,nVar)` applies MMSE equalization to the extracted resource elements of a physical channel `rxSym` and returns the equalized symbols in `eqSym`. The equalization process uses the estimated channel information `hest` and the estimate of the received noise variance `nVar`. The function also returns the soft channel state information `csi`.

Examples

Perform MMSE Equalization for PBCH

Perform MMSE equalization on extracted resource elements of the physical broadcast channel (PBCH).

Create symbols and indices for a PBCH transmission.

```
ncellid = 146;
v = 0;
E = 864;
cw = randi([0 1],E,1);
pbchTxSym = nrPBCH(cw,ncellid,v);
pbchInd = nrPBCHIndices(ncellid);
```

Generate an empty resource array for one transmitting antenna. Populate the array with the PBCH symbols by using the generated PBCH indices.

```
carrier = nrCarrierConfig('NSizeGrid',20);
P = 1;
txGrid = nrResourceGrid(carrier,P);
txGrid(pbchInd) = pbchTxSym;
```

Perform OFDM modulation.

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Create channel matrix and apply channel to transmitted waveform.

```
R = 4;
H = dftmtx(max([P R]));
H = H(1:P,1:R);
H = H / norm(H);
rxWaveform = txWaveform * H;
```

Create channel estimate.

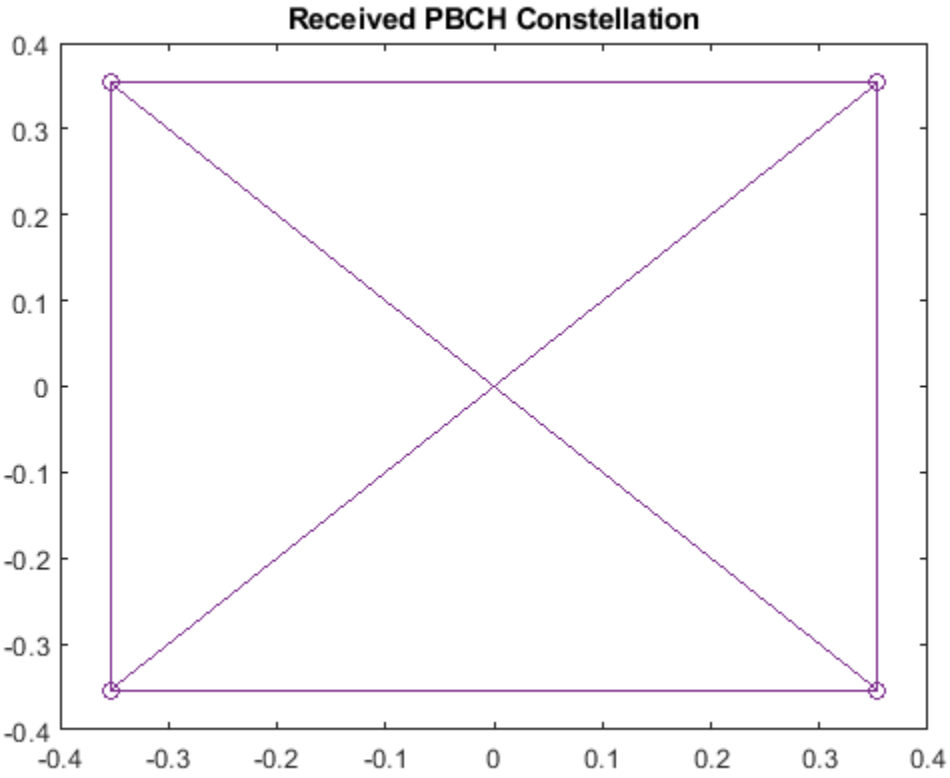
```
hEstGrid = repmat(permute(H.',[3 4 1 2]),[240 4]);
nEst = 0.1;
```

Perform OFDM demodulation.

```
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
```

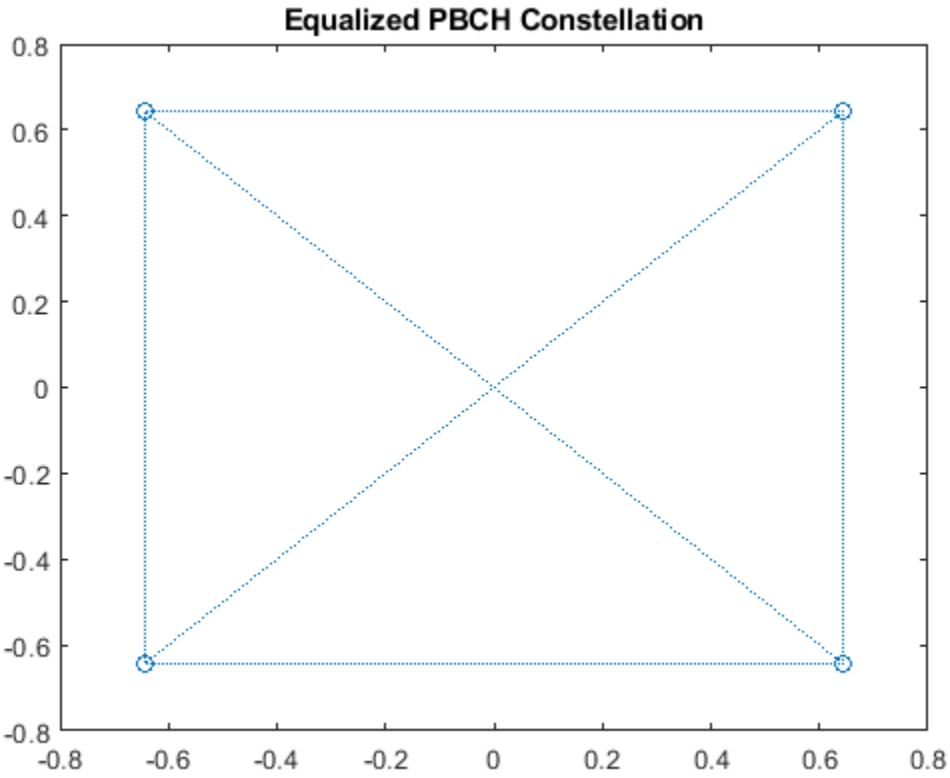
To prepare for PBCH decoding, use `nrExtractResources` to extract symbols from received and channel estimate grids. Plot the received PBCH constellation.

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



Decode the PBCH with the extracted resource elements. Plot the equalized PBCH constellation.

```
[pbchEqSym,csi] = nrEqualizeMMSE(pbchRxSym,pbchHestSym,nEst);
pbchBits = nrPBCHDecode(pbchEqSym,ncellid,v);
figure;
plot(pbchEqSym,'o:');
title('Equalized PBCH Constellation');
```



Input Arguments

`rxSym` — Extracted resource elements

2-D numeric matrix

Extracted resource elements of a physical channel, specified as an *NRE*-by-*R* numeric matrix. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *R* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`hest` — Estimated channel information

3-D numeric array

Estimated channel information, specified as an *NRE*-by-*R*-by-*P* numeric array. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *R* is the number of receive antennas. *P* is the number of transmission planes.

Data Types: `double`
Complex Number Support: Yes

`nVar` — Estimated noise variance

real nonnegative scalar

Estimated noise variance, specified as a real nonnegative scalar.

Data Types: `double`

Output Arguments

`eqSym` — Equalized symbols

2-D numeric matrix

Equalized symbols, returned as an *NRE*-by-*P* numeric matrix. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *P* is the number of transmission planes.

Data Types: `double`
Complex Number Support: Yes

`csi` — Soft channel state information

2-D numeric matrix

Soft channel state information, returned as an *NRE*-by-*P* numeric matrix. *NRE* is the number of resource elements extracted from each *K*-by-*L* plane of the received grid. *K* is the number of subcarriers and *L* is the number of OFDM symbols. *P* is the number of transmission planes.

Data Types: `double`
Complex Number Support: Yes

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrExtractResources` | `nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

Introduced in R2018b

nrExtractResources

Extract resource elements from resource array

Syntax

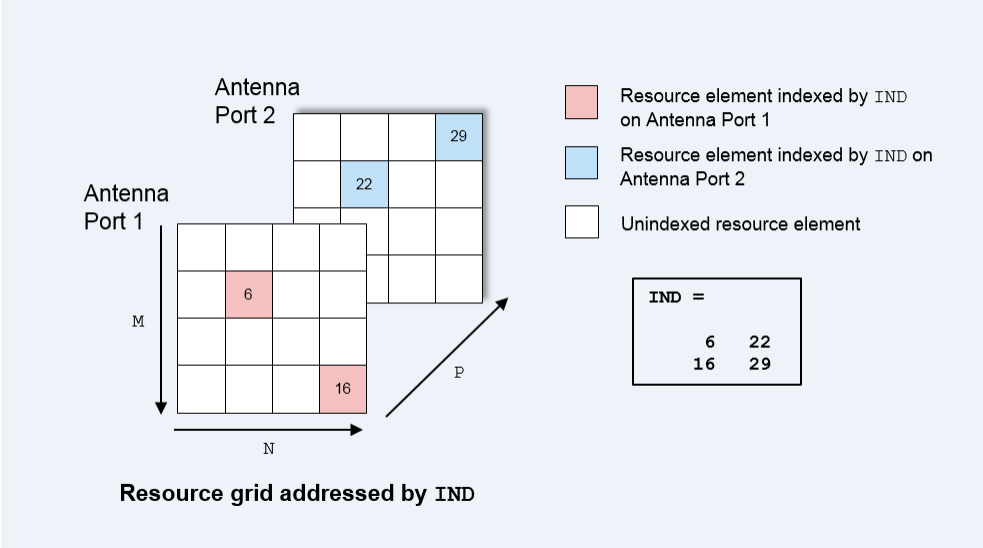
```
re = nrExtractResources(ind,grid)
[re,reind] = nrExtractResources(ind,grid)
[re1,...,reN,reind1,...,reindN] = nrExtractResources(
ind,grid1,grid2,....,gridN)
[___] = nrExtractResources(___,Name,Value)
```

Description

`re = nrExtractResources(ind,grid)` returns the resource elements from the resource array `grid` using resource element indices `ind`. The function can extract resource elements even if `grid` has a dimensionality which is different than the dimensionality of the indices `ind`. In this syntax, the specified indices are one-based using linear indexing form.

Typically, channel or signal specific functions generate resource element indices to map the channel or signal symbols to a resource grid. The indices address resource elements in an *M*-by-*N*-by-*P* array. *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *P* is the number of antenna ports.

For example, the following diagram highlights resource elements of a 4-by-4-by-2 resource array. The resource element indices are in one-based linear indexing form. The number of the antenna ports is two (*P* = 2).



`[re,reind] = nrExtractResources(ind,grid)` also returns `reind`, the indices of the extracted resource elements `re` within the resource array `grid`. The array `reind` is the same size as the extracted resource elements `re`.

`[re1,...,reN,reind1,...,reindN] = nrExtractResources( ind,grid1,grid2,....,gridN)` extracts resource elements from multiple resource arrays using the resource element indices `ind`.

`[___] = nrExtractResources(___,Name,Value)` specifies optional name-value pair arguments in addition to any of the input argument sets in previous syntaxes. Use these name-value pair arguments to specify the format of the input indices and the extraction method. Unspecified arguments take default values.

Examples

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.

```
carrier = nrCarrierConfig('NSizeGrid',20);
P = 2;
txGrid = nrResourceGrid(carrier,P);
F = [1 1i];
[~,bfInd] = nrExtractResources(pbchInd,txGrid);
txGrid(bfInd) = pbchTxSym*F;
```

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

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

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.

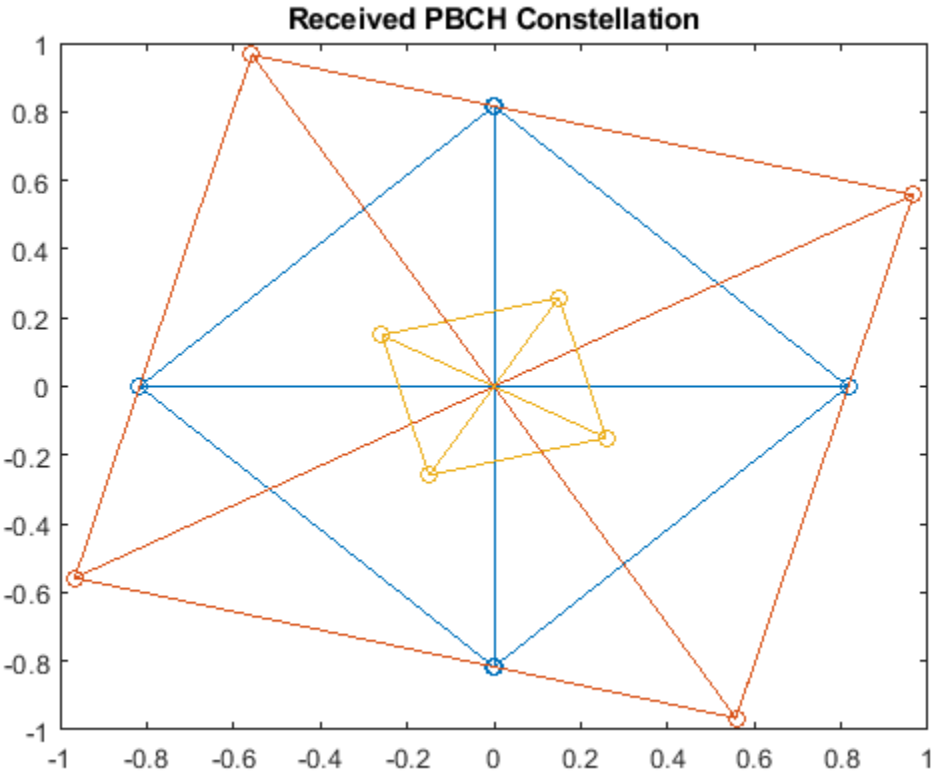
```
hEstGrid = repmat(permute(H.'*F.',[3 4 1 2]),[240 4]);
nEst = 0;
```

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

```
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
```

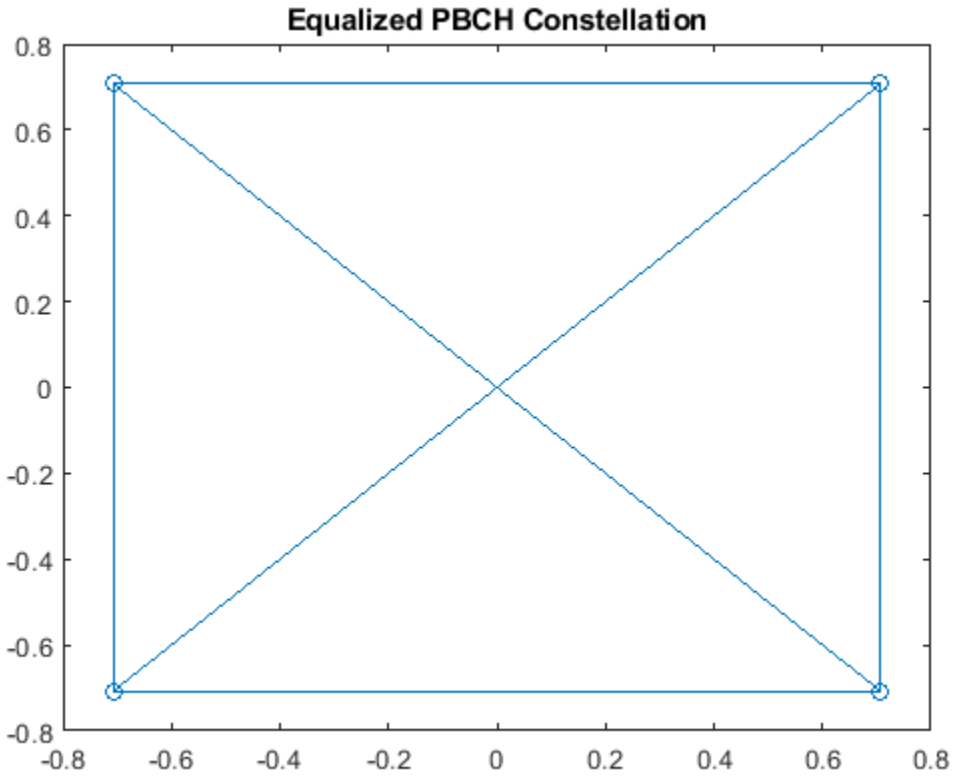
In preparation for PBCH decoding, extract symbols from the received grid and the channel estimate grid.

```
[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 soft bits 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
      ⋮
```

Input Arguments

`ind` — Resource element indices

matrix

Resource element indices, specified as a matrix.

- If `'IndexStyle'` is `'index'`, each column of the matrix contains linear indices for the corresponding antenna.
- If `'IndexStyle'` is `'subscript'`, `ind` is a three-column matrix. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and antennas, respectively.

The function assumes that the indices are one-based, unless you specify otherwise with the `'IndexBase'` argument.

Data Types: `double`

`grid` — Resource array
3-D numeric array (default) | 4-D numeric array

Resource array from which to extract resource elements, specified as one of these values:

- 3-D numeric array of size *M*-by-*N*-by-*R* that corresponds to a received grid — *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *R* is the number of receive antennas. The grid is created after OFDM demodulation.
- A 4-D numeric array of size *M*-by-*N*-by-*R*-by-*P* that corresponds to a channel estimation grid — *P* is the number of antenna ports. The grid is created after channel estimation.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example:
`nrExtractResources(ind,grid,'ExtractionMethod','direct','IndexBase','0based')`
specifies direct extraction method with zero-based indexing.

`IndexStyle` — Resource element indexing form
`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`ExtractionMethod` — Resource element extraction method

`'allplanes'` (default) | `'direct'`

Resource element extraction method, specified as the comma-separated pair consisting of `'ExtractionMethod'` and `'allplanes'` or `'direct'`.

- `'allplanes'` — The function extracts resource elements from each *M*-by-*N* plane within `grid`. The function uses indices that address unique subcarrier and symbol locations over all planes of the indexed resource array. See “All-Planes Extraction Method (Default)” on page 1-75.
- `'direct'` — The function extracts resource elements from each *M*-by-*N* plane (for a 3-D `grid`) or *M*-by-*N*-by-*R* array (for a 4-D `grid`). The function uses indices that address the corresponding plane of the indexed resource array directly. See “Direct Extraction Method” on page 1-77.

Data Types: `string` | `char`

Output Arguments

`re` — Extracted resource elements

column vector | numeric array

Extracted resource elements, returned as a column vector, or a numeric array.

When `'ExtractionMethod'` is set to `'allplanes'`, the size of `re` is N_{RE}-by-*R*-by-*P*, where:

- N_{RE} is the number of resource elements extracted from each *M*-by-*N* plane of `grid`.
- *R* number of receive antennas.
- *P* is the number of planes.

When `'ExtractionMethod'` is set to `'direct'`, the size of `re` depends on the number of indices addressing each plane of the indexed resource grid.

- If the number of indices addressing each plane is the same, then `re` is of size N_{RE}-by-*R*-by-*P*.
- If the number of indices addressing each plane is different, then `re` is a column vector containing all extracted resource elements.

For more details on the resource extraction methods, see “Algorithms” on page 1-75.

`reind` — Indices of extracted resource elements

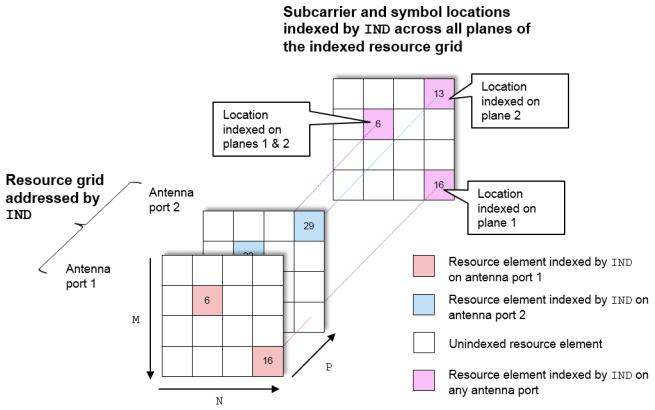
numeric array

Indices of extracted resource elements within `grid`, returned as numeric array. `reind` is the same size as the extracted resource elements array `re`. The `reind` output inherits the indexing style and index base from `ind`.

Algorithms

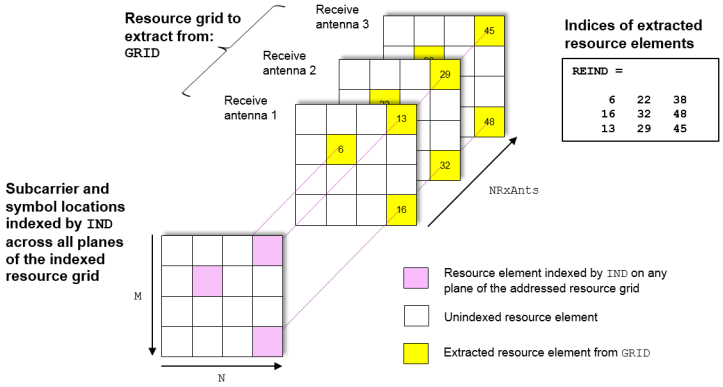
All-Planes Extraction Method (Default)

To use this method, set `'ExtractionMethod'` to `'allplanes'`. This method extracts resource elements from each *M*-by-*N* plane within `grid`. The indices address unique subcarrier and symbol locations over all the planes of the indexed resource array. The diagram highlights the indices used to extract resource elements from a resource grid with $P = 2$.



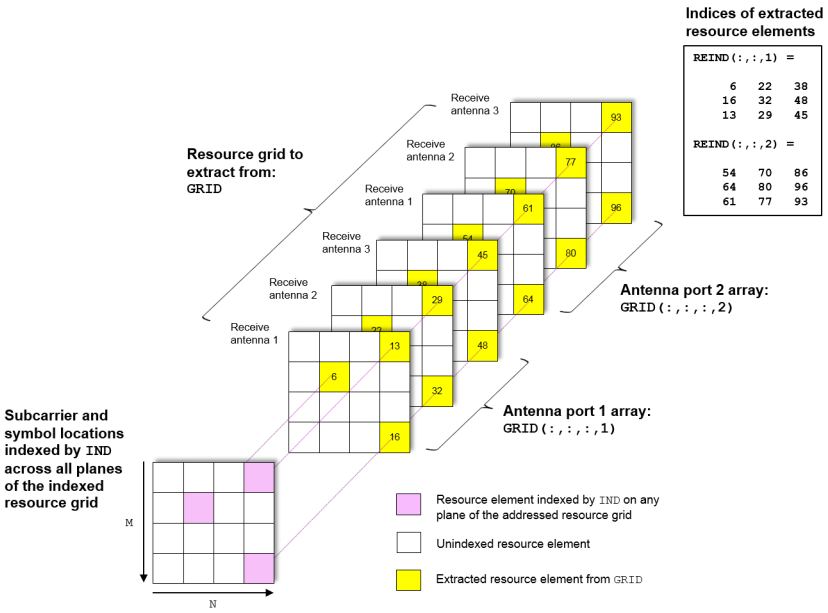
Extraction Process for a 3-D Received Grid

The following diagrams illustrate the resource element extraction from a 3-D received grid, where the number of receive antennas $R = 3$. Resource elements are extracted from the grid at the symbol and subcarrier locations.



Extraction Process for a 4-D Channel Estimate Grid

The following diagram shows the extraction process for a 4-D channel estimate grid. The number of receive antennas $R = 3$ and the number of antenna ports $P = 2$. The 4-D resource grid consists of P M-by-N-by-R arrays, each associated with an antenna port. Resource elements are extracted from all planes within these arrays.



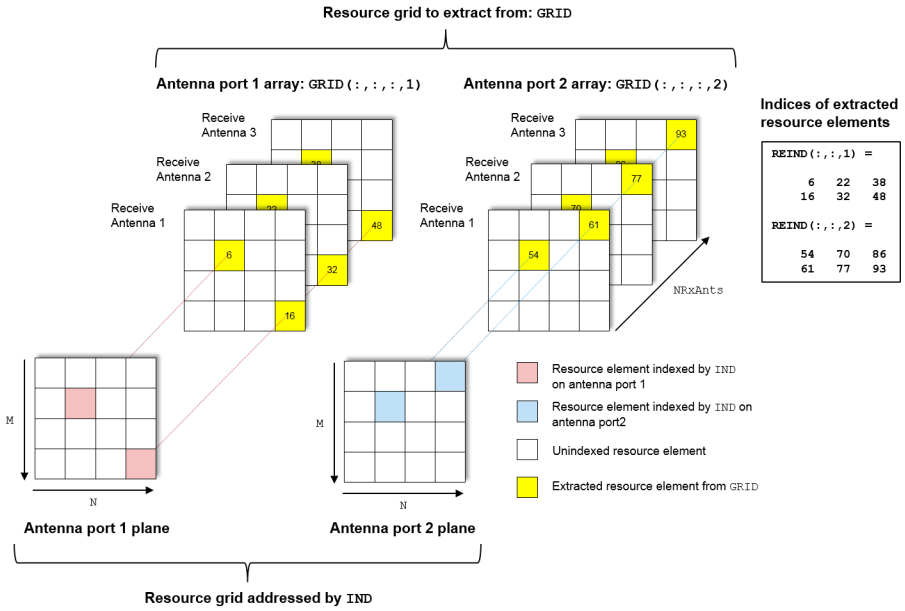
Direct Extraction Method

To use this method, set `'ExtractionMethod'` to `'direct'`. This method extracts resource elements from `grid` assuming that the third and fourth dimensions of the `grid` represent the same property as the planes of the indexed resource array such as antenna ports, layers, transmit antennas. Therefore the function extracts only the resource elements relevant to each plane of the indexed resource grid.

- For a 3-D `grid`, the direct method extracts elements from each *M*-by-*N* plane of `grid` using indices addressing the same plane of the indexed resource array. This method is the same as the standard MATLAB® operation `re = grid(ind)`. Therefore, `reind = ind`.
- For a 4-D `grid`, the direct method extracts elements from each *M*-by-*N*-by-*R* array of `grid` by using indices addressing the same plane of the indexed resource array. The function assumes that the property represented by the planes of the indexed resource array is the same as the fourth dimension of `grid`.

Extraction Process for a 4-D Channel Estimate Grid

The following diagram shows the extraction process for a 4-D channel estimate grid. The number of receive antennas $R = 3$ and the number of antenna ports $P = 2$. The 4-D resource grid consists of P number of M-by-N-by-R arrays, each associated with an antenna port. The indices corresponding to each individual antenna port in the indexed resource array are used to extract resource elements from each of these arrays.



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrEqualizeMMSE` | `nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPSSIndices` | `nrSSSIndices`

Introduced in R2018b

nrLayerDemap

Layer demapping onto scrambled and modulated codewords

Syntax

```
out = nrLayerDemap(in)
```

Description

`out = nrLayerDemap(in)` returns one or two codewords obtained from layer demapping the received layered symbols specified by `in`. The function determines the number of codewords based on the number of layers, as specified in TS 38.211 Table 7.3.1.3-1 [1].

Examples

Layer Mapping and Demapping of Single Codeword

Map a single codeword of length 20 to four transmission layers.

```
codeword = ones(20,1);
nLayers = 4;
layeredOut = nrLayerMap(codeword,nLayers);
```

Recover the original codeword using layer demapping.

```
out = nrLayerDemap(layeredOut);
```

Check for errors.

```
isequal(codeword,out{1})
```

```
ans = logical
   1
```

Input Arguments

`in` — Layered modulation symbols

complex matrix

Layered modulation symbols, specified as a complex matrix of size *M*-by-*nLayers*. *M* is the number of modulation symbols in a transmission layer. *nLayers* is the number of transmission layers in the range 1 to 8.

Data Types: `double`

Output Arguments

`out` — Modulation symbols in codewords

cell array of one or two complex column vectors

Modulation symbols in codewords, returned as a cell array of one or two complex column vectors. This output inherits the data type of the input `in`. One vector corresponds to one codeword. The number of codewords is based on the number of layers. The function determines the number of codewords using TS 38.211 Table 7.3.1.3-1.

Data Types: `cell`

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrLayerMap` | `nrPDSCHDecode` | `nrSymbolDemodulate`

Introduced in R2018b

nrLayerMap

Layer mapping of modulated and scrambled codewords

Syntax

```
out = nrLayerMap(in,nLayers)
```

Description

`out = nrLayerMap(in,nLayers)` performs layer mapping of one or two codewords, specified by `in`, based on the number of transmission layers `nLayers`. The transmission layers in the output are formed by multiplexing the modulation symbols from either one or two codewords. The function implements the transpose of the overall layer mapping specified in TS 38.211 Section 6.3.1.3 and Section 7.3.1.3 [1]. In other words, the symbols in a layer lie in columns rather than rows.

Examples

Map One Codeword to Four Transmission Layers

Perform layer mapping of one codeword of length 40, using 4 transmission layers.

```
out = nrLayerMap(ones(40,1),4);
sizeOut = size(out)
```

```
sizeOut = 1×2

    10     4
```

Map Two Codewords to Five Transmission Layers

Perform layer mapping of two codewords of length 20 and 30 respectively, using 5 transmission layers.

```
out = nrLayerMap({ones(20,1),ones(30,1)},5);
sizeOut = size(out)
```

```
sizeOut = 1×2

    10     5
```

Input Arguments

`in` — Modulation symbols in codewords

complex column vector | cell array of one or two complex column vectors

Modulation symbols in codewords, specified as one of these values:

- Complex column vector — Use this value to specify one codeword.
- Cell array of one or two complex column vectors — Use this value to specify one or two codewords.

Data Types: `double`

`nLayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8.

Data Types: `double`

Output Arguments

`out` — Layered modulation symbols
complex matrix

Layered modulation symbols, returned as a complex matrix of size *M*-by-`nLayers`. *M* is the number of modulation symbols (rows) in a transmission layer (column). The output `out` inherits the data type of the input `in`.

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrLayerDemap` | `nrPDSCH` | `nrSymbolModulate`

Introduced in R2018b

nrLDPCDecode

Low-density parity-check (LDPC) decoding

Syntax

```
[out,actNumIter,finalParityChecks] = nrLDPCDecode(in,bgn,maxNumIter)
[out,actNumIter,finalParityChecks] = nrLDPCDecode( ___ ,Name,Value)
```

Description

`[out,actNumIter,finalParityChecks] = nrLDPCDecode(in,bgn,maxNumIter)` returns the LDPC-decoded output matrix `out` for the input data matrix `in`, base graph number `bgn`, and maximum number of decoding iterations `maxNumIter`. The function also returns the actual number of iterations `actNumIter` and the final parity checks per codeword `finalParityChecks`.

The decoder uses the sum-product message-passing algorithm. The data bits must be LDPC-encoded as defined in TS 38.212 Section 5.3.2 [1].

`[out,actNumIter,finalParityChecks] = nrLDPCDecode( ___ ,Name,Value)` specifies optional name-value pair arguments, in addition to the input arguments in the previous syntax.

Examples

Decode LDPC Codeword

Create transmit data consisting of two code block segments of length 2560 and 36 filler bits at the end.

```
C = 2;
K = 2560;
F = 36;
txcbs = ones(K-F,C);
fillers = -1*ones(F,C);
txcbs = [txcbs;fillers];
```

Generate LDPC codeword for the transmit data. Use base graph number two.

```
bgn = 2;
txcodedcbs = nrLDPCEncode(txcbs,bgn);
```

Convert transmit data to soft bits. Fillers in the transmit data do not have log likelihood ratio (LLR) soft bits.

```
rxcodedcbs = double(1-2*txcodedcbs);
FillerIndices = find(txcodedcbs(:,1) == -1);
rxcodedcbs(FillerIndices,:) = 0;
```

Decode the encoded codeword with a maximum of 25 iterations.

```
[rxcbs,actualniters] = nrLDPCDecode(rxcodedcbs,bgn,25);
```

Replace filler bits with zero in transmit data and compare the results of encoding and decoding.

```
txcbs(end-F+1:end,:) = 0;
isequal(rxcbs,txcbs)
```

```
ans = logical
   1
```

```
actualniters
```

```
actualniters = 1×2

     1     1
```

Input Arguments

`in` — Rate recovered soft bits for input code block segments

real matrix

Rate recovered soft bits for input code block segments, specified as a real matrix. The number of columns in `in` is equal to the number of scheduled code block segments. The number of rows in `in` is equal to the length of the codeword, with some systematic bits punctured.

Data Types: `double` | `single`

`bgn` — Base graph number

`1` | `2`

Base graph number, specified as `1` or `2`. The value selects one of the two base graphs defined in TS 38.212 Section 5.3.2 [1].

Data Types: `double`

`maxNumIter` — Maximum number of decoding iterations

positive integer scalar

Maximum number of decoding iterations, specified as a positive integer scalar. The decoding is terminated when all parity checks are satisfied, or after `maxNumIter` number of iterations.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[out,actNumIter,finalParityChecks] = nrLDPCDecode(in,bgn,maxNumIter,'DecisionType','hard')`

`OutputFormat` — Output format

`'info'` (default) | `'whole'`

Output format, specified as the comma-separated pair consisting of `'OutputFormat'` and one of these values:

- `'info'` — The number of rows in `out` is equal to the length of the information bits.
- `'whole'` — The number of rows in `out` is equal to the length of the codeword.

Data Types: `char` | `string`

`DecisionType` — Decision method used for decoding
`'hard'` (default) | `'soft'`

Decision method used for decoding, specified as the comma-separated pair consisting of `'DecisionType'` and one of these values:

- `'hard'` — The data type of `out` is `int8`.
- `'soft'` — The output `out` contains log-likelihood ratios of the same data type as `in`.

Data Types: `char` | `string`

`Algorithm` — LDPC decoding algorithm
`'Belief propagation'` (default) | `'Layered belief propagation'` | `'Normalized min-sum'` | `'Offset min-sum'`

LDPC decoding algorithm, specified as the comma-separated pair consisting of `'Algorithm'` and one of these values:

- `'Belief propagation'` — Use this option to specify the belief-passing or message-passing algorithm. For more information, see “Belief Propagation Decoding” on page 1-87.
- `'Layered belief propagation'` — Use this option to specify the layered belief-passing algorithm, which is suitable for quasi-cyclic parity-check matrices (PCMs). For more information, see “Layered Belief Propagation Decoding” on page 1-88.
- `'Normalized min-sum'` — Use this option to specify the layered belief propagation algorithm with normalized min-sum approximation. For more information, see “Normalized Min-Sum Decoding” on page 1-89.
- `'Offset min-sum'` — Use this option to specify the layered belief propagation algorithm with offset min-sum approximation. For more information, see “Offset Min-Sum Decoding” on page 1-89.

Note When you specify the value of this name-value pair argument as `'Normalized min-sum'` or `'Offset min-sum'`, the function clips the input (log-likelihood ratio) LLR values to the [-1e10 1e10] range before decoding.

Data Types: `char` | `string`

`ScalingFactor` — Scaling factor for normalized min-sum decoding
`0.75` (default) | real scalar in the range (0, 1]

Scaling factor for normalized min-sum decoding, specified as the comma-separated pair consisting of `'ScalingFactor'` and a real scalar in the range (0, 1].

Dependencies

To enable this name-value pair argument, set the `'Algorithm'` name-value pair argument to `'Normalized min-sum'`.

Data Types: `double`

`Offset` — Offset for offset min-sum decoding

`0.5` (default) | nonnegative finite real scalar

Offset for offset min-sum decoding, specified as the comma-separated pair consisting of `'Offset'` and a nonnegative finite real scalar.

Dependencies

To enable this name-value pair argument, set the `'Algorithm'` name-value pair argument to `'Offset min-sum'`.

Data Types: `double`

`Termination` — Decoding termination criteria

`'early'` (default) | `'max'`

Decoding termination criteria, specified as the comma-separated pair consisting of `'Termination'` and one of these values:

- `'early'` — The decoding terminates when all parity checks are satisfied or after `maxNumIter` number of iterations.
- `'max'` — The decoding terminates after `maxNumIter` number of iterations.

Data Types: `char` | `string`

Output Arguments

`out` — Decoded LDPC codeword

real matrix

Decoded LDPC codeword or information bits, returned as a real matrix. The number of columns in `out` is equal to the number of scheduled code block segments. The number of rows in `out` depends on the name-value pair argument `'OutputFormat'`. The data type of `out` depends on the name-value pair argument `'DecisionType'`.

Data Types: `single` | `double` | `int8`

`actNumIter` — Actual number of iterations

row vector of positive integers

Actual number of iterations, returned as a row vector of positive integers. The length of `actNumIter` is equal to the number of columns in `in`. The *i*th element in `actNumIter` corresponds to the actual number of iterations executed for the *i*th column of `in`.

Data Types: `double`

`finalParityChecks` — Final parity checks

matrix

Final parity checks, returned as a matrix. The number of rows in `finalParityChecks` is equal to the number of parity-check bits in an LDPC codeword. The *i*th column in `finalParityChecks` corresponds to the final parity checks for the *i*th codeword.

Data Types: `double`

Algorithms

The `nrLDPCDecode` function supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The implementation of the belief propagation algorithm is based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword, c, where $c = (c_0, c_1, ..., c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 1 \,|\, \text{channel output for } c_i)}\right)$.

In each iteration, the key components of the algorithm are updated based on these equations:

$$L(r_{ji}) = 2\,\text{atanh}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right)\right),$$

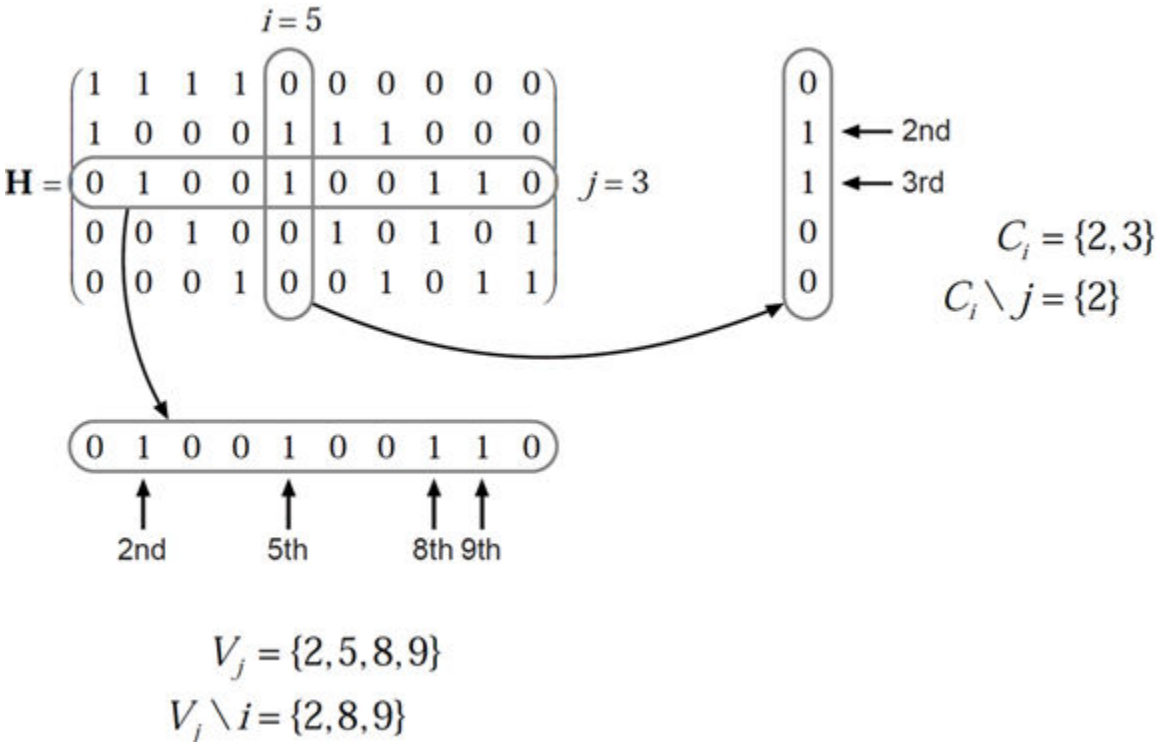
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

At the end of each iteration, $L(Q_i)$ is an updated estimate of the LLR value for the transmitted bit c_i. The value $L(Q_i)$ is the soft-decision output for c_i. If $L(Q_i) < 0$, the hard-decision output for c_i is 1. Otherwise, the output is 0.

Index sets $C_i \backslash j$ and $V_j \backslash i$ are based on the parity-check matrix (PCM). Index sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure highlights the computation of these index sets in a given PCM for $i = 5$ and $j = 3$.



To avoid infinite numbers in the algorithm equations, atanh(1) and atanh(-1) are set to 19.07 and –19.07, respectively. Due to finite precision, MATLAB returns 1 for tanh(19.07) and –1 for tanh(–19.07).

When the name-value pair argument `'Termination'` is set to `'max'`, the decoding terminates after `maxNumIter` number of iterations. When `'Termination'` is set to `'early'`, the decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^{\mathrm{T}} = 0$) or after `maxNumIter` number of iterations.

Layered Belief Propagation Decoding

The implementation of the layered belief propagation algorithm is based on the decoding algorithm presented in [3], Section II.A. The decoding loop iterates over subsets of rows (layers) of the PCM. For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations:

(1) $L(q_{mj}) = L(q_j) - R_{mj}$,

(2) $A_{mj} = \underset{n \in N(m)}{\underset{n \neq j}{\sum}} \psi(L(q_{mn}))$,

(3) $s_{mj} = \underset{n \in N(m)}{\underset{n \neq j}{\prod}} \mathrm{sign}(L(q_{mn}))$,

(4) $R_{mj} = -s_{mj}\psi(A_{mj})$, and

(5) $L(q_j) = L(q_{mj}) + R_{mj}$.

For each layer, the decoding equation (5) works on the combined input obtained from the current LLR inputs $L(q_{mj})$ and the previous layer updates R_{mj}.

Because only a subset of the nodes is updated in a layer, the layered belief propagation algorithm is faster compared to the belief propagation algorithm. To achieve the same error rate as attained with belief propagation decoding, use half the number of decoding iterations when using the layered belief propagation algorithm.

Normalized Min-Sum Decoding

The implementation of the normalized min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})|) \cdot \alpha),$$

where α is in the range (0, 1] and is the scaling factor specified by `ScalingFactor`. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The implementation of the offset min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \max(\min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| - \beta), 0),$$

where $\beta \geq 0$ and is the offset specified by `Offset`. This equation is an adaptation of equation (5) presented in [4].

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

[3] Hocevar, D.E. "A reduced complexity decoder architecture via layered decoding of LDPC codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004*. doi: 10.1109/SIPS.2004.1363033

[4] Chen, Jinghu, R.M. Tanner, C. Jones, and Yan Li. "Improved min-sum decoding algorithms for irregular LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005*. doi: 10.1109/ISIT.2005.1523374

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify soft decision type, include `{coder.Constant('DecisionType'),coder.Constant('soft')}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions

`nrCRCDecode` | `nrCodeBlockDesegmentLDPC` | `nrDLSCHInfo` | `nrLDPCEncode` | `nrRateRecoverLDPC`

Introduced in R2018b

nrLDPCEncode

Low-density parity-check (LDPC) encoding

Syntax

```
out = nrLDPCEncode(in,bgn)
```

Description

`out = nrLDPCEncode(in,bgn)` returns the LDPC-encoded output matrix for the input data matrix `in` and base graph number `bgn`, as specified in TS 38.212 Section 5.3.2 [1]. If applicable, the function replaces each filler bit represented by `-1` in the input by `0`. After encoding, the function replaces each filler bit again by `-1`. The encoding includes puncturing of some of the systematic information bits.

Examples

Generate LDPC Codeword

Create input data for encoding consisting of two code block segments of length 2560 and 36 filler bits at the end.

```
C = 2;
K = 2560;
F = 36;
cbs = ones(K-F,C);
fillers = -1*ones(F,C);
cbs = [cbs;fillers];
```

Generate LDPC codeword for the two code block segments. Use base graph number two.

```
bgn = 2;
codedcbs = nrLDPCEncode(cbs,bgn);
size(codedcbs)
```

```
ans = 1×2

       12800           2
```

Input Arguments

`in` — Code block segments before encoding

matrix | column vector

Code block segments before encoding, specified as a matrix or a column vector. The number of columns in `in` is equal to the number of scheduled code block segments in the transport block. The number of rows in `in` is equal to the length of the code block segment, including the filler bits, if any.

Note Filler bits are represented by `-1` and are treated as `0` when performing encoding.

Data Types: `double` | `int8`

bgn — Base graph number
1 | 2

Base graph number, specified as `1` or `2`. The values correspond to the two base graphs defined in TS 38.212 Section 5.3.2 [1]

Data Types: `double`

Output Arguments

`out` — Encoded LDPC codeword
matrix

Encoded LDPC codeword output, returned as a matrix. The number of columns in `out` is equal to the number of scheduled code block segments in the transport block. The number of rows in `out` is equal to the length of the codeword. Each codeword punctures some of the systematic bits and can contain filler bits.

Data Types: `double` | `int8`

References

[1] 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCRCEncode` | `nrCodeBlockSegmentLDPC` | `nrDLSCHInfo` | `nrLDPCDecode` | `nrRateMatchLDPC`

Introduced in R2018b

nrLowPAPRS

Generate low peak-to-average power ratio (low-PAPR) sequence

Syntax

```
seq = nrLowPAPRS(u,v,alpha,m)
seq = nrLowPAPRS( ___ ,'OutputDataType',datatype)
```

Description

`seq = nrLowPAPRS(u,v,alpha,m)` generates low-PAPR sequence `seq` of length `m`, as defined in TS 38.211, Section 5.2.2 [1]. `u` specifies one of the 30 sequence groups. `v` specifies the base sequence number within the sequence group, as 0 or 1. The function applies phase rotation to the base sequence corresponding to the cyclic shift specified by `alpha`. When `alpha` has more than one value, the function applies different phase rotations to the base sequence and returns several low-PAPR sequences in a matrix format.

Low-PAPR sequences are used for the generation of uplink (UL) demodulation reference signals (DM-RS), sounding reference signals (SRS), and physical uplink control channel (PUCCH) format 0 and 1 modulation symbols.

`seq = nrLowPAPRS( ___ ,'OutputDataType',datatype)` specifies the data type of the low-PAPR sequence in addition to the input arguments in the previous syntax.

Examples

Generate Low-PAPR Sequence

Generate a low-PAPR sequence of length 36 for sequence group number 9, base sequence number 0, and the specified cyclic shift.

```
u = 9;
v = 0;
alpha = 2*pi;
m = 36;
seq = nrLowPAPRS(u,v,alpha,m)
```

```
seq = 36×1 complex

   1.0000 + 0.0000i
  -0.4404 - 0.8978i
   0.9795 + 0.2013i
   0.9190 + 0.3944i
   0.1514 - 0.9885i
   0.5290 + 0.8486i
   0.1514 + 0.9885i
   0.9795 - 0.2013i
  -0.7588 + 0.6514i
  -0.9949 + 0.1012i
      ⋮
```

Generate Multiple Low-PAPR Sequences

Generate low-PAPR sequences of `single` data type and length 36 for sequence group number 9, base sequence number 0, and the specified cyclic shifts. Specifying more than one cyclic shifts as a vector results in the generation of multiple low-PAPR sequences.

```
u = 9;
v = 0;
alpha = [pi/2,pi];
m = 36;
seq = nrLowPAPRS(u,v,alpha,m,'OutputDataType','single')
```

```
seq = 36x2 single matrix

   1.0000 + 0.0000i   1.0000 + 0.0000i
   0.8978 - 0.4404i   0.4404 + 0.8978i
  -0.9795 - 0.2013i   0.9795 + 0.2013i
   0.3944 - 0.9190i  -0.9190 - 0.3944i
   0.1514 - 0.9885i   0.1514 - 0.9885i
  -0.8486 + 0.5290i  -0.5290 - 0.8486i
  -0.1514 - 0.9885i   0.1514 + 0.9885i
  -0.2013 - 0.9795i  -0.9795 + 0.2013i
  -0.7588 + 0.6514i  -0.7588 + 0.6514i
  -0.1012 - 0.9949i   0.9949 - 0.1012i
      ⋮
```

Input Arguments

`u` — Sequence group number

integer from 0 to 29

Sequence group number, specified as an integer from 0 to 29.

Data Types: `double`

`v` — Base sequence number

`0` | `1`

Base sequence number within a sequence group, specified as `0` or `1`. When the low-PAPR sequence length `m` is less than 72, the sequence group has only one base sequence. In this case, only base sequence number `0` applies. When the low-PAPR sequence length `m` is greater than or equal to 72, the sequence group has two base sequences. In this case, both base sequence number `0` and `1` apply.

Data Types: `double`

`alpha` — Cyclic shifts

nonnegative scalar | numeric vector

Cyclic shifts, specified as a nonnegative scalar or numeric vector of nonnegative values. A scalar specifies one cyclic shift. A vector of length N specifies N cyclic shifts. The number of cyclic shifts provided in `alpha` determines the number of low-PAPR sequences returned in `seq`. The function applies different phase rotations to the base sequence corresponding to the specified cyclic shifts.

Data Types: `double`

`m` — Low-PAPR sequence length
nonnegative integer

Low-PAPR sequence length, specified as a nonnegative integer. When `m` is `0`, `seq` is an empty vector.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`seq` — Low-PAPR sequence
complex matrix

Low-PAPR sequence, returned as an `m`-by-*N* complex matrix, where *N* is the number of cyclic shifts provided in the input `alpha`. When `m` is `0`, `seq` is an empty vector.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRBS` | `nrPUCCH0` | `nrPUCCH1`

Introduced in R2019a

nrOFDMDemodulate

Demodulate OFDM waveform

Syntax

```
grid = nrOFDMDemodulate(carrier,waveform)
grid = nrOFDMDemodulate(waveform,nrb,scs,initNSlot)
grid = nrOFDMDemodulate(waveform,nrb,scs,initNSlot,'CyclicPrefix',cpl)
grid = nrOFDMDemodulate( ___ ,Name,Value)
```

Description

`grid = nrOFDMDemodulate(carrier,waveform)` recovers a carrier resource array by demodulating `waveform`, an OFDM modulated waveform, for carrier configuration parameters `carrier`.

`grid = nrOFDMDemodulate(waveform,nrb,scs,initNSlot)` demodulates `waveform` for `nrb`, the specified number of resource blocks, subcarrier spacing `scs`, and initial slot number `initNSlot`.

`grid = nrOFDMDemodulate(waveform,nrb,scs,initNSlot,'CyclicPrefix',cpl)` specifies cyclic prefix length `cpl` in addition to the input arguments from the previous syntax.

`grid = nrOFDMDemodulate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to any combination of input arguments from the previous syntaxes.

Examples

Demodulate OFDM Waveform

Recover a transmitted carrier resource array by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying 106 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Generate physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
txGrid = nrResourceGrid(carrier,p);
txGrid(ind) = sym;
```

Generate OFDM modulated waveform.

```
[txWaveform,~] = nrOFDMModulate(carrier,txGrid);
```

Pass the waveform through a simple 2-by-1 channel.

```
H = [0.6; 0.4];
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform.

```
grid = nrOFDMDemodulate(carrier,waveform);
```

Demodulate OFDM Waveform with Extended Cyclic Prefix

Recover a resource array that contains PDSCH DM-RS symbols by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
scs = 60;
carrier = nrCarrierConfig('SubcarrierSpacing',scs);
```

Generate PDSCH DM-RS symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
txGrid = nrResourceGrid(carrier,p);
txGrid(ind) = sym;
```

Generate an OFDM modulated waveform, specifying the subcarrier spacing, initial slot number, and cyclic prefix length.

```
initNSlot = carrier.NSlot;
cpl = 'extended';
[txWaveform,info] = nrOFDMModulate(txGrid,scs,initNSlot,'CyclicPrefix',cpl);
```

Pass the waveform through a simple 2-by-1 channel.

```
H = [0.9; 0.95];
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform.

```
nrb = carrier.NSizeGrid;
grid = nrOFDMDemodulate(waveform,nrb,scs,initNSlot,'CyclicPrefix',cpl);
```

Demodulate OFDM Waveform with Specified Sample Rate

Recover a transmitted resource array that contains sounding reference signals (SRSs) and spans an entire frame by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity and offset.

```
srs = nrSRSConfig('SRSPeriod',[4 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating slot resource arrays.

```
frameGrid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
    frameGrid = [frameGrid slotGrid];
end
```

Generate the OFDM modulated waveform.

```
[txWaveform,~] = nrOFDMModulate(carrier,frameGrid);
```

Pass the waveform through a simple channel.

```
H = 0.86;
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform, specifying the sample rate.

```
sr = info.SampleRate;
grid = nrOFDMDemodulate(carrier,waveform,'SampleRate',sr);
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid

52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of 52 corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`waveform` — OFDM modulated waveform
complex-valued matrix

OFDM modulated waveform, specified as a complex-valued matrix of size *T*-by-*R*.

- *T* is the number of time-domain samples in the waveform.
- *R* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

`nrb` — Number of resource blocks
integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing
`15` | `30` | `60` | `120` | `240`

Subcarrier spacing, in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initNSlot` — Initial slot number

nonnegative integer

Initial slot number, in zero-based form, specified as a nonnegative integer. The function selects the appropriate cyclic prefix lengths for OFDM demodulation by using the value of `initNSlot` mod S, where S is the number of slots per subframe.

Data Types: `double`

`cpl` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'CyclicPrefixFraction',0.75` specifies the start location for demodulation relative to the cyclic prefix length.

`Nfft` — Number of FFT points

integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and either a nonnegative integer greater than 127, or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy, defined as the value of $(12 \times N_{\mathrm{RB}})/$ `Nfft`, where N_{RB} is the number of resource blocks, of 100%.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets a default value satisfying these conditions.

- The value of this input is an integer power of 2.
- The maximum occupancy is 85%.
- The minimum value of this input is 128.

Data Types: `double`

`SampleRate` — Waveform sample rate

positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{\mathrm{fft}} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- *SCS* is the subcarrier spacing specified in the `SubcarrierSpacing` property of the `config` input for the first function syntax, or the `scs` input for the other syntaxes.

Data Types: `double`

`Windowing` — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as the comma-separated pair consisting of `'Windowing'` and either a nonnegative integer or `[]`.

If you do not specify this input, or if you specify `'Windowing',[]`, the function sets this input to the maximum value that does not impact error vector magnitude (EVM) tests, as specified in [1], [2], and [3].

Data Types: `double`

`CarrierFrequency` — Carrier frequency
`0` (default) | nonnegative scalar

Carrier frequency, in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a nonnegative scalar. This input corresponds to f_0, defined in Section 5.4 of [4].

Data Types: `double`

`CyclicPrefixFraction` — FFT window position within cyclic prefix
`0.5` (default) | scalar in the interval [0, 1]

Fast Fourier transform (FFT) window position within the cyclic prefix, specified as the comma-separated pair consisting of `'CyclicPrefixFraction'` and a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.

Data Types: `double`

Output Arguments

`grid` — Carrier resource array
complex-valued array

Carrier resource array, returned as a complex-valued array of size *K*-by-*L*-by-*R*.

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.
- *R* is the number of receive antennas.

Data Types: `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

The `nrb`, `scs`, and `cpl` inputs must be constants. For example, to specify an extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

The `'SampleRate'` name-value-pair argument cannot be used with function syntaxes that use the `carrier` input.

See Also

Functions

`nrOFDMInfo` | `nrOFDMModulate` | `nrResourceGrid`

Objects

`nrCarrierConfig`

Introduced in R2020b

nrOFDMInfo

Get OFDM information

Syntax

```
info = nrOFDMInfo(carrier)
info = nrOFDMInfo(nrb,scs)
info = nrOFDMInfo(nrb,scs,'cyclicPrefix',cpl)
info = nrOFDMInfo( ___ ,Name,Value)
```

Description

`info = nrOFDMInfo(carrier)` provides dimensional information relevant to orthogonal frequency-division multiplexing (OFDM) modulation for the specified carrier configuration parameters.

`info = nrOFDMInfo(nrb,scs)` provides OFDM information for `nrb`, the specified number of resource blocks, and subcarrier spacing `scs`.

`info = nrOFDMInfo(nrb,scs,'cyclicPrefix',cpl)` specifies cyclic prefix length `cpl` in addition to the input arguments from the previous syntax.

`info = nrOFDMInfo( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to any combination of input arguments from the previous syntaxes.

Examples

Generate OFDM Modulated Waveform

Generate a waveform by performing OFDM modulation of a resource array that contains sounding reference signals (SRSs). The resource array spans an entire frame.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity to 2 and the offset to zero.

```
srs = nrSRSConfig('SRSPeriod',[2 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating individual slot resource arrays.

```
grid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
```

```
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
    grid = [grid slotGrid];
end
```

Perform OFDM modulation on the resource array for the specified carrier configuration.

```
[waveform,info] = nrOFDMModulate(carrier,grid);
```

Get OFDM Information for Extended Cyclic Prefix

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz and extended cyclic prefix.

```
scs = 60;
cpl = 'Extended';
```

Set the number of resource blocks to `150`.

Generate and display OFDM information.

```
nrb = 150;
info = nrOFDMInfo(nrb,scs,'CyclicPrefix',cpl)
```

```
info = struct with fields:
                   Nfft: 4096
             SampleRate: 245760000
    CyclicPrefixLengths: [1x48 double]
          SymbolLengths: [1x48 double]
              Windowing: 116
           SymbolPhases: [1x48 double]
         SymbolsPerSlot: 12
        SlotsPerSubframe: 4
           SlotsPerFrame: 40
```

Get OFDM Information for Specified Sample Rate

Set carrier configuration parameters, specifying 106 RBs in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Generate and display OFDM information for the specified sample rate.

```
sr = 1e8;
info = nrOFDMInfo(carrier,'SampleRate',sr)
```

```
info = struct with fields:
                   Nfft: 3200
             SampleRate: 100000000
    CyclicPrefixLengths: [1x14 double]
```

```
    SymbolLengths: [1x14 double]
        Windowing: 112
     SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
   SymbolsPerSlot: 14
 SlotsPerSubframe: 1
    SlotsPerFrame: 10
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid

52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz

`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`nrb` — Number of resource blocks

integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing

`15` | `30` | `60` | `120` | `240`

Subcarrier spacing, in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`cpl` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'SampleRate','1e9'` specifies a sample rate of 1×10^9 Hz.

`Nfft` — Number of FFT points
integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and either a nonnegative integer greater than 127, or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy, defined as the value of $(12 \times N_{RB})/$ `Nfft`, where N_{RB} is the number of resource blocks, of 100%.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets a default value satisfying these conditions.

- The value of this input is an integer power of 2.
- The maximum occupancy is 85%.
- The minimum value of this input is 128.

Data Types: `double`

`SampleRate` — Waveform sample rate
positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- SCS is the subcarrier spacing specified in the `SubcarrierSpacing` property of the `config` input for the first function syntax, or the `scs` input for the other syntaxes.

Data Types: `double`

`Windowing` — Number of time-domain samples for OFDM symbol windowing and overlapping

nonnegative integer (default depends other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as the comma-separated pair consisting of `'Windowing'` and either a nonnegative integer or `[]`.

If you do not specify this input, or if you specify `'Windowing',[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in Annexes F.5.3 and F.5.4 of TS 38.101-1 [1], Annexes F.5.3 and F.5.4 of TS 38.101-2, [2], and Annexes B.5.2 and C.5.2 of TS 38.104 [3]. E is equal to value of `floor`$((N_{CP} - W) \times$ `info.Nfft`$/N_{FFT,\,nominal})$, where N_{CP}, W, and $N_{FFT,\,nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

`CarrierFrequency` — Carrier frequency

`0` (default) | nonnegative scalar

Carrier frequency, in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a nonnegative scalar. This input corresponds to f_0, defined in Section 5.4 of [4].

Data Types: `double`

Output Arguments

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive scalar	Waveform sample rate
`CyclicPrefixLengths`	1-by-N vector of positive integers, where N is the number of OFDM symbols in a subframe.	Cyclic prefix lengths of each OFDM symbol, in samples
`SymbolLengths`	1-by-N vector of positive integers	OFDM symbol lengths, in samples
`Windowing`	Positive integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

Fields	Values	Description
SymbolPhases	1-by-*N* vector of numbers in the interval [-π, π]	Phase compensation of each OFDM symbol, in radians The `nrOFDMModulate` function applies this compensation during modulation to account for phase terms per OFDM symbol, as specified in Section 5.4 of [4]. The `nrOFDMDemodulate` function inverts this phase compensation during demodulation.
SymbolsPerSlot	Positive integer	Number of OFDM symbols in a slot
SlotsPerSubframe	Positive integer	Number of slots in a 1 ms subframe
SlotsPerFrame	Positive integer	Number of slots in a 10 ms frame

Data Types: `struct`

References

[1] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

The `nrb`, `scs`, and `cpl` inputs must be constants. For example, to specify an extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

The `'SampleRate'` name-value-pair argument cannot be used with function syntaxes that use the `carrier` input.

See Also

Functions
`nrOFDMDemodulate` | `nrOFDMModulate` | `nrResourceGrid`

Objects
`nrCarrierConfig`

Introduced in R2020b

nrOFDMModulate

Generate OFDM modulated waveform

Syntax

```
[waveform,info] = nrOFDMModulate(carrier,grid)
[waveform,info] = nrOFDMModulate(grid,scs,initNSlot)
[waveform,info] = nrOFDMModulate(grid,scs,initNSlot,'CyclicPrefix',cpl)
[waveform,info] = nrOFDMModulate( ___ ,Name,Value)
```

Description

`[waveform,info] = nrOFDMModulate(carrier,grid)` generates `waveform`, a time-domain waveform, by performing orthogonal frequency-division multiplexing (OFDM) modulation of carrier resource array `grid` for carrier configuration parameters `carrier`. The function also returns `info`, a structure containing OFDM information.

`[waveform,info] = nrOFDMModulate(grid,scs,initNSlot)` modulates the carrier resource array with subcarrier spacing `scs` and initial slot number `initNSlot`.

`[waveform,info] = nrOFDMModulate(grid,scs,initNSlot,'CyclicPrefix',cpl)` specifies cyclic prefix length `cpl`.

`[waveform,info] = nrOFDMModulate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to any combination of input arguments from the previous syntaxes.

Examples

Generate OFDM Modulated Waveform

Generate a waveform by performing OFDM modulation of a resource array that contains sounding reference signals (SRSs). The resource array spans an entire frame.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity to 2 and the offset to zero.

```
srs = nrSRSConfig('SRSPeriod',[2 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating individual slot resource arrays.

```
grid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
```

```
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
    grid = [grid slotGrid];
end
```

Perform OFDM modulation on the resource array for the specified carrier configuration.

```
[waveform,info] = nrOFDMModulate(carrier,grid);
```

Generate OFDM Modulated Waveform for Extended Cyclic Prefix

Generate a waveform by performing OFDM modulation of a resource array that contains physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) symbols.

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
scs = 60;
carrier = nrCarrierConfig('SubcarrierSpacing',scs);
```

Generate PDSCH DM-RS symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
grid = nrResourceGrid(carrier,p);
grid(ind) = sym;
```

Generate an OFDM modulated waveform, specifying the subcarrier spacing, initial slot number, and cyclic prefix type. Display the OFDM information.

```
initNSlot = carrier.NSlot;
cpl = 'extended';
[waveform,info] = nrOFDMModulate(grid,scs,initNSlot,'CyclicPrefix',cpl);
disp(info)
```

```
                  Nfft: 1024
            SampleRate: 61440000
    CyclicPrefixLengths: [1x48 double]
          SymbolLengths: [1x48 double]
              Windowing: 36
           SymbolPhases: [1x48 double]
         SymbolsPerSlot: 12
       SlotsPerSubframe: 4
          SlotsPerFrame: 40
```

Generate OFDM Modulated Waveform for Specified Sample Rate

Generate a waveform by performing OFDM modulation of a resource array that contains PDSCH DM-RS symbols.

Set carrier configuration parameters, specifying 106 RBs in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Configure PDSCH and generate the corresponding symbols and indices.

```
p = 4;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array and map the PDSCH symbols.

```
grid = nrResourceGrid(carrier,p,'OutputDataType','single');
grid(ind) = sym;
```

Generate OFDM modulated waveform, specifying the sample rate.

```
sr = 1e8;
[waveform,info] = nrOFDMModulate(carrier,grid,'SampleRate',sr);
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid

52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz

`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSlot` — Slot number

`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB

simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`grid` — Carrier resource array
complex-valued array

Carrier resource array, specified as a complex-valued array of size *K*-by-*N*-by-*P*.

- *K* is the number of subcarriers.
- *N* is the number of OFDM symbols.
- *P* is the number of transmit antennas.

Data Types: `single` | `double` | `aveform`
Complex Number Support: Yes

`scs` — Subcarrier spacing
`15` | `30` | `60` | `120` | `240`

Subcarrier spacing, in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initNSlot` — Initial slot number
nonnegative integer

Initial slot number, in zero-based form, specified as a nonnegative integer. The function selects the appropriate cyclic prefix lengths for OFDM modulation by using the value of `initNSlot` mod *S*, where *S* is the number of slots per subframe.

Data Types: `double`

`cpl` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'CyclicPrefix','extended'` specifies extended cyclic prefix length.

`Nfft` — Number of FFT points

integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and either a nonnegative integer greater than 127, or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy, defined as the value of $(12 \times N_{RB})/$ `Nfft`, where N_{RB} is the number of resource blocks, of 100%.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets a default value satisfying these conditions.

- The value of this input is an integer power of 2.
- The maximum occupancy is 85%.
- The minimum value of this input is `128`.

Data Types: `double`

`SampleRate` — Waveform sample rate

positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- SCS is the subcarrier spacing specified in the `SubcarrierSpacing` property of the `config` input for the first function syntax, or the `scs` input for the other syntaxes.

Data Types: `double`

`Windowing` — Number of time-domain samples for OFDM symbol windowing and overlapping

nonnegative integer (default depends on other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as the comma-separated pair consisting of `'Windowing'` and either a nonnegative integer or `[]`.

If you do not specify this input, or if you specify `'Windowing',[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in Annexes F.5.3 and F.5.4 of TS 38.101-1 [1], Annexes F.5.3 and F.5.4 of TS 38.101-2 [2], and Annexes B.5.2 and C.5.2 Annex TS 38.104 [3]. E is equal to value of `floor`$((N_{CP} - W) \times$ `info.Nfft`$/N_{FFT,\,nominal})$, where

N_{CP}, W, and $N_{FFT,\,nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

`CarrierFrequency` — Carrier frequency
`0` (default) | nonnegative scalar

Carrier frequency, in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a nonnegative scalar. This input corresponds to f_0, defined in TS 38.211 Section 5.4 [4].

Data Types: `double`

Output Arguments

`waveform` — OFDM modulated waveform
complex-valued matrix

OFDM modulated waveform, returned as a complex-valued matrix of size *T*-by-*P*.

- *T* is the number of time-domain samples in the waveform.
- *P* is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`info` — OFDM information
structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive scalar	Waveform sample rate
`CyclicPrefixLengths`	1-by-*N* vector of positive integers, where *N* is the number of OFDM symbols in a subframe.	Cyclic prefix lengths of each OFDM symbol, in samples
`SymbolLengths`	1-by-*N* vector of positive integers	OFDM symbol lengths, in samples
`Windowing`	Positive integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

Fields	Values	Description
SymbolPhases	1-by-*N* vector of scalars in the interval [-π, π]	Phase compensation of each OFDM symbol, in radians The function applies this compensation during modulation to account for phase terms per OFDM symbol, as specified in TS 38.211 Section 5.4 [4]. The `nrOFDMDemodulate` function inverts this phase compensation during demodulation.
SymbolsPerSlot	Positive integer	Number of OFDM symbols in a slot
SlotsPerSubframe	Positive integer	Number of slots in a 1 ms subframe
SlotsPerFrame	Positive integer	Number of slots in a 10 ms frame

Data Types: `struct`

References

[1] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

The `nrb`, `scs`, and `cpl` inputs must be constants. For example, to specify an extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

The `'SampleRate'` name-value-pair argument cannot be used with function syntaxes that use the `carrier` input.

See Also

Functions
nrOFDMDemodulate | nrOFDMInfo | nrResourceGrid

Objects
nrCarrierConfig

Introduced in R2020b

nrPBCH

Generate PBCH modulation symbols

Syntax

```
sym = nrPBCH(cw,ncellid,v)
sym = nrPBCH(cw,ncellid,v,'OutputDataType',datatype)
```

Description

`sym = nrPBCH(cw,ncellid,v)` returns the physical broadcast channel (PBCH) modulation symbols for the physical layer cell identity number `ncellid`. The function implements TS 38.211 Section 7.3.3 [1]. The input `cw` is the BCH codeword, as described in TS 38.212 Section 7.1.5 [2]. The input `v` specifies the scrambling sequence phase.

`sym = nrPBCH(cw,ncellid,v,'OutputDataType',datatype)` specifies the data type of the PBCH symbol.

Examples

Generate Physical Broadcast Channel Symbols

Consider the first Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst and assume that the number of SS/PBCH blocks per half-frame is 4.

```
ssbindex = 0;
v = mod(ssbindex,4);
```

Generate a random sequence of binary values that represent encoded BCH bits. The length of the random sequence corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5.

```
E = 864;
cw = randi([0 1],E,1);
```

Generate the sequence of 432 PBCH quadrature phase shift keying (QPSK) modulation symbols.

```
ncellid = 17;
sym = nrPBCH(cw,ncellid,v);
```

Input Arguments

`cw` — BCH codeword

column vector of binary values

BCH codeword, specified as a column vector of binary values. The size of the vector is $E = 864$, as specified in TS 38.212 Section 7.1.5.

Data Types: `double` | `int8` | `logical`

`ncellid` — Physical layer cell identity number
integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`v` — Scrambling sequence phase
integer from 0 to 7

Scrambling sequence phase, specified as an integer from 0 to 7. `v` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index.

- If the number of SS/PBCH blocks per half-frame is 4, then `v` is the two LSBs of the SS/PBCH block index (0 to 3).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, then `v` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PBCH modulation symbols
complex column vector

PBCH modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
nrPBCHDMRS | nrPBCHDMRSIndices | nrPBCHDecode | nrPBCHIndices | nrPBCHPRBS | nrPRBS | nrPSS | nrSSS

Introduced in R2018b

nrPBCHDecode

Decode PBCH modulation symbols

Syntax

```
cw = nrPBCHDecode(sym,ncellid,v)
cw = nrPBCHDecode(sym,ncellid,v,nVar)
```

Description

`cw = nrPBCHDecode(sym,ncellid,v)` returns a vector of soft bits `cw` resulting from performing the inverse of the physical broadcast channel (PBCH) processing defined in TS 38.211 Section 7.3.3 [1]. `sym` specifies the received PBCH symbols, `ncellid` is the physical layer cell identity number, and `v` specifies the scrambling sequence phase.

`cw = nrPBCHDecode(sym,ncellid,v,nVar)` specifies the noise variance scaling factor of the soft bits in the PBCH demodulation.

Examples

Demodulate Physical Broadcast Channel Symbols

Consider the first Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst and assume that the number of SS/PBCH blocks per half-frame is 4.

```
ssbindex = 0;
v = mod(ssbindex,4);
```

Generate a random sequence of binary values that represent encoded BCH bits. The length of the random sequence corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5.

```
E = 864;
cw = randi([0 1],E,1);
```

Generate the sequence of 432 PBCH quadrature phase shift keying (QPSK) modulation symbols.

```
ncellid = 17;
sym = nrPBCH(cw,ncellid,v);
```

Create bit estimates by demodulating the PBCH symbols. Compare the result with the original input by casting the bit estimates to logical values.

```
rxcw = nrPBCHDecode(sym,ncellid,v);
isequal(cw,rxcw<0)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Received PBCH modulation symbols

complex column vector

Received PBCH modulation symbols, specified as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`v` — Scrambling sequence phase

integer from 0 to 7

Scrambling sequence phase, specified as an integer from 0 to 7. `v` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index.

- If the number of SS/PBCH blocks per half-frame is 4, then `v` is the two LSBs of the SS/PBCH block index (0 to 3).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, then `v` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

`nVar` — Noise variance

`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

Output Arguments

`cw` — Approximate LLR soft bits

column vector of binary values

Approximate log likelihood ratio (LLR) soft bits, returned as a column vector of binary values. The length of `cw` is twice the length of the input `sym`.

Data Types: `double` | `single`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPBCH` | `nrPBCHDMRS` | `nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPBCHPRBS` | `nrPRBS` | `nrPSS` | `nrSSS`

Introduced in R2018b

nrPBCHDMRS

Generate PBCH DM-RS symbols

Syntax

```
sym = nrPBCHDMRS(ncellid,ibar_SSB)
sym = nrPBCHDMRS(ncellid,ibar_SSB,'OutputDataType',datatype)
```

Description

`sym = nrPBCHDMRS(ncellid,ibar_SSB)` returns the physical broadcast channel (PBCH) demodulation reference signal (DM-RS) symbols for the physical layer cell, identified by `ncellid`. The `ibar_SSB` input specifies the time-dependent part of the DM-RS scrambling initialization. The function implements TS 38.211 Section 7.4.1.4.1 [1].

`sym = nrPBCHDMRS(ncellid,ibar_SSB,'OutputDataType',datatype)` specifies the data type of the DM-RS symbol.

Examples

Generate PBCH DM-RS Symbols

Generate the sequence of 144 PBCH DM-RS symbols associated with the third SS block (`i_SSB` = 2) in the second half frame (`n_hf` = 1) of a frame.

```
ncellid = 17;
i_SSB = 2;
n_hf = 1;
ibar_SSB = i_SSB + (4*n_hf);

dmrs = nrPBCHDMRS(ncellid,ibar_SSB);
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`ibar_SSB` — Time-dependent part of DM-RS scrambling initialization

integer from 0 to 7 (default)

Time-dependent part of the DM-RS scrambling initialization, specified as an integer from 0 to 7. `ibar_SSB` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index and the half-frame number.

- If the number of SS/PBCH blocks per half-frame is 4, `ibar_SSB` = i_{SSB} + 4 × n_{hf}, where i_{SSB} is the two LSBs of the SS/PBCH block index (0 to 3). n_{hf} is the half-frame number within the frame (0,1).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, `ibar_SSB` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PBCH DM-RS symbols
complex column vector

PBCH DM-RS symbols, returned as a complex column vector.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCH` | `nrPBCHDMRSIndices` | `nrPRBS` | `nrPSS` | `nrSSS` | `nrSymbolModulate`

Introduced in R2018b

nrPBCHDMRSIndices

Generate PBCH DM-RS resource element indices

Syntax

```
ind = nrPBCHDMRSIndices(ncellid)
ind = nrPBCHDMRSIndices(ncellid,Name,Value)
```

Description

`ind = nrPBCHDMRSIndices(ncellid)` returns the resource element indices for the physical broadcast channel (PBCH) demodulation reference signal (DM-RS). The function implements TS 38.211 Section 7.4.3.1 [1]. The corresponding physical layer cell is identified by `ncellid`. The returned indices are one-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the PBCH DM-RS modulation symbols are mapped.

`ind = nrPBCHDMRSIndices(ncellid,Name,Value)` specifies additional index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get PBCH DM-RS Resource Element Indices

Generate the 144 resource element indices associated with the PBCH DM-RS symbols within a single SS/PBCH block for a given cell identity.

```
ncellid = 17;
indices = nrPBCHDMRSIndices(ncellid)
```

```
indices = 144x1 uint32 column vector

   242
   246
   250
   254
   258
   262
   266
   270
   274
   278
      ⋮
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

Output Arguments

`ind` — PBCH DM-RS resource element indices

column vector | *M*-by-3 matrix

PBCH DM-RS resource element indices, returned as one of the following.

- Column vector — When `'IndexStyle'` is `'index'`.
- *M*-by-3 matrix — When `'IndexStyle'` is `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in a SS/PBCH block, and the number of antennas, respectively.

Depending on `'IndexBase'`, the indices are either one-based or zero-based.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPBCHDMRS` | `nrPBCHIndices` | `nrPSSIndices` | `nrSSSIndices`

Introduced in R2018b

nrPBCHIndices

Generate PBCH resource element indices

Syntax

```
[ind,info] = nrPBCHIndices(ncellid)
[ind,info] = nrPBCHIndices(ncellid,Name,Value)
```

Description

`[ind,info] = nrPBCHIndices(ncellid)` returns the resource element indices `ind` for the physical broadcast channel (PBCH) and related index information `info`. The function implements TS 38.211 Section 7.4.3.1 [1]. The corresponding physical layer cell identity number is `ncellid`. The returned indices are one-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the PBCH modulation symbols are mapped.

`[ind,info] = nrPBCHIndices(ncellid,Name,Value)` specifies additional index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get PBCH Resource Element Indices

Generate the 432 resource element indices associated with the PBCH symbols within a single SS/PBCH block for a given cell identity.

```
ncellid = 17;
indices = nrPBCHIndices(ncellid);
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

IndexStyle — Resource element indexing form
'index' (default) | 'subscript'

Resource element indexing form, specified as the comma-separated pair consisting of 'IndexStyle' and one of these values:

- 'index' — The indices are in linear index form.
- 'subscript' — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: char | string

IndexBase — Resource element indexing base
'1based' (default) | '0based'

Resource element indexing base, specified as the comma-separated pair consisting of 'IndexBase' and one of these values:

- '1based' — The index counting starts from one.
- '0based' — The index counting starts from zero.

Data Types: char | string

Output Arguments

ind — PBCH resource element indices
column vector | *M*-by-3 matrix

PBCH resource element indices, returned as one of the following.

- column vector — When 'IndexStyle' is 'index'.
- *M*-by-3 matrix — When 'IndexStyle' is 'subscript'. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in a SS/PBCH block, and the number of antennas, respectively.

Depending on 'IndexBase', the indices are either one-based or zero-based.

Data Types: uint32

info — Characteristic information about PBCH indices
structure

Characteristic information about PBCH indices, returned as a structure with the following fields.

Parameter Field	Value	Description
G	864	Number of coded and rate matched PBCH data bits.
Gd	432	Number of coded and rate matched PBCH data symbols. Gd is equal to the number of rows in the PBCH indices.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPBCH` | `nrPBCHDMRSIndices` | `nrPSSIndices` | `nrSSSIndices`

Introduced in R2018b

nrPBCHPRBS

Generate PBCH scrambling sequence

Syntax

```
[seq,cinit] = nrPBCHPRBS(ncellid,v,n)
[seq,cinit] = nrPBCHPRBS(ncellid,v,n,Name,Value)
```

Description

`[seq,cinit] = nrPBCHPRBS(ncellid,v,n)` returns the first `n` elements of the physical broadcast channel (PBCH) scrambling sequence. The pseudorandom binary sequence (PRBS) generator is initialized with the physical layer cell identity number `ncellid` and scrambling sequence phase `v`. The function implements TS 38.211 Section 7.3.3.1 [1]. The function also returns the initialization value `cinit` for the PRBS generator.

`[seq,cinit] = nrPBCHPRBS(ncellid,v,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate PBCH Scrambling Sequence

Generate the first 864 outputs of the PBCH scrambling sequence initialized with the specified physical layer cell identity number. The specified length of 864 corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5. Consider the 43rd Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst. Assume that the number of SS/PBCH blocks per half-frame is 64.

```
ncellid = 17;
ssbindex = 42;
v = mod(ssbindex,8); % assume L_max = 64
E = 864;
seq = nrPBCHPRBS(ncellid,v,E);
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`v` — Scrambling sequence phase

integer from 0 to 7

Scrambling sequence phase, specified as an integer from 0 to 7. `v` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index.

- If the number of SS/PBCH blocks per half-frame is 4, then `v` is the two LSBs of the SS/PBCH block index (0 to 3).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, then `v` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

`n` — Number of elements in output sequence

nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MappingType','signed'` specifies nondefault output sequence formatting.

`MappingType` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence

`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — PBCH scrambling sequence

logical column vector | numeric column vector

PBCH scrambling sequence, returned as a logical or numeric column vector. The output `seq` contains the first `n` elements of the PBCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the data type of `seq` is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the data type of `seq` is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator
nonnegative integer from 0 to 1007

Initialization value for PRBS generator, returned as a nonnegative integer from 0 to 1007. `cinit` is the same value as `ncellid`.

Data Types: `double`

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCH` | `nrPBCHDecode` | `nrPBCHIndices` | `nrPRBS`

Introduced in R2018b

nrPDCCH

Generate PDCCH modulation symbols

Syntax

```
sym = nrPDCCH(dcicw,nid,nrnti)
sym = nrPDCCH( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPDCCH(dcicw,nid,nrnti)` returns the physical downlink control channel (PDCCH) modulation symbols, as defined in TS 38.211 Section 7.3.2 [1]. `dcicw` is the encoded downlink control information (DCI) codeword, as specified in TS 38.212 Section 7.3 [2]. The generation process consists of scrambling the input DCI codeword with scrambling identity `nid`, and QPSK symbol modulation. `nrnti` specifies the user equipment (UE).

`sym = nrPDCCH( ___ ,'OutputDataType',datatype)` specifies the PDCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PDCCH Modulation Symbols Using DMRS Scrambling Identity

Specify a random sequence of binary values corresponding to a DCI codeword of 560 bits. Generate modulation symbols by scrambling with the PDCCH demodulation reference signal (DMRS) scrambling identity.

```
dcicw = randi([0 1],560,1);
nid = 2^11;                    % pdcch-DMRS-ScramblingID
nrnti = 123;                   % C-RNTI
sym = nrPDCCH(dcicw,nid,nrnti)
```

```
sym = 280×1 complex

   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
       ⋮
```

Generate PDCCH Modulation Symbols Using NcellID for Scrambling

Specify a random sequence of binary values corresponding to a DCI codeword of 560 bits. Generate PDCCH modulation symbols by setting the scrambling identity to the physical layer cell identity (NcellID).

```
dcicw = randi([0 1],560,1);
nid = 123;                      % NcellID (0 to 1007)
nrnti = 0;
sym = nrPDCCH(dcicw,nid,nrnti)
```

```
sym = 280×1 complex

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Input Arguments

`dcicw` — Encoded DCI codeword

column vector of binary values

Encoded DCI codeword, specified as a column vector of binary values.

Data Types: `double` | `int8` | `logical`

`nid` — Scrambling identity

integer from 0 to 65,535

Scrambling identity, specified as an integer from 0 to 65,535. Specify with `nid` the physical layer cell identity number, ranging from 0 to 1007, or higher layer parameter *pdcch-DMRS-ScramblingID*, ranging from 0 to 65,535. For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — UE identifier

integer from 0 to 65,519

UE identifier, specified as an integer from 0 to 65,519.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a UE-specific search space.
- If `nid` is the physical layer cell identity, `nrnti` is `0`.

For more information, TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PDCCH modulation symbols
complex column vector

PDCCH modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

Compatibility Considerations

Updates to upper limit of UE identifier
Behavior changed in R2020b

Starting in R2020b, the upper limit of UE identifier `nrnti` is 65,519 instead of 65,535.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrDCIDecode` | `nrDCIEncode` | `nrPDCCHDecode` | `nrPDCCHPRBS` | `nrPDCCHResources`

Introduced in R2018b

nrPDCCHDecode

Decode PDCCH modulation symbols

Syntax

```
dcicw = nrPDCCHDecode(sym,nid,nrnti)
dcicw = nrPDCCHDecode(sym,nid,nrnti,nVar)
```

Description

`dcicw = nrPDCCHDecode(sym,nid,nrnti)` returns the soft bits resulting from the inverse operation of the physical downlink control channel (PDCCH) processing specified in TS 38.211 Section 7.3.2 [1]. The decoding consists of the QPSK demodulation of `sym`, and descrambling with the scrambling identity `nid`. The argument `nrnti` specifies the user equipment (UE).

`dcicw = nrPDCCHDecode(sym,nid,nrnti,nVar)` specifies the noise variance scaling factor of the soft bits in the PDCCH demodulation.

Examples

Decode PDCCH Modulation Symbols

Specify a random sequence of binary values corresponding to a DCI codeword of 560 bits. Generate PDCCH modulation symbols by scrambling with the PDCCH demodulation reference signal (DMRS) scrambling identity. Specify the user equipment by using the cell radio network temporary identifier.

```
dcicw = randi([0 1],560,1);
nid = 2^11;                    % pdcch-DMRS-ScramblingID
nrnti = 123;                   % C-RNTI
sym = nrPDCCH(dcicw,nid,nrnti)
```

```
sym = 280×1 complex

   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
       ⋮
```

Demodulate and compare the soft bits with the input codeword.

```
nVar = 0;
rxdcicw = nrPDCCHDecode(sym,nid,nrnti,nVar);
isequal(dcicw,rxdcicw<0)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Received PDCCH modulation symbols

complex column vector

Received PDCCH modulation symbols, specified as a complex column vector.

Data Types: `single` | `double`

`nid` — Scrambling identity

integer from 0 to 65,535

Scrambling identity, specified as an integer from 0 to 65,535. Specify with `nid` the physical layer cell identity number, ranging from 0 to 1007, or higher layer parameter *pdcch-DMRS-ScramblingID*, ranging from 0 to 65,535. For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — UE identifier

integer from 0 to 65,519

UE identifier, specified as an integer from 0 to 65,519.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a UE-specific search space.
- If `nid` is the physical layer cell identity, `nrnti` is `0`.

For more information, TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nVar` — Noise variance

`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

Output Arguments

`dcicw` — Approximate LLR soft bits

column vector of real numbers

Approximate log-likelihood ratio (LLR) soft bits, returned as a column vector of real numbers. `dcicw` inherits the data type of `sym`.

Data Types: `double` | `single`

Compatibility Considerations

Updates to upper limit of UE identifier

Behavior changed in R2020b

Starting in R2020b, the upper limit of UE identifier `nrnti` is 65,519 instead of 65,535.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrDCIDecode` | `nrDCIEncode` | `nrPDCCH` | `nrPDCCHPRBS` | `nrPDCCHResources` | `nrPDCCHSpace`

Introduced in R2018b

nrPDCCHPRBS

Generate PDCCH scrambling sequence

Syntax

```
[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n)
[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n,Name,Value)
```

Description

`[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n)` returns the first `n` elements of the physical downlink control channel (PDCCH) scrambling sequence. The function also returns the initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on the scrambling identity number `nid` and the user equipment (UE) identifier `nrnti`. The function implements TS 38.211 Section 7.3.2.3 [1].

`[seq,cinit] = nrPDCCHPRBS(nid,nrnti,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate PDCCH Scrambling Sequence Using DMRS Scrambling Identity

Generate the first 100 elements of the PDCCH scrambling sequence. The PDCCH demodulation reference signal (DMRS) scrambling identity and the cell radio network temporary identifier determine the initialization value.

```
n = 100;
nid = 10;                 % pdcch-DMRS-ScramblingID
nrnti = 20;               % C-RNTI
seq = nrPDCCHPRBS(nid,nrnti,n);
```

Generate PDCCH Scrambling Sequence Using *NcellID*

Generate the first 120 elements of the PDCCH scrambling sequence initialized with the physical layer cell identity number (*NcellID*).

```
n = 120;
nid = 123;                    % NcellID (0 to 1007)
nrnti = 0;
seq = nrPDCCHPRBS(nid,nrnti,n);
```

Input Arguments

nid — Scrambling identity

integer from 0 to 65,535

Scrambling identity, specified as an integer from 0 to 65,535. Specify with `nid` the physical layer cell identity number, ranging from 0 to 1007, or higher layer parameter *pdcch-DMRS-ScramblingID*, ranging from 0 to 65,535. For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — UE identifier

integer from 0 to 65,519

UE identifier, specified as an integer from 0 to 65,519.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a UE-specific search space.
- If `nid` is the physical layer cell identity, `nrnti` is `0`.

For more information, TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`n` — Number of elements in output sequence

nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MappingType','signed'` specifies nondefault output sequence formatting.

`MappingType` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence

`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — PDCCH scrambling sequence

logical column vector | numeric column vector

PDCCH scrambling sequence, returned as a logical or numeric column vector. `seq` contains the first `n` elements of the PDCCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the output data type is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator

nonnegative integer

Initialization value for the PRBS generator, returned as a nonnegative integer.

Data Types: `double`

Compatibility Considerations

Updates to upper limit of UE identifier

Behavior changed in R2020b

Starting in R2020b, the upper limit of UE identifier `nrnti` is 65,519 instead of 65,535.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDCCH` | `nrPDCCHDecode` | `nrPRBS`

Introduced in R2018b

nrPDCCHResources

Generate PDCCH and PDCCH DM-RS resources

Syntax

```
ind = nrPDCCHResources(carrier,pdcch)
[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch)
[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch,Name,Value)
```

Description

`ind = nrPDCCHResources(carrier,pdcch)` returns physical downlink control channel (PDCCH) resource element indices `ind`, as defined in TS 38.211 Section 7.3.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `pdcch` specifies PDCCH configuration parameters.

`[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch)` also returns PDCCH demodulation reference signal (DM-RS) symbols `dmrsSym` and PDCCH DM-RS resource element indices `dmrsInd`, as defined in TS 38.211 Section 7.4.1.3.

`[ind,dmrsSym,dmrsInd] = nrPDCCHResources(carrier,pdcch,Name,Value)` specifies output formatting options using one or more name-value pair arguments.

Examples

Generate and Map PDCCH Symbols to Carrier Grid

Configure the carrier and the PDCCH with default configuration parameters.

```
carrier = nrCarrierConfig;
pdcch = nrPDCCHConfig;
```

Generate PDCCH symbols for a random DCI codeword by using PDCCH configuration parameters for scrambling and identifying the UE.

```
dciCW = randi([0 1],864,1);
sym = nrPDCCH(dciCW,pdcch.DMRSScramblingID,pdcch.RNTI);
```

Generate PDCCH resource element indices by using the specified carrier and PDCCH objects.

```
ind = nrPDCCHResources(carrier,pdcch);
```

Create a grid for mapping PDCCH symbols to the grid.

```
cgrid = zeros(12*carrier.NSizeGrid,carrier.SymbolsPerSlot);
```

Map PDCCH symbols to the grid.

```
cgrid(ind) = sym;
```

Generate PDCCH DM-RS Symbols and Indices

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH DM-RS symbols and indices for the specified carrier and PDCCH.

```
[~,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
```

Generate PDCCH and DM-RS Indices Relative to BWP Grid

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH resource element indices and DM-RS symbol indices using 1-based, subscript indexing form relative to the BWP grid.

```
[ind,~,dmrsInd] = nrPDCCHResources(carrier,pdcch,...
    'IndexOrientation','bwp','IndexStyle','subscript');
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`pdcch` — PDCCH configuration parameters

`nrPDCCHConfig` object

PDCCH configuration parameters, specified as an `nrPDCCHConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the indexing form and indexing base of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of PDCCH and DM-RS resource elements

`'carrier'` (default) | `'bwp'`

Indexing orientation of PDCCH and DM-RS resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.

- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

`OutputDataType` — Data type of PDCCH DM-RS symbols
`'double'` (default) | `'single'`

Data type of PDCCH DM-RS symbols, specified as the comma-separated pair consisting of `'OutputDataType'` and one of these values:

- `'double'` — Output symbols are of `double` data type.
- `'single'` — Output symbols are of `single` data type.

Data Types: `char` | `string`

Output Arguments

`ind` — PDCCH resource element indices
M-by-1 vector (default) | *M*-by-3 matrix

PDCCH resource element indices, returned as one of these values:

- *M*-by-1 vector — When '`IndexStyle`' is set to `'index'`.
- *M*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

M depends on the aggregation level of the PDCCH and is equal to `pdcch.AggregationLevel` × 6 × 12 × 3 / 4.

Depending on the value of '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

`dmrsSym` — DM-RS symbols
N-by-1 complex vector

DM-RS symbols, returned as an *N*-by-1 complex vector. *N* depends on the aggregation level of the PDCCH and is equal to `pdcch.AggregationLevel` × 6 × 12 × 1 / 4.

Data Types: `single` | `double`

`dmrsInd` — DM-RS resource element indices
N-by-1 vector (default) | *N*-by-3 matrix

DM-RS resource element indices, returned as one of these values:

- *N*-by-1 vector — When '`IndexStyle`' is set to `'index'`.
- *N*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

N depends on the aggregation level of the PDCCH and is equal to `pdcch.AggregationLevel` × 6 × 12 × 1 / 4.

Depending on the value of '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

Compatibility Considerations

DM-RS reference point update for CORESET ID 0

Behavior changed in R2020b

Starting in R2020b, the reference point for the DM-RS sequence-to-subcarrier resource mapping for CORESET ID 0 is the lowest physical resource block of the CORESET instead of CRB 0. This update affects the resources that you generate with this function for CORESET ID 0. For all other CORESET ID values, the reference point remains CRB0.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDCCH` | `nrPDCCHSpace`

Objects

`nrCORESETConfig` | `nrCarrierConfig` | `nrPDCCHConfig` | `nrSearchSpaceConfig`

Topics

“Downlink Control Processing and Procedures”

Introduced in R2020a

nrPDCCHSpace

Generate PDCCH resources for all candidates and aggregation levels

Syntax

```
allInd = nrPDCCHSpace(carrier,pdcch)
[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch)
[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch,Name,Value)
```

Description

`allInd = nrPDCCHSpace(carrier,pdcch)` returns physical downlink control channel (PDCCH) resource element indices `allInd` for all candidates at each aggregation level, as defined in TS 38.211 Section 7.3.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `pdcch` specifies PDCCH configuration parameters.

`[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch)` also returns PDCCH demodulation reference signal (DM-RS) symbols `allDMRSSym` and PDCCH DM-RS resource element indices `allDMRSInd` for all candidates at each aggregation level, as defined in TS 38.211 Section 7.4.1.3.

`[allInd,allDMRSSym,allDMRSInd] = nrPDCCHSpace(carrier,pdcch,Name,Value)` specifies output formatting options using one or more name-value pair arguments.

Examples

Generate PDCCH Indices for All Candidates and Aggregation Levels

Configure the carrier and the PDCCH with default configuration parameters.

```
carrier = nrCarrierConfig;
pdcch = nrPDCCHConfig;
```

Generate all PDCCH resource element indices for all candidates and aggregation levels.

```
allInd = nrPDCCHSpace(carrier,pdcch)
```

```
allInd=5×1 cell array
    { 54x8 uint32}
    {108x8 uint32}
    {216x4 uint32}
    {432x2 uint32}
    {864x1 uint32}
```

Verify that the number of generated candidates for the PDCCH indices at each aggregation level matches the number of candidates specified by the default search space set.

```
numCandidates = [...
    size(allInd{1},2) ...
```

```
    size(allInd{2},2) ...
    size(allInd{3},2) ...
    size(allInd{4},2) ...
    size(allInd{5},2)];
isequaln(pdcch.SearchSpace.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Generate PDCCH DM-RS Symbols for All Candidates and Aggregation Levels

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS symbols for all candidates and aggregation levels.

```
[~,allDMRS] = nrPDCCHSpace(carrier,pdcch)
```

```
allDMRS=5×1 cell array
    { 18x5 double}
    { 36x5 double}
    { 72x3 double}
    {144x2 double}
    {288x1 double}
```

Verify that the number of generated candidates for the PDCCH DM-RS symbols at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRS{1},2) ...
    size(allDMRS{2},2) ...
```

```
    size(allDMRS{3},2) ...
    size(allDMRS{4},2) ...
    size(allDMRS{5},2)];
isequaln(cfgSS.NumCandidates,numCandidates)

ans = logical
   1
```

Generate PDCCH DM-RS Indices for All Candidates and Aggregation Levels

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS resource element indices for all candidates and aggregation levels using 1-based, subscript indexing form relative to the BWP grid.

```
[~,~,allDMRSInd] = nrPDCCHSpace(carrier,pdcch, ...
    'IndexOrientation','bwp','IndexStyle','subscript')

allDMRSInd=5×1 cell array
    { 18x3x5 uint32}
    { 36x3x5 uint32}
    { 72x3x3 uint32}
    {144x3x2 uint32}
    {288x3   uint32}
```

Verify that the number of generated candidates for PDCCH DM-RS indices at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRSInd{1},3) ...
    size(allDMRSInd{2},3) ...
    size(allDMRSInd{3},3) ...
    size(allDMRSInd{4},3) ...
    size(allDMRSInd{5},3)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`pdcch` — PDCCH configuration parameters

`nrPDCCHConfig` object

PDCCH configuration parameters, specified as an `nrPDCCHConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the indexing form and indexing base of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of PDCCH and DM-RS resource elements
`'carrier'` (default) | `'bwp'`

Indexing orientation of PDCCH and DM-RS resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the bandwidth part.

Data Types: `char` | `string`

`OutputDataType` — Data type of PDCCH DM-RS symbols
`'double'` (default) | `'single'`

Data type of PDCCH DM-RS symbols, specified as the comma-separated pair consisting of `'OutputDataType'` and one of these values:

- `'double'` — Output symbols are of `double` data type.
- `'single'` — Output symbols are of `single` data type.

Data Types: `char` | `string`

Output Arguments

`allInd` — PDCCH resource element indices for all candidates at each aggregation level
5-by-1 cell array

PDCCH resource element indices for all candidates at each aggregation level, returned as a 5-by-1 cell array. The five cells correspond to aggregation levels 1, 2, 4, 8, and 16, respectively. Each cell contains a matrix corresponding to all candidates at the appropriate aggregation level. The dimensionality of all matrices is either two or three, depending on the '`IndexStyle`' name-value pair argument. The last dimension of each matrix corresponds to the number of candidates specified by the `pdcch.SearchSpace.NumCandidates` property for the appropriate aggregation level.

Data Types: `uint32`

`allDMRSSym` — PDCCH DM-RS symbols for all candidates at each aggregation level
5-by-1 cell array

PDCCH DM-RS symbols for all candidates at each aggregation level, returned as a 5-by-1 cell array. The five cells correspond to aggregation levels 1, 2, 4, 8, and 16, respectively. Each cell contains a 2-D matrix corresponding to all candidates at the appropriate aggregation level. The number of matrix columns in each cell corresponds to the number of candidates specified by the `pdcch.SearchSpace.NumCandidates` property for the appropriate aggregation level.

Data Types: `single` | `double`

`allDMRSInd` — PDCCH DM-RS resource element indices for all candidates at each aggregation level
5-by-1 cell array

PDCCH DM-RS resource element indices for all candidates at each aggregation level, returned as a 5-by-1 cell array. The five cells correspond to aggregation levels 1, 2, 4, 8, and 16, respectively. Each cell contains a matrix corresponding to all candidates at the appropriate aggregation level. The

dimensionality of all matrices is either two or three, depending on the 'IndexStyle' name-value pair argument. The last dimension of each matrix corresponds to the number of candidates specified by the pdcch.SearchSpace.NumCandidates property for the appropriate aggregation level.

Data Types: uint32

Compatibility Considerations

DM-RS reference point update for CORESET ID 0

Behavior changed in R2020b

Starting in R2020b, the reference point for the DM-RS sequence-to-subcarrier resource mapping for CORESET ID 0 is the lowest physical resource block of the CORESET instead of CRB 0. This update affects the resources that you generate with this function for CORESET ID 0. For all other CORESET ID values, the reference point remains CRB0.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include {coder.Constant('IndexStyle'),coder.Constant('index')} in the -args value of the codegen function. For more information, see the coder.Constant class.

See Also

Functions

nrPDCCH | nrPDCCHDecode | nrPDCCHResources

Objects

nrCORESETConfig | nrCarrierConfig | nrPDCCHConfig | nrSearchSpaceConfig

Topics

“Downlink Control Processing and Procedures”

Introduced in R2020a

nrPDSCH

Generate PDSCH modulation symbols

Syntax

```
sym = nrPDSCH(cws,mod,nlayers,nid,rnti)
sym = nrPDSCH(carrier,pdsch,cws)
sym = nrPDSCH( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPDSCH(cws,mod,nlayers,nid,rnti)` returns `sym` containing physical downlink shared channel (PDSCH) modulation symbols, as defined in TS 38.211 Sections 7.3.1.1–3 [1]. The process consists of scrambling with scrambling identity `nid`, performing symbol modulation with modulation scheme `mod`, and layer mapping. `cws` represents one or two downlink shared channel (DL-SCH) codewords, as described in TS 38.212 Section 7.2.6. `nlayers` specifies the number of transmission layers. `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPDSCH(carrier,pdsch,cws)` returns PDSCH modulation symbols for the specified carrier configuration, `carrier`, and PDSCH configuration, `pdsch`. The input `cws` specifies a downlink shared channel (DL-SCH) codeword.

`sym = nrPDSCH( ___ ,'OutputDataType',datatype)` specifies the PDSCH symbol data type, in addition to the input arguments in any of the previous syntaxes.

Examples

Generate PDSCH Symbols for Single Codeword

Specify a random sequence of binary values corresponding to a codeword of 8000 bits using 256-QAM modulation. Generate PDSCH modulation symbols for the specified physical layer cell identity number, RNTI, and number of transmission layers.

```
modulation = '256QAM';
nlayers = 4;
ncellid = 42;
rnti = 6143;
data = randi([0 1],8000,1);
sym = nrPDSCH(data,modulation,nlayers,ncellid,rnti)
```

```
sym = 250×4 complex

  -0.2301 + 0.5369i  -0.3835 + 0.9971i   0.3835 + 1.1504i  -0.2301 + 0.9971i
   0.8437 - 0.0767i  -0.9971 + 0.6903i  -0.6903 - 0.6903i   0.6903 - 0.6903i
   0.2301 - 1.1504i  -0.9971 + 0.0767i   0.6903 - 1.1504i   1.1504 + 0.6903i
  -0.3835 - 1.1504i  -0.0767 - 0.0767i  -0.3835 + 0.3835i  -0.3835 - 0.3835i
   0.9971 + 0.5369i  -0.3835 - 0.5369i   0.3835 - 0.6903i  -0.3835 - 0.8437i
  -0.0767 + 1.1504i   0.6903 - 0.8437i  -0.2301 + 0.2301i   0.8437 - 0.0767i
  -0.3835 - 1.1504i  -0.6903 - 0.9971i   0.9971 - 0.3835i  -0.9971 + 0.0767i
  -0.0767 + 0.6903i  -0.0767 + 0.8437i   1.1504 + 0.0767i   0.6903 + 1.1504i
```

```
  -0.5369 - 0.9971i  -0.8437 + 0.0767i   0.8437 - 0.3835i  -0.9971 - 1.1504i
   0.2301 - 0.6903i  -0.6903 - 0.5369i  -0.6903 + 1.1504i   0.8437 - 0.2301i
      ⋮
```

Generate PDSCH Symbols for Codewords with Different Modulation Scheme

Specify two random sequences of binary values. The first sequence corresponds to a codeword of 6000 bits using 64-QAM modulation. The second sequence corresponds to a codeword of 8000 bits using 256-QAM modulation. Generate PDSCH modulation symbols for the specified physical layer cell identity number and RNTI using a total of 8 transmission layers.

```
modulation = {'64QAM' '256QAM'};
nlayers = 8;
ncellid = 1;
rnti = 6143;
data = {randi([0 1],6000,1) randi([0 1],8000,1)};
sym = nrPDSCH(data,modulation,nlayers,ncellid,rnti)
```

```
sym = 250×8 complex

  -0.4629 - 0.7715i   0.4629 - 0.4629i   0.4629 + 0.1543i   0.7715 - 1.0801i   0.3835 - 0.9971i
   0.1543 + 0.4629i  -1.0801 + 1.0801i  -0.7715 + 0.7715i  -0.1543 + 0.7715i  -0.2301 + 0.9971i
  -0.1543 + 0.1543i   0.7715 - 1.0801i  -0.4629 + 0.7715i   0.1543 + 1.0801i   0.0767 - 0.8437i
  -0.7715 - 0.4629i  -0.1543 + 0.7715i  -0.7715 - 0.7715i  -0.4629 - 0.1543i  -0.6903 + 0.5369i
   1.0801 - 1.0801i  -1.0801 + 0.7715i   0.1543 - 0.4629i   0.4629 - 0.4629i  -1.1504 + 0.2301i
   0.4629 + 0.4629i   0.1543 + 0.1543i  -0.1543 + 0.1543i   0.1543 - 0.4629i   0.6903 + 0.2301i
  -1.0801 + 0.7715i   0.4629 - 1.0801i   0.4629 + 1.0801i  -0.4629 + 0.4629i  -0.6903 + 0.8437i
  -1.0801 + 0.7715i  -0.1543 - 0.1543i   0.7715 + 1.0801i  -0.4629 - 0.1543i   0.8437 + 0.5369i
  -0.4629 - 1.0801i  -0.7715 - 0.1543i   0.1543 - 1.0801i  -0.1543 + 0.1543i   0.2301 - 0.3835i
   0.7715 + 1.0801i   1.0801 - 0.4629i   1.0801 + 1.0801i  -0.1543 - 1.0801i  -0.0767 + 0.0767i
      ⋮
```

Generate PDSCH Symbols and Indices

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42 and slot number as 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
carrier.NSlot = 10;
```

Create a PDSCH configuration object with a 16-QAM modulation scheme. Set the radio network temporary identifier to 1005, size of the BWP to 25, starting PRB index of the BWP to 10, and PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.Modulation = '16QAM';
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
```

```
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH indices in subscript form and set the index orientation to bandwidth part.

```
[ind,info] = nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 3900x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
      ⋮
```

```
info = struct with fields:
                G: 15600
               Gd: 3900
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PDSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPDSCH(carrier,pdsch,cws,'OutputDataType','single')
```

```
sym = 3900x1 single column vector

  -0.9487 + 0.9487i
  -0.9487 - 0.9487i
  -0.3162 - 0.9487i
   0.9487 - 0.3162i
  -0.9487 + 0.3162i
   0.3162 + 0.9487i
   0.3162 + 0.9487i
  -0.3162 + 0.3162i
   0.3162 + 0.3162i
   0.9487 - 0.3162i
      ⋮
```

Input Arguments

cws — DL-SCH codewords

cell array of binary column vectors | binary column vector

DL-SCH codewords, specified as one of these values:

- Cell array of one or two binary column vectors — Use this value to specify one or two DL-SCH codewords, as described in TS 38.212 Section 7.2.6.
- Binary column vector — Use this value to specify one DL-SCH codeword.

Data Types: `double` | `single` | `cell`

`mod` — Modulation scheme
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. If `cws` contains two codewords, the modulation scheme applies to both codewords. Alternatively, you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Example: To specify different modulation schemes for two codewords, you can use any of these formats: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]`.

Data Types: `char` | `string` | `cell`

`nlayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.

Data Types: `double`

`nid` — Scrambling identity
integer

Scrambling identity, specified as an integer from 0 to 1023. `nid` is the physical layer cell identity number (0 to 1007) or higher layer parameter *dataScramblingIdentityPDSCH* (0 to 1023). For more information, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only `NCellID` property of this `nrCarrierConfig` object.

Property Field	Values	Description
`NCellID`	1 (default), integer from 0 to 1007	Physical layer cell identity

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function uses only these `nrPDSCHConfig` object properties.

Property Field	Values	Description
`Modulation`	'QPSK' (default), '16QAM', '64QAM', or '256QAM', a string scalar, a string array, or a cell array of character vectors	Modulation scheme(s) of codeword(s)
`NumLayers`	1 (default), integer from 1 to 8	Number of transmission layers
`NID`	`[]` (default), integer from 0 to 1023	Scrambling identity, specified as an integer from 0 to 1023. Use `[]` value to allow this property to be equal to `NCellID` of `carrier` input.
`RNTI`	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment

Output Arguments

`sym` — PDSCH modulation symbols
complex matrix

PDSCH modulation symbols, returned as a complex matrix.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrDLSCHInfo` | `nrLayerMap` | `nrPDSCHDecode` | `nrPDSCHIndices` | `nrPDSCHPRBS` | `nrSymbolModulate`

Objects

`nrCarrierConfig` | `nrPDSCHConfig`

Introduced in R2018b

nrPDSCHDecode

Decode PDSCH modulation symbols

Syntax

```
[cws,symbols] = nrPDSCHDecode(sym,mod,nid,rnti)
[cws,symbols] = nrPDSCHDecode(carrier,pdsch,sym)
[cws,symbols] = nrPDSCHDecode( ___ ,nVar)
```

Description

`[cws,symbols] = nrPDSCHDecode(sym,mod,nid,rnti)` returns soft bits `cws` and constellation symbols `symbols` resulting from the inverse operation of the physical downlink shared channel (PDSCH) processing specified in TS 38.211 Sections 7.3.11–3 [1]. The decoding consists of layer demapping, demodulation of `sym` with modulation scheme `mod`, and descrambling with scrambling identity `nid`. The input `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`[cws,symbols] = nrPDSCHDecode(carrier,pdsch,sym)` returns soft bits `cws` and constellation symbols `symbols` for the specified carrier configuration, `carrier`, and PDSCH configuration, `pdsch`. The input `sym` specifies the received PDSCH symbols to decode.

`[cws,symbols] = nrPDSCHDecode( ___ ,nVar)` specifies the noise variance scaling factor of the soft bits in the PDSCH demodulation, in addition to the input arguments in any of the previous syntaxes.

Examples

Decode PDSCH Modulation Symbols

Generate and decode PDSCH modulation symbols.

Specify a random sequence of binary values corresponding to a codeword of 8000 bits using 256-QAM modulation. Generate PDSCH modulation symbols for the specified physical layer cell identity number, RNTI, and number of transmission layers.

```
modulation = '256QAM';
nlayers = 4;
ncellid = 42;
rnti = 6143;
data = randi([0 1],8000,1);
txsym = nrPDSCH(data,modulation,nlayers,ncellid,rnti);
```

Add an additive white Gaussian noise (AWGN) to the PDSCH symbols. Then demodulate to produce soft bit estimates.

```
SNR = 30; % SNR in dB
rxsym = awgn(txsym,SNR);
rxbits = nrPDSCHDecode(rxsym,modulation,ncellid,rnti);
```

Decode PDSCH Symbols Using Configuration Parameters

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
```

Create a PDSCH configuration object with default properties. Set the radio network temporary identifier to 1005, size of the bandwidth part to 25, starting PRB index of the BWP to 10, and the PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH symbols for a single codeword of 8000 bits with the specified carrier configuration and PDSCH configuration.

```
cws = randi([0 1],8000,1);
sym = nrPDSCH(carrier,pdsch,cws);
```

Add an additive white Gaussian noise (AWGN) to the PDSCH symbols. Then demodulate the symbols to produce soft-bit estimates.

```
SNR = 30; % SNR in dB
rxsym = awgn(sym,SNR);
[rxbits,symbols] = nrPDSCHDecode(carrier,pdsch,rxsym);
```

Input Arguments

`sym` — Received PDSCH modulation symbols

complex matrix

Received PDSCH modulation symbols, specified as a complex matrix of size N_{RE}-by-N_{Layers}. N_{RE} is the number of resource elements in a layer, and N_{Layers} is the number of layers. N_{Layers} determines the number of codewords in `cws`.

- If N_{Layers} is from 1 to 4, the function returns one codeword in `cws`.
- If N_{Layers} is from 5 to 8, the function returns two codewords in `cws`.

Data Types: `single` | `double`
Complex Number Support: Yes

`mod` — Modulation scheme

`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. If `cws` contains two codewords, the modulation

scheme applies to both codewords. Alternatively, you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Example: To specify different modulation schemes for two codewords, you can use any of these formats: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]`.

Data Types: `char` | `string` | `cell`

`nid` — Scrambling identity
integer

Scrambling identity, specified as an integer from 0 to 1023. `nid` is the physical layer cell identity number (0 to 1007) or higher layer parameter *dataScramblingIdentityPDSCH* (0 to 1023). For more information, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`nVar` — Noise variance
`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only `NCellID` property of this `nrCarrierConfig` object.

Property Field	Values	Description
NCellID	1 (default), integer from 0 to 1007	Physical layer cell identity

pdsch — PDSCH configuration parameters
nrPDSCHConfig object

PDSCH configuration parameters, specified as an nrPDSCHConfig object. This function uses only these nrPDSCHConfig object properties.

Property Field	Values	Description
Modulation	'QPSK' (default), '16QAM', '64QAM', or '256QAM', a string scalar, a string array, or a cell array of character vectors	Modulation scheme(s) of codeword(s)
NID	[] (default), integer from 0 to 1023	Scrambling identity, specified as an integer from 0 to 1023. Use [] value to allow this property to be equal to NCellID of carrier input.
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment

Output Arguments

cws — Approximate LLR soft bits
cell array of real column vectors

Approximate log likelihood ratio (LLR) soft bits, returned as a cell array of one or two real column vectors. The output cws inherits the data type of sym. The number of column vectors depends on the number layers in sym. The sign of the output represents the hard bits.

Data Types: double | single | cell

symbols — Symbol constellation for each codeword
cell array of one or two column vectors of complex numbers

Symbol constellation for each codeword in cws, returned as a cell array of one or two column vectors of complex numbers. symbols inherits the data type of sym.

Data Types: double | single | cell

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrDLSCHInfo | nrLayerMap | nrPDSCH | nrPDSCHIndices | nrPDSCHPRBS | nrSymbolModulate

Objects
nrCarrierConfig | nrPDSCHConfig

Introduced in R2018b

nrPDSCHDMRS

Generate PDSCH DM-RS symbols

Syntax

```
sym = nrPDSCHDMRS(carrier,pdsch)
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType',datatype)
```

Description

`sym = nrPDSCHDMRS(carrier,pdsch)` returns a matrix containing demodulation reference signal (DM-RS) symbols of physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.1.1 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pdsch` specifies the PDSCH configuration parameters.

`sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType',datatype)` specifies the data type for the DM-RS symbols.

Examples

Generate PDSCH DM-RS Symbols and Indices

Create a carrier configuration object specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a physical downlink shared channel (PDSCH) configuration object, `pdsch`, with physical resource blocks (PRBs) allocated from 0 to 30.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:30;
```

Create a PDSCH demodulation reference signal (DM-RS) object, `dmrs`, with specified properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 2;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 5;
dmrs.NIDNSCID = 10;
dmrs.NSCID = 0;
```

Assign the PDSCH DM-RS configuration object to DMRS property of PDSCH configuration object.

```
pdsch.DMRS = dmrs;
```

Generate PDSCH DM-RS symbols and indices for the specified carrier, PDSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType','single')
```

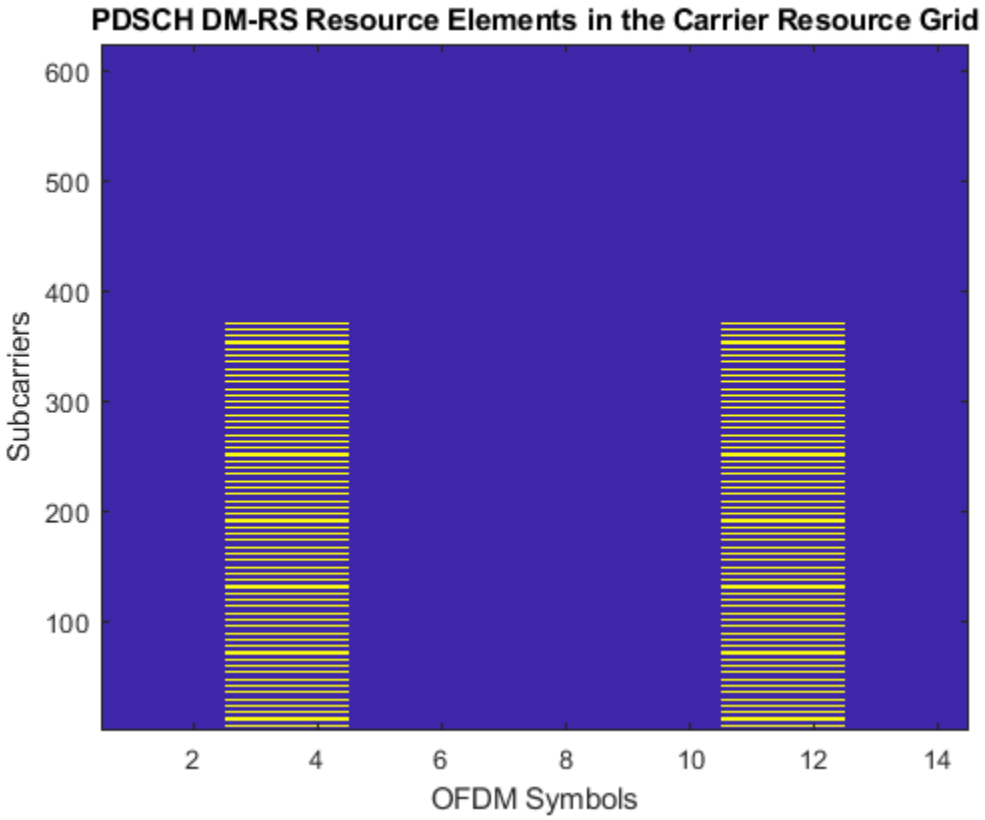
```
sym = 496x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
      ⋮
```

```
ind = nrPDSCHDMRSIndices(carrier,pdsch,'IndexBase','0based','IndexOrientation','carrier')
```

```
ind = 496x1 uint32 column vector

   1252
   1253
   1258
   1259
   1264
   1265
   1270
   1271
   1276
   1277
      ⋮
```

Display the generated DM-RS symbols on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pdsch.NumLayers]));
grid(ind+1) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PDSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`NCellID`	1 (default), integer from 0 to 1007	Physical layer cell identity
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz

Property Field	Values	Description
CyclicPrefix	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • 'normal' corresponds to 14 OFDM symbols in a slot. • 'extended' corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set NSlot to a value larger than the number of slots per frame.

pdsch — PDSCH configuration parameters
nrPDSCHConfig object

PDSCH configuration parameters, specified as an nrPDSCHConfig object. This function uses only these properties of the nrPDSCHConfig object.

Property Field	Values	Description
NSizeBWP	[] (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of [] implies that the value is equal to NSizeGrid of carrier input argument.
NStartBWP	[] (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of [] implies that the value is equal to NStartGrid of carrier input argument.
ReservedPRB	nrPDSCHReservedConfig (default), cell array of nrPDSCHReservedConfig objects	Reserved PRBs and OFDM symbols pattern in BWP, specified as a cell array of objects of class nrPDSCHReservedConfig.
NumLayers	1 (default), integer from 1 to 8	Number of transmission layers. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.
MappingType	'A' (default), 'B'	Mapping type of PDSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PDSCH
PRBSet	[0:51] (default), vector of nonnegative integer from 0 to 274	PRBs allocated for PDSCH within the BWP

Property Field	Values	Description
RNTI	1 (default), integer from 0 to 65535	Radio network temporary identifier of the user equipment
DMRS	`nrPDSCHDMRSConfig` object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSReferencePoint • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet • NIDNSCID • NSCID For more information, see `nrPDSCHDMRSConfig`.

`datatype` — Data type for generated DM-RS symbols
`'double'` (default) | `'single'`

Data type for the generated DM-RS symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — DM-RS symbols
complex matrix

DM-RS symbols, returned as a complex matrix. The number of columns correspond to the number of antenna ports configured.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include

`{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrChannelEstimate` | `nrPDSCH` | `nrPDSCHDMRSIndices` | `nrPDSCHPTRS` | `nrTimingEstimate`

Objects
`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHReservedConfig`

Introduced in R2020a

nrPDSCHDMRSIndices

Generate PDSCH DM-RS indices

Syntax

```
ind = nrPDSCHDMRSIndices(carrier,pdsch)
ind = nrPDSCHDMRSIndices(carrier,pdsch,Name,Value)
```

Description

`ind = nrPDSCHDMRSIndices(carrier,pdsch)` returns a matrix containing demodulation reference signal (DM-RS) resource element (RE) indices of a physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.1.2 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pdsch` specifies the PDSCH configuration parameters. The returned indices are 1-based using linear indexing form.

`ind = nrPDSCHDMRSIndices(carrier,pdsch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PDSCH DM-RS Symbols and Indices

Create a carrier configuration object specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a physical downlink shared channel (PDSCH) configuration object, `pdsch`, with physical resource blocks (PRBs) allocated from 0 to 30.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:30;
```

Create a PDSCH demodulation reference signal (DM-RS) object, `dmrs`, with specified properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 2;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 5;
dmrs.NIDNSCID = 10;
dmrs.NSCID = 0;
```

Assign the PDSCH DM-RS configuration object to DMRS property of PDSCH configuration object.

```
pdsch.DMRS = dmrs;
```

Generate PDSCH DM-RS symbols and indices for the specified carrier, PDSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType','single')
```

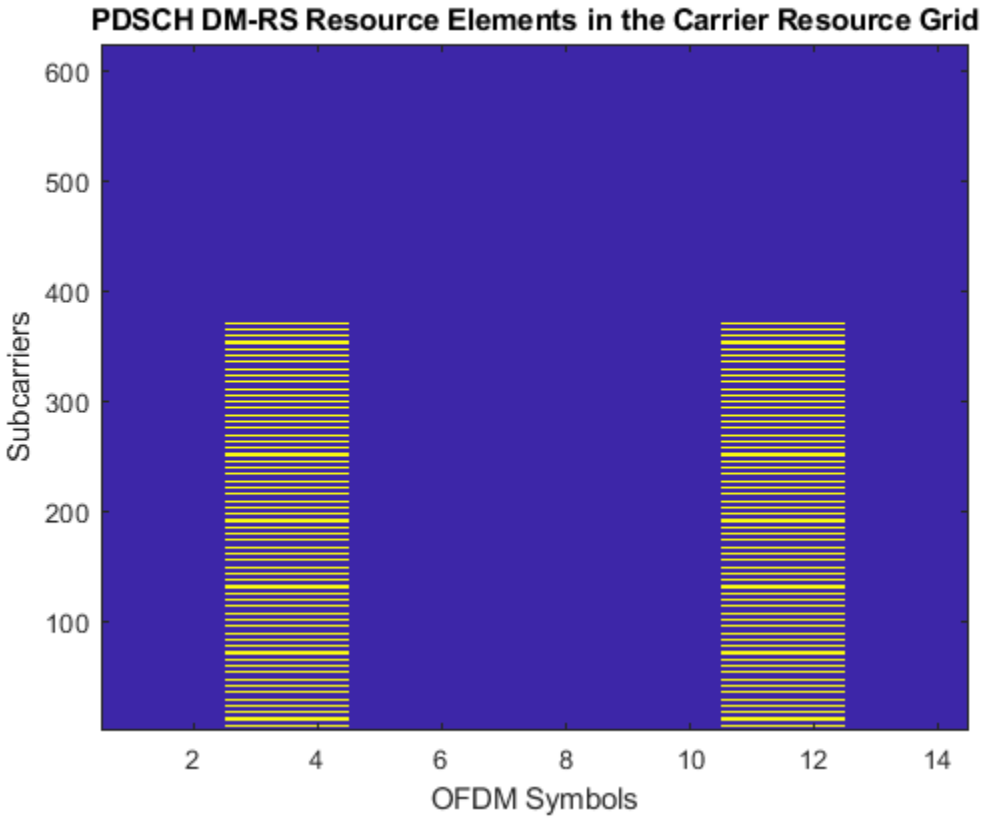
```
sym = 496x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
      ⋮
```

```
ind = nrPDSCHDMRSIndices(carrier,pdsch,'IndexBase','0based','IndexOrientation','carrier')
```

```
ind = 496x1 uint32 column vector

   1252
   1253
   1258
   1259
   1264
   1265
   1270
   1271
   1276
   1277
      ⋮
```

Display the generated DM-RS symbols on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pdsch.NumLayers]));
grid(ind+1) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PDSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz
`CyclicPrefix`	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.

Property Field	Values	Description
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function uses only these properties of the `nrPDSCHConfig` object.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The defualt value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
ReservedPRB	`nrPDSCHReservedConfig` (default), cell array of `nrPDSCHReservedConfig` objects	Reserved PRBs and OFDM symbols pattern in BWP, specified as a cell array of objects of class `nrPDSCHReservedConfig`.
NumLayers	1 (default), integer from 1 to 8	Number of transmission layers. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.
MappingType	'A' (default), 'B'	Mapping type of PDSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PDSCH
PRBSet	[0:51] (default), vector of nonnegative integer from 0 to 274	PRBs allocated for PDSCH within the BWP

Property Field	Values	Description
DMRS	`nrPDSCHDMRSConfig` object	DMRS configuration object only uses these properties. • DMRSConfigurationType • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet For more information, see `nrPDSCHDMRSConfig`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements

`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to carrier grid.
- `'bwp'` — Indices are referenced with respect to bandwidth part.

Data Types: `char` | `string`

Output Arguments

`ind` — DM-RS RE indices

N-by-*P* matrix | *M*-by-3 matrix

DM-RS RE indices, returned as one of these values:

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`. The matrix columns correspond to the antenna ports configured.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrChannelEstimate` | `nrPDSCHDMRS` | `nrPDSCHIndices` | `nrPDSCHPTRSIndices` | `nrTimingEstimate`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHReservedConfig`

Introduced in R2020a

nrPDSCHIndices

Generate PDSCH resource element indices

Syntax

```
[ind,info] = nrPDSCHIndices(carrier,pdsch)
[ind,info] = nrPDSCHIndices(carrier,pdsch,Name,Value)
```

Description

`[ind,info] = nrPDSCHIndices(carrier,pdsch)` returns `ind` in matrix form, which contains 1-based physical downlink shared channel (PDSCH) resource element (RE) indices, as defined in TS 38.211 Sections 7.3.1.5 and 7.3.1.6 [1]. The number of columns in `ind` is equal to the number of antenna ports configured. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pdsch` specifies the PDSCH configuration . The function also returns the structure `info`, which contains additional information about the associated physical reference signals, PDSCH bit capacity, and PDSCH symbol capacity.

`[ind,info] = nrPDSCHIndices(carrier,pdsch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PDSCH Symbols and Indices

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42 and slot number as 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
carrier.NSlot = 10;
```

Create a PDSCH configuration object with a 16-QAM modulation scheme. Set the radio network temporary identifier to 1005, size of the BWP to 25, starting PRB index of the BWP to 10, and PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.Modulation = '16QAM';
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH indices in subscript form and set the index orientation to bandwidth part.

```
[ind,info] = nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 3900x3 uint32 matrix
```

```
    1     1     1
    2     1     1
    3     1     1
    4     1     1
    5     1     1
    6     1     1
    7     1     1
    8     1     1
    9     1     1
   10     1     1
      ⋮
```

```
info = struct with fields:
                 G: 15600
                Gd: 3900
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PDSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPDSCH(carrier,pdsch,cws,'OutputDataType','single')
```

```
sym = 3900x1 single column vector

  -0.9487 + 0.9487i
  -0.9487 - 0.9487i
  -0.3162 - 0.9487i
   0.9487 - 0.3162i
  -0.9487 + 0.3162i
   0.3162 + 0.9487i
   0.3162 + 0.9487i
  -0.3162 + 0.3162i
   0.3162 + 0.3162i
   0.9487 - 0.3162i
      ⋮
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz

Property Field	Values	Description
CyclicPrefix	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function only uses these `nrPDSCHConfig` object properties.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
ReservedPRB	`nrPDSCHReservedConfig` (default), cell array of `nrPDSCHReservedConfig` objects	Reserved PRBs and OFDM symbols pattern in BWP, specified as a cell array of objects, of class `nrPDSCHReservedConfig`.
ReservedRE	`[]` (default), nonnegative integer vector	Reserved resource elements (RE) indices within the BWP
Modulation	'QPSK' (default), '16QAM', '64QAM', or '256QAM', a string scalar, a string array, or a cell array of character vectors	Modulation scheme(s) of codeword(s)
NumLayers	1 (default), integer from 1 to 8	Number of transmission layers. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.

Property Field	Values	Description
MappingType	'A' (default), 'B'	Mapping type of PDSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PDSCH
PRBSet	[0:51] (default), vector of nonnegative integer from 0 to 274	PRBs allocated for PDSCH within the BWP
VRBToPRBInterleaving	0 (default), 1	VRB-to-PRB interleaving, specified as one of these values. • `0` — Disable VRB-to-PRB interleaving. • `1` — Enable VRB-to-PRB interleaving.
VRBBundleSize	2 (default), 4	Bundle size in terms of number of PRBs for VRB-to-PRB interleaving
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment
DMRS	`nrPDSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet • NumCDMGroupsWithoutData For more information, see `nrPDSCHDMRSConfig`.
EnablePTRS	0 (default), 1	PT-RS configuration, specified as one of these values. • `0` — Disable PT-RS configuration. • `1` — Enable PT-RS configuration.
PTRS	`nrPDSCHPTRSConfig` configuration object	PDSCH PT-RS configuration, specified as a `nrPDSCHPTRSConfig` configuration object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements
`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to carrier grid.
- `'bwp'` — Indices are referenced with respect to bandwidth part.

Data Types: `char` | `string`

Output Arguments

`ind` — PDSCH resource element indices
N-by-*P* matrix | *M*-by-3 matrix

PDSCH resource element indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

`info` — PDSCH resource information
structure

PDSCH resource information, returned as a structure containing these fields.

Fields	Description
G	Bit capacity of the PDSCH. This value must be equal to the length of the codeword from the DL-SCH transport channel. Nominally, the value of **G** is set to the `outlen` property of `nrDLSCH` System object.
Gd	Number of resource elements per layer or port
DMRSSymbolSet	The OFDM symbol locations in a slot containing demodulation reference signal (DM-RS) (0-based)
NREPerPRB	Number of REs per PRB allocated to PDSCH. This value excludes any reserved resources.
PTRSSymbolSet	The OFDM symbol locations in a slot containing phase tracking reference signal (PT-RS) (0-based)

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDSCH` | `nrPDSCHDecode`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig` | `nrPDSCHReservedConfig`

Introduced in R2020a

nrPDSCHPTRS

Generate PDSCH PT-RS symbols

Syntax

```
sym = nrPDSCHPTRS(carrier,pdsch)
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType',datatype)
```

Description

`sym = nrPDSCHPTRS(carrier,pdsch)` returns `sym`, which contains phase tracking reference signal (PT-RS) symbols of the physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.2.1 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pdsch` specifies the PDSCH configuration parameters.

`sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType',datatype)` specifies the data type of the output PT-RS symbols.

Examples

Generate PDSCH PT-RS Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a default PDSCH configuration object, and then enable the PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.EnablePTRS = 1;
```

Create a PDSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
```

Assign the PDSCH PT-RS configuration object to PTRS property of PDSCH configuration object.

```
pdsch.PTRS = ptrs;
```

Generate PDSCH PT-RS symbols of data type single.

```
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 78x1 single column vector

  -0.7071 - 0.7071i
```

```
-0.7071 - 0.7071i
 0.7071 - 0.7071i
 0.7071 - 0.7071i
 0.7071 - 0.7071i
-0.7071 - 0.7071i
-0.7071 + 0.7071i
-0.7071 - 0.7071i
-0.7071 + 0.7071i
 0.7071 + 0.7071i
    ⋮
```

Generate PDSCH PT-RS indices in subscript form and set the index orientation to bandwidth part.

```
ind = nrPDSCHPTRSIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 78x3 uint32 matrix

    19     1     1
    67     1     1
   115     1     1
   163     1     1
   211     1     1
   259     1     1
   307     1     1
   355     1     1
   403     1     1
   451     1     1
     ⋮
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`NCellID`	1 (default), integer from 0 to 1007	Physical layer cell identity
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz

Property Field	Values	Description
CyclicPrefix	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function only uses these `nrPDSCHConfig` object properties.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
ReservedPRB	`nrPDSCHReservedConfig` (default), cell array of `nrPDSCHReservedConfig` objects	Reserved PRBs and OFDM symbols pattern in BWP, specified as a cell array of objects, of class `nrPDSCHReservedConfig`.
ReservedRE	`[]` (default), nonnegative integer vector	Reserved resource elements (RE) indices within the BWP
NumLayers	1 (default), integer from 1 to 8	Number of transmission layers. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.
MappingType	'A' (default), 'B'	Mapping type of PDSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PDSCH

Property Field	Values	Description
PRBSet	[0:51] (default), vector of nonnegative integer from 0 to 274	PRBs allocated for PDSCH within the BWP
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment
DMRS	`nrPDSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSReferencePoint • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet • NIDNSCID • NSCID For more information, see `nrPDSCHDMRSConfig`.
EnablePTRS	0 (default), 1	PT-RS configuration, specified as one of these values. • `0` — Disable PT-RS configuration. • `1` — Enable PT-RS configuration.
PTRS	`nrPDSCHPTRSConfig` configuration object	PDSCH PT-RS configuration, specified as a `nrPDSCHPTRSConfig` configuration object.

`datatype` — Data type of generated PT-RS symbols
`'double'` (default) | `'single'`

Data type for the generated PT-RS symbols, specified as 'double' or 'single'.

Data Types: `char` | `string`

Output Arguments

`sym` — PT-RS symbols
complex column vector

PT-RS symbols, returned as a complex column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDSCH` | `nrPDSCHDMRS` | `nrPDSCHDMRSIndices` | `nrPDSCHDecode` | `nrPDSCHIndices` | `nrPDSCHPTRSIndices`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig` | `nrPDSCHReservedConfig`

Introduced in R2020a

nrPDSCHPTRSIndices

Generate PDSCH PT-RS Indices

Syntax

```
ind = nrPDSCHPTRSIndices(carrier,pdsch)
ind = nrPDSCHPTRSIndices(carrier,pdsch,Name,Value)
```

Description

`ind = nrPDSCHPTRSIndices(carrier,pdsch)` returns `ind`, which contains 1-based phase tracking reference signal (PT-RS) resource elements (RE) of the physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.2.2 [1]. The function returns `ind` in linear form, for given carrier configuration `carrier` and physical downlink shared channel configuration `pdsch`.

`ind = nrPDSCHPTRSIndices(carrier,pdsch,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PDSCH PT-RS Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a default PDSCH configuration object, and then enable the PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.EnablePTRS = 1;
```

Create a PDSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
```

Assign the PDSCH PT-RS configuration object to PTRS property of PDSCH configuration object.

```
pdsch.PTRS = ptrs;
```

Generate PDSCH PT-RS symbols of data type single.

```
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 78x1 single column vector

  -0.7071 - 0.7071i
```

```
-0.7071 - 0.7071i
 0.7071 - 0.7071i
 0.7071 - 0.7071i
 0.7071 - 0.7071i
-0.7071 - 0.7071i
-0.7071 + 0.7071i
-0.7071 - 0.7071i
-0.7071 + 0.7071i
 0.7071 + 0.7071i
    ⋮
```

Generate PDSCH PT-RS indices in subscript form and set the index orientation to bandwidth part.

```
ind = nrPDSCHPTRSIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 78x3 uint32 matrix

    19     1     1
    67     1     1
   115     1     1
   163     1     1
   211     1     1
   259     1     1
   307     1     1
   355     1     1
   403     1     1
   451     1     1
    ⋮
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz
`CyclicPrefix`	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.

Property Field	Values	Description
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pdsch` — PDSCH configuration parameters
`nrPDSCHConfig` object

PDSCH configuration parameters, specified as an `nrPDSCHConfig` object. This function only uses these `nrPDSCHConfig` object properties.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
ReservedPRB	`nrPDSCHReservedConfig` (default), cell array of `nrPDSCHReservedConfig` objects	Reserved PRBs and OFDM symbols pattern in BWP, specified as a cell array of objects, of class `nrPDSCHReservedConfig`.
ReservedRE	`[]` (default), nonnegative integer vector	Reserved resource elements (RE) indices within the BWP
NumLayers	1 (default), integer from 1 to 8	Number of transmission layers. For one codeword, use an integer between 1 to 4. For two codewords, use an integer between 5 to 8.
MappingType	'A' (default), 'B'	Mapping type of PDSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PDSCH
PRBSet	[0:51] (default), vector of nonnegative integer from 0 to 274	PRBs allocated for PDSCH within the BWP
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment

Property Field	Values	Description
DMRS	`nrPDSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet For more information, see `nrPDSCHDMRSConfig`.
EnablePTRS	0 (default), 1	PT-RS configuration, specified as one of these values. • `0` — Disable PT-RS configuration. • `1` — Enable PT-RS configuration.
PTRS	`nrPDSCHPTRSConfig` configuration object	PDSCH PT-RS configuration, specified as a `nrPDSCHPTRSConfig` configuration object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements

`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to carrier grid.
- `'bwp'` — Indices are referenced with respect to bandwidth part.

Data Types: `char` | `string`

Output Arguments

`ind` — PT-RS resource element indices

column vector (default) | *M*-by-3 matrix

PT-RS resource element indices, returned as one of these values.

- Column vector — The function returns this type of value when `'IndexStyle'` is set to `'index'`.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDSCH` | `nrPDSCHDMRS` | `nrPDSCHDMRSIndices` | `nrPDSCHDecode` | `nrPDSCHIndices` | `nrPDSCHPTRS`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig` | `nrPDSCHReservedConfig`

Introduced in R2020a

nrPDSCHPRBS

Generate PDSCH scrambling sequence

Syntax

```
[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n)
[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n,Name,Value)
```

Description

`[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n)` returns the first `n` elements of the physical downlink shared channel (PDSCH) scrambling sequence. The function also returns the initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on the scrambling identity number `nid`, the radio network temporary identifier (RNTI) of the user equipment (UE) `rnti`, and the codeword number `q`. The function implements TS 38.211 Section 7.3.1.1 [1].

`[seq,cinit] = nrPDSCHPRBS(nid,rnti,q,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate PDSCH Scrambling Sequence

Generate the first 300 outputs of the PDSCH scrambling sequence when initialized with the specified physical layer cell identity number, RNTI, and codeword number.

```
ncellid = 17;
rnti = 120;
q = 0;
n = 300;
seq = nrPDSCHPRBS(ncellid,rnti,q,n)
```

```
seq = 300x1 logical array

   0
   1
   1
   0
   1
   1
   0
   1
   0
   0
      ⋮
```

Input Arguments

`nid` — Scrambling identity

integer

Scrambling identity, specified as an integer from 0 to 1023. `nid` is the physical layer cell identity number (0 to 1007) or higher layer parameter *dataScramblingIdentityPDSCH* (0 to 1023). For more information, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`q` — Codeword number

`0` | `1`

Codeword number, specified as `0` or `1`.

Data Types: `double`

`n` — Number of elements in output sequence

nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MappingType','signed'` specifies nondefault sequence formatting.

`MappingType` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence

`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — PDSCH scrambling sequence

logical column vector | numeric column vector

PDSCH scrambling sequence, returned as a logical or numeric column vector. `seq` contains the first `n` elements of the PDSCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the output data type is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator

nonnegative integer

Initialization value for PRBS generator, returned as a nonnegative integer.

Data Types: `double`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPDSCH` | `nrPDSCHDecode` | `nrPRBS`

Introduced in R2018b

nrPerfectChannelEstimate

Perfect channel estimation

Syntax

```
h = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,initialSlot)
h = nrPerfectChannelEstimate(carrier,pathGains,pathFilters)
h = nrPerfectChannelEstimate(___,toffset)
h = nrPerfectChannelEstimate(___,toffset,sampleTimes)
h = nrPerfectChannelEstimate(___,cpl)
h = nrPerfectChannelEstimate(___,'CyclicPrefix',cpl)
h = nrPerfectChannelEstimate(___,Name,Value)
```

Description

`h = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,initialSlot)` performs perfect channel estimation. The function first reconstructs the channel impulse response from the channel path gains `pathGains` and the path filter impulse response `pathFilters`. The function then performs orthogonal frequency division multiplexing (OFDM) demodulation for `nrb` number of resource blocks with subcarrier spacing `scs`, and initial slot number `initialSlot`.

`h = nrPerfectChannelEstimate(carrier,pathGains,pathFilters)` specifies carrier configuration parameters.

`h = nrPerfectChannelEstimate(___,toffset)` also specifies the timing offset in addition to the input arguments in any of the previous syntaxes. The timing offset indicates the OFDM demodulation starting point on the reconstructed waveform.

`h = nrPerfectChannelEstimate(___,toffset,sampleTimes)` also specifies the sample times of the channel snapshots in addition to the input arguments in the first and second previous syntaxes.

`h = nrPerfectChannelEstimate(___,cpl)` or `h = nrPerfectChannelEstimate(___,'CyclicPrefix',cpl)` also specifies the cyclic prefix length in addition to the input arguments in all the previous syntaxes that do not include the `carrier` input.

`h = nrPerfectChannelEstimate(___,Name,Value)` specifies options by using one or more name-value pair arguments in addition to any combination of input arguments from the previous syntaxes.

Examples

Plot Estimated Channel Magnitude Response for TDL-C Channel Model

Define a channel configuration structure using an `nrTDLChannel` System object. Use delay profileTDL-C from TR 38.901 Section 7.7.2.

```
SR = 7.68e6;
tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
```

```
tdl.DelaySpread = 100e-9;
tdl.MaximumDopplerShift = 300;
tdl.SampleRate = SR;
```

Create a random waveform with a duration of 1 subframe.

```
T = SR*1e-3;
tdlInfo = info(tdl);
Nt = tdlInfo.NumTransmitAntennas;
in = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel. Obtain the path filters used in channel filtering.

```
[~,pathGains] = tdl(in);
pathFilters = getPathFilters(tdl);
```

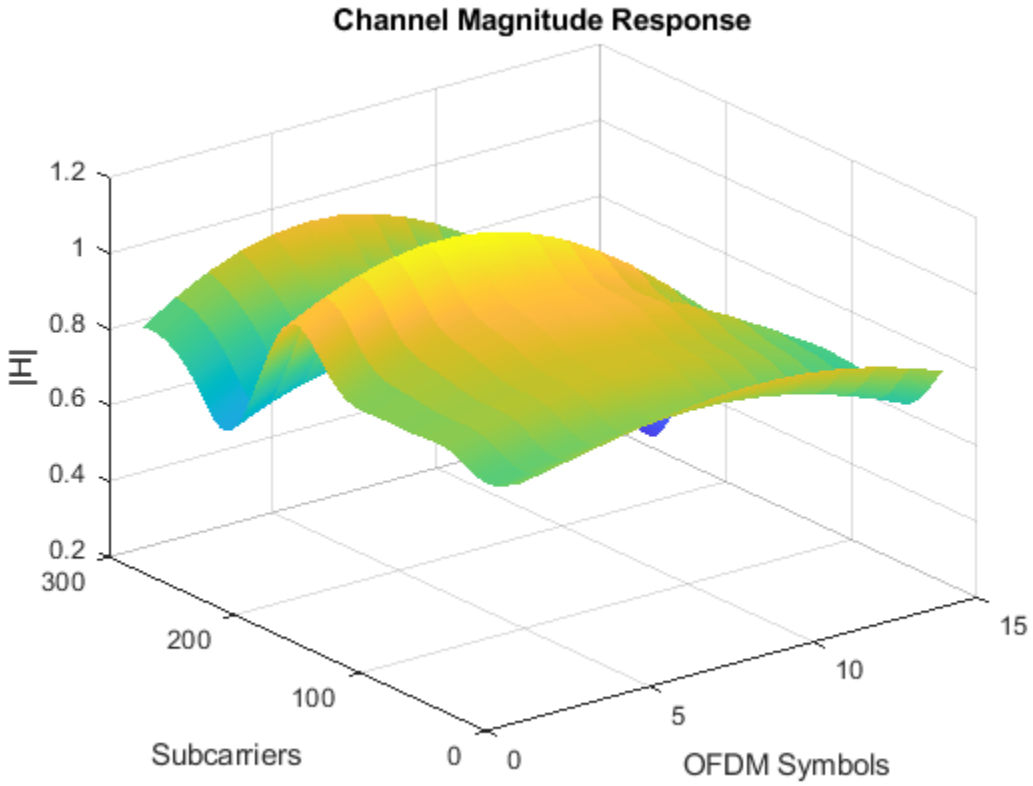
Perform perfect channel estimation using the specified number of blocks, subcarrier spacing, and slot number.

```
NRB = 25;
SCS = 15;
nSlot = 0;

hest = nrPerfectChannelEstimate(pathGains,pathFilters,NRB,SCS,nSlot);
size(hest)
```

```
ans = 1×3

   300    14     2
```

Plot the estimated channel magnitude response for the first receive antenna.

```
figure;
surf(abs(hest(:,:,1)));
shading('flat');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
zlabel('|H|');
title('Channel Magnitude Response');
```



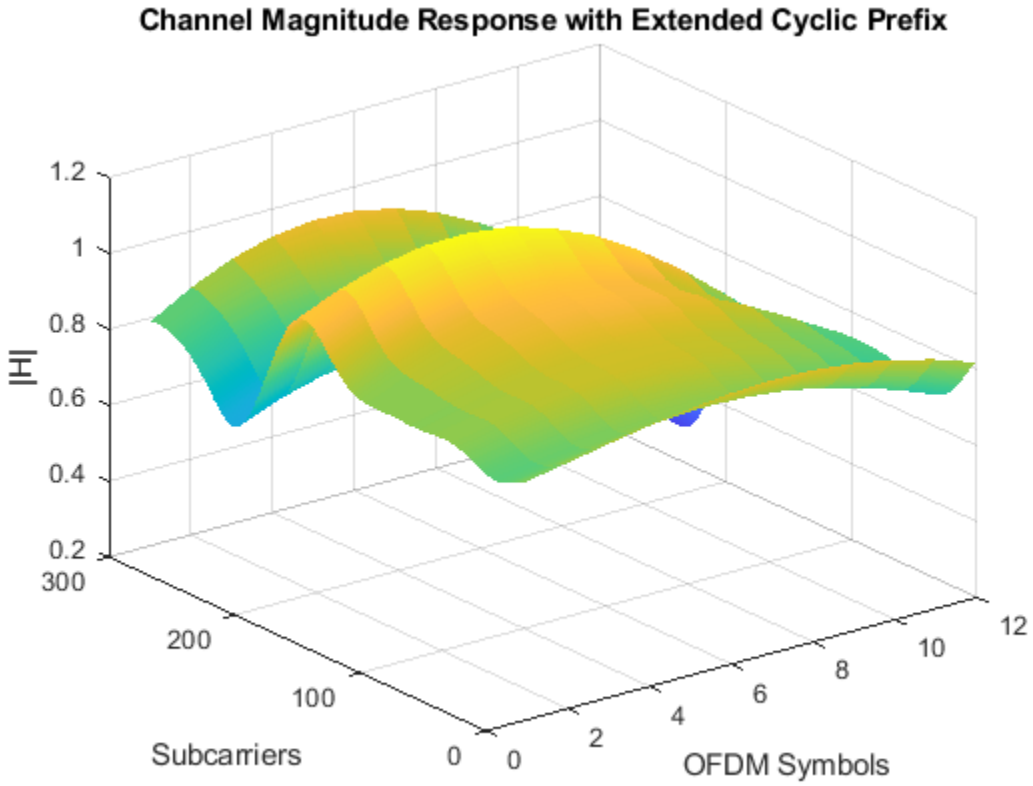
Repeat the channel estimate for extended cyclic prefix.

```
hest = nrPerfectChannelEstimate(pathGains,pathFilters,NRB,SCS, ...
    nSlot,'extended');
size(hest)
```

```
ans = 1×3

   300    12     2
```

Plot the updated results.

```
figure;
surf(abs(hest(:,:,1)));
shading('flat');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
zlabel('|H|');
title('Channel Magnitude Response with Extended Cyclic Prefix');
```



Plot Estimated Channel Magnitude Response for CDL-D Channel Model

Define a channel configuration structure using an `nrCDLChannel` System object. Use delay profile CDL-C from TR 38.901 Section 7.7.1.

```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-D';
cdl.DelaySpread = 30e-9;
cdl.MaximumDopplerShift = 5;
```

Create a random waveform with a duration of 1 subframe.

```
SR = 15.36e6;
T = SR*1e-3;
cdl.SampleRate = SR;
cdlInfo = info(cdl);
Nt = cdlInfo.NumTransmitAntennas;
in = complex(randn(T,Nt),randn(T,Nt));
```

Transmit the input waveform through the channel. Obtain the path filters used in channel filtering.

```
[~,pathGains,sampleTimes] = cdl(in);
pathFilters = getPathFilters(cdl);
```

Perform timing offset estimation using the path filter and path gains.

```
offset = nrPerfectTimingEstimate(pathGains,pathFilters);
```

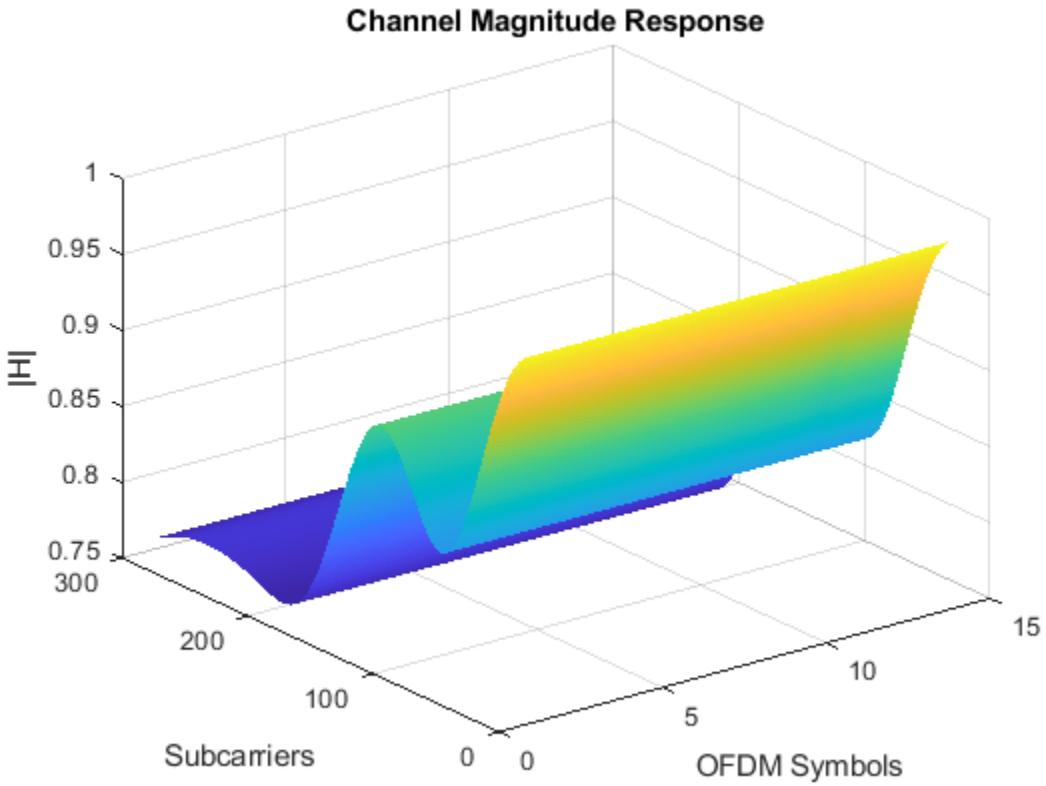
Perform perfect channel estimation. Use the specified number of blocks, subcarrier spacing, slot number, timing offset, and sample times.

```
NRB = 25;
SCS = 15;
nSlot = 0;
hest = nrPerfectChannelEstimate(pathGains,pathFilters,...
    NRB,SCS,nSlot,offset,sampleTimes);
size(hest)
```

```
ans = 1×4

   300    14     2     8
```

Plot the estimated channel magnitude response for the first receive antenna.

```
figure;
surf(abs(hest(:,:,1)));
shading('flat');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
zlabel('|H|');
title('Channel Magnitude Response');
```



Input Arguments

`pathGains` — Channel path gains of fading process

N_{CS}-by-N_P-by-N_T-by-N_R complex matrix

Channel path gains of the fading process, specified as an N_{CS}-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_{CS} is the number of channel snapshots.
- N_P is the number of paths.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathFilters` — Path filter impulse response

N_H-by-N_P real matrix

Path filter impulse response, specified as an N_H-by-N_P real matrix, where:

- N_H is the number of impulse response samples.
- N_P is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

`nrb` — Number of resource blocks

integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing in kHz

`15` | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initialSlot` — Zero-based initial slot number

nonnegative integer

Zero-based initial slot number, specified as a nonnegative integer. The function selects the appropriate cyclic prefix length for the OFDM demodulation based on the value of `initialSlot` modulo the number of slots per subframe.

Data Types: `double`

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid
`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`toffset` — Timing offset in samples
nonnegative integer

Timing offset in samples, specified as a nonnegative integer. The timing offset indicates the OFDM demodulation starting point on the reconstructed waveform. The offset accounts for propagation delays, which is essential when obtaining the perfect estimate of the channel seen by a synchronized receiver. `toffset` defaults to the value `nrPerfectTimingEstimate(pathGains,pathFilters)` when not specified as an input argument.

Data Types: `double`

`sampleTimes` — Sample times of channel snapshots
N_{CS}-by-1 column vector of nonnegative real numbers

Sample times of channel snapshots, specified as an N_{CS}-by-1 column vector of nonnegative real numbers. `sampleTimes` specifies the time of occurrence of each channel snapshot. The number of channel snapshots, N_{CS}, is identical to the first dimension of `pathGains`. When not specified,

`sampleTimes` defaults to an N_{CS}-by-1 vector of times starting at zero with sampling rate used for the OFDM modulation of the number of resource blocks `nrb` and subcarrier spacing `scs`. Ensure that the channel snapshots span at least one slot. The function performs channel estimation for each complete slot.

Data Types: `double`

`cpl` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'CyclicPrefixFraction',0.75` specifies the start location for demodulation relative to the cyclic prefix length.

`Nfft` — Number of FFT points

integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and either a nonnegative integer greater than 127, or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy, defined as the value of $(12 \times N_{RB})/$ `Nfft`, where N_{RB} is the number of resource blocks, of 100%.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets a default value satisfying these conditions.

- The value of this input is an integer power of 2.
- The maximum occupancy is 85%.
- The minimum value of this input is 128.

Data Types: `double`

`SampleRate` — Waveform sample rate

positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{fft} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- *SCS* is the subcarrier spacing specified in the `SubcarrierSpacing` property of the `config` input for the first function syntax, or the `scs` input for the other syntaxes.

Data Types: `double`

`CyclicPrefixFraction` — FFT window position within cyclic prefix
`0.5` (default) | scalar in the interval [0, 1]

Fast Fourier transform (FFT) window position within the cyclic prefix, specified as the comma-separated pair consisting of `'CyclicPrefixFraction'` and a scalar in the interval [0, 1].

The value that you specify indicates the start location for OFDM demodulation relative to the beginning of the cyclic prefix.

Data Types: `double`

Output Arguments

`h` — Perfect channel estimate
N_{SC}-by-N_{SYM}-by-N_R-by-N_T complex array

Perfect channel estimate, returned as an N_{SC}-by-N_{SYM}-by-N_R-by-N_T complex array, where:

- N_{SC} is the number of subcarriers.
- N_{SYM} is the number of OFDM symbols.
- N_R is the number of receive antennas.
- N_T is the number of transmit antennas.

`h` inherits its data type from `pathGains`.

Data Types: `double` | `single`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

The `nrb`, `scs`, and `cpl` inputs must be constant when you specify any name-value pair arguments. For example, to specify an extended cyclic prefix, include `{coder.Constant('CyclicPrefix'),coder.Constant('Extended')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

The `'SampleRate'` name-value-pair argument cannot be used with function syntaxes that use the `carrier` input.

See Also

Functions
nrChannelEstimate | nrPerfectTimingEstimate | nrTimingEstimate

Objects
nrCDLChannel | nrCarrierConfig | nrTDLChannel

Introduced in R2018b

nrPerfectTimingEstimate

Perfect timing estimation

Syntax

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

Description

`[offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters)` performs perfect timing estimation. To find the peak of the channel impulse response, the function first reconstructs the impulse response from the channel path gains `pathGains` and the path filter impulse response `pathFilters`. The channel impulse response is averaged across all channel snapshots and summed across all transmit and receive antennas before timing estimation. The function returns the estimated timing offset `offset` and the channel impulse response magnitude `mag`.

Examples

Plot Channel Impulse Magnitude and Timing Offset for TDL-C Channel Model

Define a channel configuration structure using an `nrTDLChannel` System object. Use delay profile TDL-C from TR 38.901 Section 7.7.2.

```
tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
tdl.DelaySpread = 100e-9;
```

Create a random waveform with a duration of 1 subframe.

```
tdlInfo = info(tdl);
Nt = tdlInfo.NumTransmitAntennas;
in = complex(zeros(100,Nt),zeros(100,Nt));
```

Transmit the input waveform through the channel.

```
[~,pathGains] = tdl(in);
```

Obtain the path filters used in channel filtering.

```
pathFilters = getPathFilters(tdl);
```

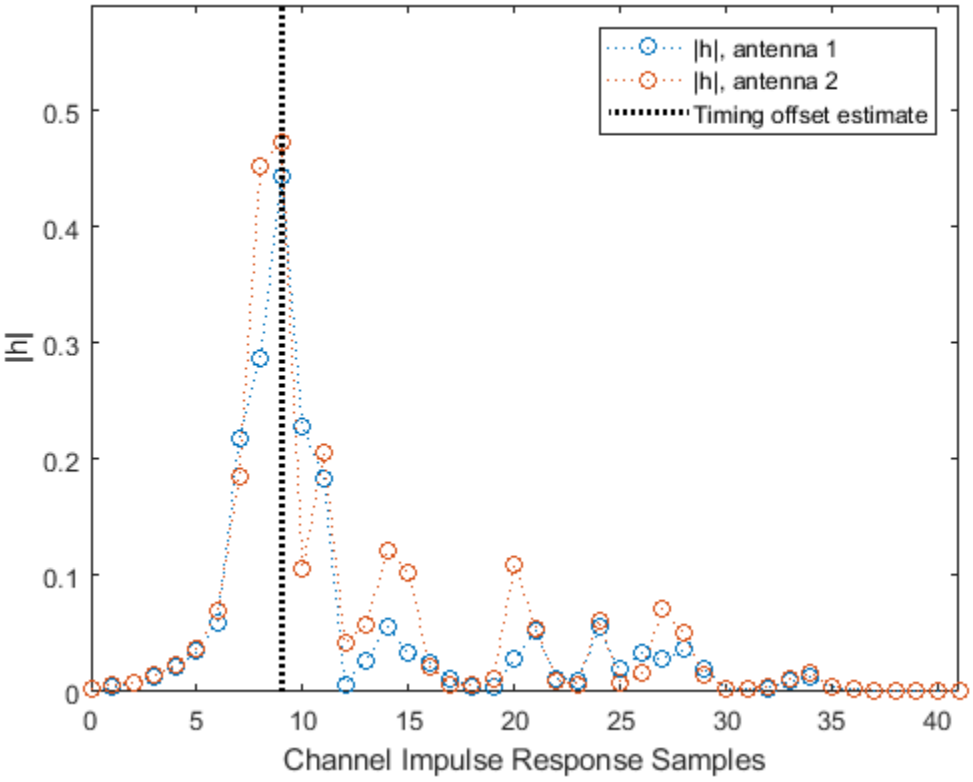
Estimate timing offset.

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

Plot the magnitude of the channel impulse response and the timing offset estimate.

```
[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');
```



Input Arguments

`pathGains` — Channel path gains of fading process

N_{CS}-by-N_P-by-N_T-by-N_R complex matrix

Channel path gains of the fading process, specified as an N_{CS}-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_{CS} is the number of channel snapshots.
- N_P is the number of paths.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathFilters` — Path filter impulse response

N_H-by-N_P real matrix

Path filter impulse response, specified as an N_H-by-N_P real matrix, where:

- N_H is the number of impulse response samples.
- N_P is the number of paths.

Each column of the matrix contains the filter impulse response for each path of the delay profile.

Data Types: `double`

Output Arguments

`offset` — Timing offset in samples

nonnegative integer

Timing offset in samples, returned as a nonnegative integer. The number of samples is relative to the first sample of the channel impulse response reconstructed from `pathGains` and `pathFilters`.

Data Types: `double`

`mag` — Channel impulse response magnitude

N_H-by-N_R real matrix

Channel impulse response magnitude for each receive antenna, returned as an N_H-by-N_R real matrix.

- N_H is the number of impulse response samples.
- N_R is the number of receive antennas.

`mag` inherits its data type from `pathGains`.

Data Types: `single` | `double`

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrChannelEstimate` | `nrPerfectChannelEstimate` | `nrTimingEstimate`

Objects

`nrCDLChannel` | `nrTDLChannel`

Introduced in R2018b

nrPolarDecode

Polar decoding

Syntax

```
decbits = nrPolarDecode(rec,K,E,L)
decbits = nrPolarDecode(rec,K,E,L,padCRC)
decbits = nrPolarDecode(rec,K,E,L,padCRC,rnti)
decbits = nrPolarDecode(rec,K,E,L,nmax,iil,CRClen)
```

Description

`decbits = nrPolarDecode(rec,K,E,L)` decodes the rate-recovered input `rec` for an (*N*,K) polar code, where *N* is the length of `rec` and K is the length of decoded bits `decbits`, as specified in TS 38.212 Section 5 [1]. The function uses a cyclic redundancy check (CRC)-aided successive-cancellation list decoder of length `L`. By default, output deinterleaving is enabled, the maximum length of the input is 512, and the number of appended CRC bits is 24. Use this syntax for downlink configuration.

`decbits = nrPolarDecode(rec,K,E,L,padCRC)` specifies whether the information block on the transmit end was prepadded with ones before CRC encoding.

`decbits = nrPolarDecode(rec,K,E,L,padCRC,rnti)` specifies a radio network temporary identifier (RNTI). You can use this syntax when the value of `rnti` masks the CRC parity bits at the transmit end.

`decbits = nrPolarDecode(rec,K,E,L,nmax,iil,CRClen)` decodes the input with a specified maximum length of 2^{nmax} , output deinterleaving specified by `iil`, and number of appended CRC bits specified by `CRClen`. This syntax assumes that the information block on the transmit end was not prepadded with ones before CRC encoding and that the RNTI is equal to `0`.

- For downlink (DL) configuration, valid values for `nmax`, `iil`, and `CRClen` are 9, `true`, and 24, respectively.
- For uplink (UL) configuration, valid values for `nmax` and `iil` are 10 and `false`, respectively, and for `CRClen` is 11 or 6.

Examples

Transmit and Decode Polar Encoded Data

Transmit polar-encoded block of data and decode it using successive-cancellation list decoder.

Initial Setup

Create a channel that adds white Gaussian noise (WGN) to an input signal. Set the noise variance to 1.5.

```
nVar = 1.5;
chan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Create a binary phase shift keying (BSPK) modulator and demodulator.

```
bpskMod = comm.BPSKModulator;
bpskDemod = comm.BPSKDemodulator('DecisionMethod', ...
    'Approximate log-likelihood ratio','Variance',nVar);
```

Simulate a Frame

Perform polar encoding of a random message of length `K`. The rate-matched output is of length `E`.

```
K = 132;
E = 256;
msg = randi([0 1],K,1,'int8');
enc = nrPolarEncode(msg,E);
```

Modulate the polar encoded data using BSPK modulation, add WGN, and demodulate.

```
mod = bpskMod(enc);
rSig = chan(mod);
rxLLR = bpskDemod(rSig);
```

Perform polar decoding using successive-cancellation list decoder of length `L`.

```
L = 8;
rxBits = nrPolarDecode(rxLLR,K,E,L);
```

Determine the number of bit errors.

```
numBitErrs = biterr(rxBits,msg);
disp(['Number of bit errors: ' num2str(numBitErrs)])
```

```
Number of bit errors: 0
```

The transmitted and received messages are identical.

Input Arguments

`rec` — Rate-recovered input

column vector of real values

Rate-recovered input, specified as a column vector of real values. The input `rec` represents the log-likelihood ratios per bit with a negative bipolar mapping. So a `0` is mapped to `1`, and a `1` is mapped to `-1`. The length of `rec` must be a power of two.

Data Types: `single` | `double`

`K` — Length of information block in bits

positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable.

Data Types: `double`

`E` — Rate-matched output length in bits

positive integer

Rate-matched output length in bits, specified as a positive integer.

- If $18 \le K \le 25$, `E` must be in the range $K + 3 < E \le 8192$.
- If $K > 30$, `E` must be in the range $K < E \le 8192$.

Data Types: `double`

`L` — Length of decoding list
power of two

Length of decoding list, specified as a power of two.

Data Types: `double`

`padCRC` — Prepadding before CRC encoding
`false` (default) | `true`

Prepadding before CRC encoding, specified as `false` or `true`. Set `padCRC` to `true` if the information block on the transmit end, before polar encoding, was prepadded with all ones before CRC encoding.

Data Types: `logical`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`nmax` — Base-2 logarithm of rate-recovered input's maximum length
`9` (default) | `10`

Base-2 logarithm of rate-recovered input's maximum length, specified as `9` or `10`.

- For DL configuration, specify `9`.
- For UL configuration, specify `10`.

If N is the length of `rec` in bits, $N \le 2^{nmax}$, see TS 38.212 Section 5.3.1.2.

Data Types: `double`

`iil` — Output deinterleaving
`true` (default) | `false`

Output deinterleaving, specified as `true` or `false`.

- For DL configuration, specify `true`.
- For UL configuration, specify `false`.

Data Types: `logical`

`CRClen` — Number of appended CRC bits
`24` (default) | `11` | `6`

Number of appended CRC bits, specified as `24`, `11`, or `6`.

- For DL configuration, specify `24`.
- For UL configuration, specify `11` or `6`.

The numbers 24, 11, and 6 correspond to the polynomials gCRC24C, gCRC11, and gCRC6, respectively, as described in TS 38.212. Section 5.1 [1].

Data Types: `double`

Output Arguments

`decbits` — Decoded message

column vector of binary values

Decoded message, returned as a K-by-1 column vector of binary values.

Data Types: `int8`

Compatibility Considerations

Polar decoding metric update

Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrPolarDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] Tal, I. and Vardy, A., “List decoding of Polar Codes“, *IEEE Transactions on Information Theory*. Vol. 61, No. 5, pp. 2213-2226, May 2015.

[3] Niu, K., and Chen, K., “CRC-Aided Decoding of Polar Codes“, *IEEE Communications Letters*, Vol. 16, No. 10, pp. 1668-1671, Oct. 2012.

[4] Stimming, A. B., Parizi, M. B., and Burg, A., “LLR-Based Successive Cancellation List Decoding of Polar Codes“, *IEEE Transaction on Signal Processing*, Vol. 63, No. 19, pp.5165-5179, 2015.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

The input argument `L` must be a compile-time constant. Include `{coder.Constant(L)}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrCRCDecode` | `nrDCIDecode` | `nrPolarEncode` | `nrRateRecoverPolar` | `nrUCIDecode`

Topics
“5G New Radio Polar Coding”

Introduced in R2018b

nrPolarEncode

Polar encoding

Syntax

```
enc = nrPolarEncode(in,E)
enc = nrPolarEncode(in,E,nmax,iil)
```

Description

`enc = nrPolarEncode(in,E)` returns the polar-encoded output for the input message `in` and rate-matched output length `E` as specified in TS 38.212 Section 5 [1]. By default, input interleaving is enabled and the maximum length of the encoded message is 512. Use this syntax for downlink configuration.

`enc = nrPolarEncode(in,E,nmax,iil)` encodes the input with a specified maximum length of 2^{nmax} and input interleaving specified by `iil`.

- For downlink (DL) configuration, valid values for `nmax` and `iil` are 9 and `true`, respectively.
- For uplink (UL) configuration, valid values for `nmax` and `iil` are 10 and `false`, respectively.

Examples

Perform Polar Encoding

Perform polar encoding of a random message of length `K`. `E` specifies the length of the rate-matched output which is different from the length of the encoded message `enc`. The length of `enc` is always a power of two.

```
K = 132;
E = 300;
msg = randi([0 1],K,1,'int8');
enc = nrPolarEncode(msg,E)
```

```
enc = 512x1 int8 column vector

   0
   0
   0
   0
   0
   0
   1
   1
   1
   0
      ⋮
```

Transmit and Decode Polar Encoded Data

Transmit polar-encoded block of data and decode it using successive-cancellation list decoder.

Initial Setup

Create a channel that adds white Gaussian noise (WGN) to an input signal. Set the noise variance to 1.5.

```
nVar = 1.5;
chan = comm.AWGNChannel('NoiseMethod','Variance','Variance',nVar);
```

Create a binary phase shift keying (BSPK) modulator and demodulator.

```
bpskMod = comm.BPSKModulator;
bpskDemod = comm.BPSKDemodulator('DecisionMethod', ...
    'Approximate log-likelihood ratio','Variance',nVar);
```

Simulate a Frame

Perform polar encoding of a random message of length `K`. The rate-matched output is of length `E`.

```
K = 132;
E = 256;
msg = randi([0 1],K,1,'int8');
enc = nrPolarEncode(msg,E);
```

Modulate the polar encoded data using BSPK modulation, add WGN, and demodulate.

```
mod = bpskMod(enc);
rSig = chan(mod);
rxLLR = bpskDemod(rSig);
```

Perform polar decoding using successive-cancellation list decoder of length `L`.

```
L = 8;
rxBits = nrPolarDecode(rxLLR,K,E,L);
```

Determine the number of bit errors.

```
numBitErrs = biterr(rxBits,msg);
disp(['Number of bit errors: ' num2str(numBitErrs)])
```

```
Number of bit errors: 0
```

The transmitted and received messages are identical.

Input Arguments

`in` — Input message

column vector of binary values

Input message, specified as a column vector of binary values. `in` includes the CRC bits if applicable.

Data Types: `double` | `int8`

`E` — Rate-matched output length in bits

positive integer

Rate-matched output length in bits, specified as a positive integer. `E` depends on K, the length of the input message `in`.

- If $18 \le K \le 25$, `E` must be in the range $K + 3 < E \le 8192$.
- If $K > 30$, `E` must be in the range $K < E \le 8192$.

Data Types: `double`

`nmax` — Base-2 logarithm of the encoded message's maximum length

`9` (default) | `10`

Base-2 logarithm of the encoded message's maximum length, specified as `9` or `10`.

- For DL configuration, specify `9`.
- For UL configuration, specify `10`.

If N is the length of the polar-encoded message in bits, then $N \le 2^{nmax}$. See TS 38.212 Section 5.3.1.2 [1].

Data Types: `double`

`iil` — Input interleaving

`true` (default) | `false`

Input interleaving, specified as `true` or `false`.

- For DL configuration, specify `true`.
- For UL configuration, specify `false`.

Data Types: `logical`

Output Arguments

`enc` — Polar-encoded message

column vector of binary values

Polar-encoded message, returned as a column vector of binary values. `enc` inherits its data type from the input message `in`.

The length of the polar-encoded message, N, is a power of two. For more information, see TS 38.212 Section 5.3.1.

- For DL configuration, $N \le 512$.
- For UL configuration, $N \le 1024$.

Data Types: `double` | `int8`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*..

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCRCEncode` | `nrDCIEncode` | `nrPolarDecode` | `nrRateMatchPolar` | `nrUCIEncode`

Topics

“5G New Radio Polar Coding”

Introduced in R2018b

nrPRACH

Generate PRACH symbols

Syntax

```
[sym,info] = nrPRACH(carrier,prach)
[sym,info] = nrPRACH(carrier,prach,'OutputDataType',datatype)
```

Description

`[sym,info] = nrPRACH(carrier,prach)` returns the physical random access channel (PRACH) symbols, as defined in TS 38.211 Section 6.3.3 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `prach` specifies PRACH configuration parameters. The function also returns the structure `info`, which contains carrier-dependent information about the PRACH.

`[sym,info] = nrPRACH(carrier,prach,'OutputDataType',datatype)` specifies the data type of the PRACH symbols.

Examples

Generate and Map PRACH Symbols to Grid

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach = nrPRACHConfig;
```

Generate PRACH symbols and indices using the specified carrier and PRACH configuration parameters.

```
prachSym = nrPRACH(carrier,prach);
prachInd = nrPRACHIndices(carrier,prach);
```

Generate a PRACH resource grid of all zeros.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid by using the indices.

```
prachGrid(prachInd) = prachSym;
```

Analyze PRACH Root Sequence Indices

Analyze physical root Zadoff-Chu sequence indices by generating 64 orthogonal PRACH preambles for two different PRACH configurations.

Root Sequence Indices with Single Value

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach1 = nrPRACHConfig;
```

Set the PRACH logical root sequence index to `0`. For this value, the physical root sequence index is `129`, as defined in TS 38.211 Table 6.3.3.1-3.

```
prach1.SequenceIndex = 0;
```

Set the PRACH cyclic shift configuration index to `1`. For this value, each PRACH preamble has a different cyclic shift value, based on N_{CS} from TS 38.211 Table 6.3.3.1-5.

```
prach1.ZeroCorrelationZone = 1;
```

Generate 64 PRACH preambles to store the physical root sequence indices and cyclic shift values.

```
rootSequence1 = NaN(1,64);
cyclicShift1 = NaN(1,64);
for preambleIndex = 0:63
    prach1.PreambleIndex = preambleIndex;
    [~,info] = nrPRACH(carrier,prach1);
    rootSequence1(preambleIndex+1) = info.RootSequence;
    cyclicShift1(preambleIndex+1) = info.CyclicShift;
end
```

Verify that in each preamble, the physical root sequence index is `129`, which is the expected value from configuring the logical root sequence index to `0`.

```
disp(rootSequence1)
```

```
  Columns 1 through 13

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 14 through 26

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 27 through 39

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 40 through 52

   129   129   129   129   129   129   129   129   129   129   129   129   129

  Columns 53 through 64

   129   129   129   129   129   129   129   129   129   129   129   129
```

Verify that each preamble has a different cyclic shift value.

```
disp(cyclicShift1)
```

```
  Columns 1 through 13
```

```
     0    13    26    39    52    65    78    91   104   117   130   143   156

  Columns 14 through 26

   169   182   195   208   221   234   247   260   273   286   299   312   325

  Columns 27 through 39

   338   351   364   377   390   403   416   429   442   455   468   481   494

  Columns 40 through 52

   507   520   533   546   559   572   585   598   611   624   637   650   663

  Columns 53 through 64

   676   689   702   715   728   741   754   767   780   793   806   819
```

Root Sequence Indices with Different Values

Configure another PRACH with default properties.

```
prach2 = nrPRACHConfig;
```

Set the PRACH logical root sequence index to `0`. For this value, the physical root sequence index is `129`, as defined in TS 38.211 Table 6.3.3.1-3.

```
prach2.SequenceIndex = 0;
```

Set the PRACH cyclic shift configuration index to `0`. For this value, each PRACH preamble has the same cyclic shift value, equal to `0`, based on TS 38.211 Table 6.3.3.1-5.

```
prach2.ZeroCorrelationZone = 0;
```

Generate 64 PRACH preambles to store the physical root sequence indices and cyclic shift values.

```
rootSequence2 = NaN(1,64);
cyclicShift2 = NaN(1,64);
for preambleIndex = 0:63
    prach2.PreambleIndex = preambleIndex;
    [~,info] = nrPRACH(carrier,prach2);
    rootSequence2(preambleIndex+1) = info.RootSequence;
    cyclicShift2(preambleIndex+1) = info.CyclicShift;
end
```

Check the physical root sequence indices and cyclic shift values. Even though the logical root sequence index, `prach.SequenceIndex`, is `0`, not each physical root sequence index value is the expected value of `129`. Because the cyclic shift value is zero in each preamble, the function `nrPRACH` obtains the physical root sequence indices by taking consecutive logical index values. The returned physical root sequence indices correspond to logical indices 0 to 63 from TS 38.211 Table 6.3.3.1-3.

```
disp(rootSequence2)

  Columns 1 through 13

   129   710   140   699   120   719   210   629   168   671    84   755   105

  Columns 14 through 26
```

```
   734    93   746    70   769    60   779     2   837     1   838    56   783

  Columns 27 through 39

   112   727   148   691    80   759    42   797    40   799    35   804    73

  Columns 40 through 52

   766   146   693    31   808    28   811    30   809    27   812    29   810

  Columns 53 through 64

    24   815    48   791    68   771    74   765   178   661   136   703

disp(cyclicShift2)

  Columns 1 through 13

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 14 through 26

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 27 through 39

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 40 through 52

     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 53 through 64

     0     0     0     0     0     0     0     0     0     0     0     0
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`prach` — PRACH configuration parameters

`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object.

`datatype` — Data type of output symbols

`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PRACH symbols

complex column vector | `[]`

PRACH symbols, returned as a complex column vector or an empty array. The number of symbols depends on the PRACH configuration `prach`. The function returns an empty array when the PRACH preamble is not active in the current slot.

Data Types: `single` | `double`

`info` — Carrier-dependent PRACH information

structure

Carrier-dependent PRACH information, returned as a structure containing these fields:

Fields	Description
`RootSequence`	Index or indices of physical root Zadoff-Chu sequence
`CyclicShift`	Cyclic shift or shifts of Zadoff-Chu sequence
`CyclicOffset`	Cyclic shift or shifts corresponding to a Doppler shift of $1/T_{SEQ}$, where T_{SEQ} is the length of the PRACH sequence (applies to restricted set only)
`NumCyclicShifts`	Number of cyclic shifts corresponding to a single PRACH preamble sequence

Note Logical root sequence index `prach.SequenceIndex` determines the returned physical root Zadoff-Chu sequence index `RootSequence`, based on TS 38.211 Table 6.3.3.1-3 and Table 6.3.3.1-4. However, if the preamble index within the cell, specified by `prach.PreambleIndex`, results in insufficient amount of cyclic shifts available at index `prach.SequenceIndex`, the function `nrPRACH` obtains the physical root sequence index by taking consecutive logical root sequence indices, following the process described in TS 38.211 Section 6.3.3.1. In this case, the value of `RootSequence` differs from the expected index, specified by `prach.SequenceIndex`. For an example, see “Analyze PRACH Root Sequence Indices” on page 1-221.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACHGrid` | `nrPRACHIndices`

Objects
`nrPRACHConfig`

Introduced in R2020a

nrPRACHGrid

Generate PRACH resource grid

Syntax

```
grid = nrPRACHGrid(carrier,prach)
grid = nrPRACHGrid(carrier,prach,p)
grid = nrPRACHGrid( ___ ,'OutputDataType',datatype)
```

Description

`grid = nrPRACHGrid(carrier,prach)` returns the physical random access channel (PRACH) resource grid for one antenna. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `prach` specifies PRACH configuration parameters.

`grid = nrPRACHGrid(carrier,prach,p)` returns the PRACH resource grid for `p` antennas.

`grid = nrPRACHGrid( ___ ,'OutputDataType',datatype)` specifies the resource grid symbol data type in addition to the input arguments in any of the previous syntaxes.

Examples

Generate and Map PRACH Symbols to Grid

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach = nrPRACHConfig;
```

Generate PRACH symbols and indices using the specified carrier and PRACH configuration parameters.

```
prachSym = nrPRACH(carrier,prach);
prachInd = nrPRACHIndices(carrier,prach);
```

Generate a PRACH resource grid of all zeros.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid by using the indices.

```
prachGrid(prachInd) = prachSym;
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`prach` — PRACH configuration parameters
`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object.

`p` — Number of antennas
positive integer

Number of antennas, specified as a positive integer.

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`grid` — PRACH resource grid
K-by-*L* or *K*-by-*L*-by-`p` complex array of all zeros

PRACH resource grid, returned as an *N*-by-*S* or *N*-by-*S*-by-`p` complex array of all zeros.

- *K* is equal to (`carrier.SubcarrierSpacing` / `prach.SubcarrierSpacing`) × `carrier.NSizeGrid` × 12
- *L* is the number of OFDM symbols and depends on the PRACH format.
  - For long formats, *L* = `prach.PRACHDuration`.
  - For short format C0, *L* = 7.
  - For all other short formats, *L* = 14.

    For more information on PRACH preamble formats, see TS 38.211 Tables 6.3.3.1-1 and 6.3.3.1-2 [1].

Data Types: `double` | `single`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include

`{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACH` | `nrPRACHIndices`

Objects
`nrPRACHConfig`

Introduced in R2020a

nrPRACHIndices

Generate PRACH resource element indices

Syntax

```
[ind,info] = nrPRACHIndices(carrier,prach)
[ind,info] = nrPRACHIndices(carrier,prach,Name,Value)
```

Description

`[ind,info] = nrPRACHIndices(carrier,prach)` returns resource element indices `ind` for the physical random access channel (PRACH), as defined in TS 38.211 Section 5.3.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `prach` specifies PRACH configuration parameters. The function also returns the structure `info`, which contains carrier-dependent information about the PRACH.

`[ind,info] = nrPRACHIndices(carrier,prach,Name,Value)` specifies output formatting options using one or more name-value pair arguments.

Examples

Generate and Map PRACH Symbols to Grid

Configure the PRACH and the carrier with default properties.

```
carrier = nrCarrierConfig;
prach = nrPRACHConfig;
```

Generate PRACH symbols and indices using the specified carrier and PRACH configuration parameters.

```
prachSym = nrPRACH(carrier,prach);
prachInd = nrPRACHIndices(carrier,prach);
```

Generate a PRACH resource grid of all zeros.

```
prachGrid = nrPRACHGrid(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid by using the indices.

```
prachGrid(prachInd) = prachSym;
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`prach` — PRACH configuration parameters
`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the indexing form and indexing base of the output.

`IndexStyle` — Resource element indexing form
`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

Output Arguments

`ind` — PRACH resource element indices
M-by-1 vector (default) | *M*-by-3 matrix

PRACH resource element indices, returned as one of these values.

- *M*-by-1 vector — When '`IndexStyle`' is set to `'index'`.
- *M*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

M depends on the length of the Zadoff-Chu preamble sequence and is equal to `prach.LRA`.

Depending on '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

`info` — Carrier-dependent PRACH information
structure

Carrier-dependent PRACH information, returned as a structure containing one field.

Field	Description
`PRBSet`	Physical resource block (PRB) indices occupied by the PRACH preamble for the PUSCH (0-based)

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPRACH` | `nrPRACHGrid`

Objects
`nrPRACHConfig`

Introduced in R2020a

nrPRACHOFDMInfo

Get PRACH OFDM information

Syntax

```
info = nrPRACHOFDMInfo(carrier,prach)
info = nrPRACHOFDMInfo(carrier,prach,'Windowing',samples)
```

Description

`info = nrPRACHOFDMInfo(carrier,prach)` provides dimensional information relevant to physical random access channel (PRACH) orthogonal frequency-division multiplexing (OFDM) modulation for carrier configuration parameters `carrier` and PRACH configuration parameters `prach`.

`info = nrPRACHOFDMInfo(carrier,prach,'Windowing',samples)` specifies the number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols in addition to the input arguments from the previous syntax.

Examples

Get PRACH OFDM Information

Specify carrier configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure PRACH for format A1.

```
prach = nrPRACHConfig('ConfigurationIndex',106,'SubcarrierSpacing',15);
```

Generate and display the PRACH OFDM information.

```
info = nrPRACHOFDMInfo(carrier,prach)
```

```
info = struct with fields:
                  Nfft: 1024
            SampleRate: 15360000
    CyclicPrefixLengths: [152 0 144 0 144 0 152 0 144 0 144 0 0 0]
           GuardLengths: [0 0 0 0 0 0 0 0 0 0 0 0 0 144]
          SymbolLengths: [1x14 double]
           OffsetLength: 0
              Windowing: 72
```

Get PRACH OFDM Information for Custom Windowing

Set carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
carrier = nrCarrierConfig('SubcarrierSpacing',60);
```

Configure and generate PRACH symbols.

```
prach = nrPRACHConfig;
```

Generate and display the PRACH OFDM information, specifying the number of samples over which the OFDM modulator applies windowing and overlapping of OFDM symbols.

```
samples = 95;
info = nrPRACHOFDMInfo(carrier,prach,'Windowing',samples)
```

```
info = struct with fields:
                  Nfft: 49152
            SampleRate: 61440000
    CyclicPrefixLengths: 6336
           GuardLengths: 5952
          SymbolLengths: 61440
           OffsetLength: 0
              Windowing: 95
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid

52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz

`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`prach` — PRACH configuration parameters
`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object. The function uses only these properties of this input.

`FrequencyRange` — Frequency range
`'FR1'` (default) | `'FR2'`

Frequency range, specified as `'FR1'` or `'FR2'`.

Use this property together with the `DuplexMode` property to specify these PRACH configuration tables from TS 38.211.

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`DuplexMode` — Duplex mode for uplink transmission
`'FDD'` (default) | `'TDD'` | `'SUL'`

Duplex mode for uplink transmission, specified as one of these values:

- `'FDD'` — Use this value to specify frequency division duplex (FDD) mode for paired spectrum.
- `'TDD'` — Use this value to specify time division duplex (TDD) mode for unpaired spectrum.
- `'SUL'` — Use this value to specify supplementary uplink.

Use this property together with the `FrequencyRange` property to specify these PRACH configuration tables from TS 38.211:

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`ConfigurationIndex` — Time resource of PRACH preamble
`27` (default) | integer from 0 to 255

Time resource of PRACH preamble, specified as an integer from 0 to 255. This property specifies a configuration index from Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211. Properties `FrequencyRange` and `DuplexMode` determine the actual configuration table to consider.

This property is the higher layer parameter *prach-ConfigurationIndex*.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing for PRACH in kHz
`1.25` (default) | `5` | `15` | `30` | `60` | `120`

Subcarrier spacing for the PRACH in kHz, specified as `1.25`, `5`, `15`, or `30` for frequency range FR1 and `60` or `120` for frequency range FR2.

Set this property in relation to the preamble format property `Format`. To identify valid preamble format and subcarrier spacing combinations, see the `LongPreambleFormats` and `ShortPreambleFormats` fields of the `Tables` property. For more information, see Table 6.3.3.1-1 for long preambles and Table 6.3.3.1-2 for short preambles.

Data Types: `double`

`NPRACHSlot` — PRACH slot number
`0` (default) | nonnegative integer

PRACH slot number, specified as a nonnegative integer. You can set `NPRACHSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you might have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`samples` — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as a nonnegative integer or `[]`.

If this input as `[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in Annexes F.5.5 of [2] and [3]. E is equal to value of `floor`$((N_{CP} - W) \times$ `info.Nfft`$/N_{FFT,\,nominal})$, where N_{CP}, W, and $N_{FFT,\,nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

Output Arguments

`info` — OFDM information
structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive integer	Waveform sample rate
`CyclicPrefixLengths`	1-by-*N* vector of nonnegative integers, where *N* is the number of OFDM symbols in a PRACH slot	Cyclic prefix lengths of each OFDM symbol, in samples
`GuardLengths`	1-by-*N* vector of positive integers, where *N* is the number of OFDM symbols in a PRACH slot	Guard lengths of OFDM symbols, in samples

Fields	Values	Description
`SymbolLengths`	1-by-*N* vector of nonnegative integers, where *N* is the number of OFDM symbols in a PRACH slot	OFDM symbol lengths, in samples
`OffsetLength`	Nonnegative integer	Length, in samples, of the initial time offset between the start of the configured PRACH slot period to the start of the cyclic prefix
`Windowing`	Nonnegative integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

For long formats, for which the `LRA` property of the `prach` input is `839`, the first slot of a PRACH preamble can occur part of the way through the nominal PRACH slot period. In this case, the function increases the value of the `OffsetLength` field, which ensures that the OFDM waveform spans the entire active PRACH preamble. To balance these slots with the nominal PRACH slot period, some inactive PRACH slots have OFDM waveforms that are shorter than the nominal PRACH slot period. The function conveys this by returning the `CyclicPrefixLengths` and `GuardLengths` fields as `[]`, corresponding to no OFDM symbols, and setting the `OffsetLength` field equal to the number of empty subframes required.

Data Types: `struct`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPRACH` | `nrPRACHGrid` | `nrPRACHIndices` | `nrPRACHOFDMModulate`

Objects

`nrPRACHConfig`

Introduced in R2020b

nrPRACHOFDMModulate

Generate PRACH OFDM modulated waveform

Syntax

```
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid)
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid,'Windowing',samples)
```

Description

`[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid)` generates `waveform`, a physical random access channel (PRACH) time-domain waveform, by performing orthogonal frequency-division multiplexing (OFDM) modulation of PRACH carrier resource array `grid` for carrier configuration parameters `carrier` and PRACH configuration parameters `prach`. The function also returns `info`, a structure containing OFDM information.

`[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid,'Windowing',samples)` specifies the number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols in addition to the input arguments from the previous syntax.

Examples

Generate PRACH OFDM Modulated Waveform

Generate a PRACH waveform by performing OFDM modulation of a resource array that contains PRACH symbols.

Specify carrier configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure PRACH for format A1.

```
prach = nrPRACHConfig('ConfigurationIndex',106,'SubcarrierSpacing',15);
```

Generate PRACH symbols and map to a PRACH slot resource grid.

```
sym = nrPRACH(carrier,prach);
ind = nrPRACHIndices(carrier,prach);
grid = nrPRACHGrid(carrier,prach);
grid(ind) = sym;
```

Generate the PRACH OFDM waveform by modulating the grid. Display PRACH OFDM information.

```
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid);
disp(info)
```

```
                  Nfft: 1024
            SampleRate: 15360000
    CyclicPrefixLengths: [152 0 144 0 144 0 152 0 144 0 144 0 0 0]
```

```
 GuardLengths: [0 0 0 0 0 0 0 0 0 0 0 0 0 144]
SymbolLengths: [1x14 double]
 OffsetLength: 0
    Windowing: 72
```

Generate PRACH OFDM Modulated Waveform with Custom Windowing

Generate a PRACH OFDM modulated waveform by modulating a carrier containing PRACH symbols, specifying the number of samples over which the modulator applies windowing and overlapping of OFDM symbols.

Specify carrier configuration parameters, specifying a subcarrier spacing of 60 kHz.

```
carrier = nrCarrierConfig('SubcarrierSpacing',60);
```

Configure and generate PRACH symbols and map to a PRACH slot resource grid.

```
prach = nrPRACHConfig('FrequencyRange','FR2','SubcarrierSpacing',60,'DuplexMode','TDD');
sym = nrPRACH(carrier,prach);
ind = nrPRACHIndices(carrier,prach);
grid = nrPRACHGrid(carrier,prach);
grid(ind) = sym;
```

Generate the PRACH OFDM waveform by modulating the grid, specifying the number of time domain samples over which the PRACH OFDM modulator applies windowing and overlapping of OFDM symbols.

```
samples = 80;
[waveform,info] = nrPRACHOFDMModulate(carrier,prach,grid,'Windowing',samples);
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid

52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz

`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`prach` — PRACH configuration parameters
`nrPRACHConfig` object

PRACH configuration parameters, specified as an `nrPRACHConfig` object. The function uses only these properties of this input.

`FrequencyRange` — Frequency range
`'FR1'` (default) | `'FR2'`

Frequency range, specified as `'FR1'` or `'FR2'`.

Use this property together with the `DuplexMode` property to specify these PRACH configuration tables from TS 38.211.

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`DuplexMode` — Duplex mode for uplink transmission
`'FDD'` (default) | `'TDD'` | `'SUL'`

Duplex mode for uplink transmission, specified as one of these values:

- `'FDD'` — Use this value to specify frequency division duplex (FDD) mode for paired spectrum.
- `'TDD'` — Use this value to specify time division duplex (TDD) mode for unpaired spectrum.
- `'SUL'` — Use this value to specify supplementary uplink.

Use this property together with the `FrequencyRange` property to specify these PRACH configuration tables from TS 38.211:

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`ConfigurationIndex` — Time resource of PRACH preamble
27 (default) | integer from 0 to 255

Time resource of PRACH preamble, specified as an integer from 0 to 255. This property specifies a configuration index from Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211. Properties `FrequencyRange` and `DuplexMode` determine the actual configuration table to consider.

This property is the higher layer parameter *prach-ConfigurationIndex*.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing for PRACH in kHz
`1.25` (default) | `5` | `15` | `30` | `60` | `120`

Subcarrier spacing for the PRACH in kHz, specified as `1.25`, `5`, `15`, or `30` for frequency range FR1 and `60` or `120` for frequency range FR2.

Set this property in relation to the preamble format property `Format`. To identify valid preamble format and subcarrier spacing combinations, see the `LongPreambleFormats` and `ShortPreambleFormats` fields of the `Tables` property. For more information, see Table 6.3.3.1-1 for long preambles and Table 6.3.3.1-2 for short preambles.

Data Types: `double`

`NPRACHSlot` — PRACH slot number
`0` (default) | nonnegative integer

PRACH slot number, specified as a nonnegative integer. You can set `NPRACHSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you might have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`grid` — PRACH carrier resource array
complex-valued array

PRACH carrier resource array, specified as a complex-valued array of size *K*-by-*L*-by-*P*.

- *K* is the number of subcarriers, equal to the value of 12 × (`carrier.NSizeGrid`) × (`carrier.SubcarrierSpacing`)/(`prach.SubcarrierSpacing`).
- *L* is the number of OFDM symbols in the grid.
  - For long preamble formats, *L* is equal to the value of `prach.PRACHDuration`.
  - For short preamble format C0, *L* is `7`.
  - For all other short preamble formats, *L* is `14`.

  For more information on PRACH preamble formats, see tables 6.3.3.1-1 and 6.3.3.1-2 of [1].
- *P* is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`samples` — Number of time-domain samples for OFDM symbol windowing and overlapping
nonnegative integer (default depends other input values) | `[]`

Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols, specified as a nonnegative integer or `[]`.

If you do not specify this input, or if you specify `'Windowing',[]`, the function sets this input to the maximum value E that does not impact error vector magnitude (EVM) tests, as specified in Annexes F.5.5 of [2] and [3]. E is equal to value of `floor`$((N_{CP} - W) \times$ `info.Nfft`$/N_{FFT,\,nominal})$, where N_{CP}, W, and $N_{FFT,\,nominal}$ are the values in the table columns labeled "Cyclic prefix length", "EVM window length", and "FFT size", respectively.

Data Types: `double`

Output Arguments

`waveform` — PRACH OFDM modulated waveform

complex-valued matrix

PRACH OFDM modulated waveform, returned as a complex-valued matrix of size T-by-P.

- T is the number of time domain samples in the waveform for the current slot, equal to the value of `info.OffsetLengths + sum(info.SymbolLengths)`.

  The `NPRACHSlot` property of the `prach` input determines the current slot.
- P is the number of transmit antennas.

Data Types: `double`
Complex Number Support: Yes

`info` — OFDM information

structure

OFDM information, returned as a structure containing these fields.

Fields	Values	Description
`Nfft`	Positive integer	Number of FFT points
`SampleRate`	Positive integer	Waveform sample rate
`CyclicPrefixLengths`	1-by-N vector of nonnegative integers, where N is the number of OFDM symbols in a PRACH slot	Cyclic prefix lengths of each OFDM symbol, in samples
`GuardLengths`	1-by-N vector of positive integers, where N is the number of OFDM symbols in a PRACH slot	Guard lengths of OFDM symbols, in samples
`SymbolLengths`	1-by-N vector of nonnegative integers, where N is the number of OFDM symbols in a PRACH slot	OFDM symbol lengths, in samples
`OffsetLength`	Nonnegative integer	Length, in samples, of the initial time offset between the start of the configured PRACH slot period to the start of the cyclic prefix

Fields	Values	Description
`Windowing`	Nonnegative integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols

For long formats, for which the `LRA` property of the `prach` input is `839`, the first slot of a PRACH preamble can occur part of the way through the nominal PRACH slot period. In this case, the function increases the value of the `OffsetLength` field, which ensures that the OFDM waveform spans the entire active PRACH preamble. To balance these slots with the nominal PRACH slot period, some inactive PRACH slots have OFDM waveforms that are shorter than the nominal PRACH slot period. The function conveys this by returning the `CyclicPrefixLengths` and `GuardLengths` fields as `[]`, corresponding to no OFDM symbols, and setting the `OffsetLength` field equal to the number of empty subframes required.

Data Types: `struct`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.101-2. “NR; User Equipment (UE) radio transmission and reception; Part 2: Range 2 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPRACH` | `nrPRACHGrid` | `nrPRACHIndices` | `nrPRACHOFDMInfo`

Objects

`nrPRACHConfig`

Introduced in R2020b

nrPRBS

Generate PRBS

Syntax

```
[seq,cinit] = nrPRBS(cinit,n)
[seq,cinit] = nrPRBS(cinit,n,Name,Value)
```

Description

`[seq,cinit] = nrPRBS(cinit,n)` returns the elements specified by `n` of the pseudorandom binary sequence (PRBS) generator, when initialized with `cinit`. The function implements the generator specified in TS 38.211 Section 5.2.1 [1] on page 1-246. For uniformity with the channel-specific PRBS functions, the function also returns the initialization value `cinit`.

`[seq,cinit] = nrPRBS(cinit,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified options take their default values.

Examples

Generate Pseudorandom Scrambling Sequence

Generate a 1000-bit binary scrambling sequence. Initialize the PRBS generator with the specified value.

```
cinit = 9;
prbs = nrPRBS(cinit,1000);
```

Input Arguments

`cinit` — Initialization value for PRBS generator

integer from 0 to $2^{31} - 1$

Initialization value for the PRBS generator, specified as an integer from 0 to $2^{31} - 1$.

Data Types: `double`

`n` — Elements in returned sequence

nonnegative integer | `[p m]` row vector

Elements in returned sequence, specified as one of these values:

- Nonnegative integer — `seq` contains the first `n` elements of the PRBS generator.
- `[p m]` row vector — `seq` contains `m` contiguous elements of the PRBS generator, starting at position `p` (zero-based).

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MappingType','signed'` specify non-default sequence formatting properties.

`MappingType` — Output sequence formatting
`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence
`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — Pseudorandom scrambling sequence
logical column vector | numeric column vector

Pseudorandom scrambling sequence, returned as a logical or numeric column vector. The output `seq` contains the elements of the PRBS generator specified by `n`. If you set `'MappingType'` to `'signed'`, the data type of `seq` is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCHPRBS` | `nrPDCCHPRBS` | `nrPDSCHPRBS`

Introduced in R2018b

nrPSS

Generate PSS symbols

Syntax

```
sym = nrPSS(ncellid)
sym = nrPSS(ncellid,'OutputDataType',datatype)
```

Description

`sym = nrPSS(ncellid)` returns the primary synchronization signal (PSS) symbols for the physical layer cell identity number `ncellid`. The function implements TS 38.211 Section 7.4.2.2 [1].

`sym = nrPSS(ncellid,'OutputDataType',datatype)` specifies the data type of the PSS symbol.

Examples

Generate PSS Symbols

Generate the sequence of 127 PSS binary phase shift keying (BPSK) modulation symbols for a given cell identity. The PSS is transmitted in the first symbol of a Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block.

```
ncellid = 17;
pss = nrPSS(ncellid)
```

```
pss = 127×1

    -1
    -1
    -1
    -1
    -1
    -1
     1
     1
     1
    -1
      ⋮
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PSS symbols
column vector of real numbers

PSS symbols, returned as a column vector of real numbers.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCH` | `nrPBCHDMRS` | `nrPSSIndices` | `nrSSS`

Introduced in R2018b

nrPSSIndices

Generate PSS resource element indices

Syntax

```
ind = nrPSSIndices
ind = nrPSSIndices(Name,Value)
```

Description

`ind = nrPSSIndices` returns the resource element indices for the primary synchronization signal (PSS), as defined in TS 38.211 Section 7.4.3.1 [1]. The returned indices are one-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the PSS modulation symbols are mapped.

`ind = nrPSSIndices(Name,Value)` specifies index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get PSS Resource Element Indices

Generate the 127 resource element indices associated with the PSS within a single SS/PBCH block.

```
ind = nrPSSIndices
```

```
ind = 127x1 uint32 column vector

   57
   58
   59
   60
   61
   62
   63
   64
   65
   66
      ⋮
```

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting options.

`IndexStyle` — Resource element indexing form
`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

Output Arguments

`ind` — PSS resource element indices
column vector | *M*-by-3 matrix

PSS resource element indices, returned as one of these values:

- Column vector — When `'IndexStyle'` is `'index'`.
- *M*-by-3 matrix — When `'IndexStyle'` is `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in an SS/PBCH block, and the number of antennas, respectively.

Depending on `'IndexBase'`, the indices are either one-based or zero-based.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPSS` | `nrSSSIndices`

Introduced in R2018b

nrPUCCH0

Generate PUCCH format 0 modulation symbols

Syntax

```
sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,
freqHopping)
sym = nrPUCCH0( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS, freqHopping)` returns the physical uplink control channel (PUCCH) format 0 modulation symbols, as defined in TS 38.211 Section 6.3.2.3 [1], based on these input arguments:

- Hybrid automatic repeat-request acknowledgment (HARQ-ACK) `ack`
- Scheduling request (SR) `sr`
- PUCCH symbol allocation `symAllocation`
- Cyclic prefix `cp`
- Radio frame slot number `nslot`
- Scrambling identity `nid`
- Group hopping configuration `groupHopping`
- Initial cyclic shift `initialCS`
- Intra-slot frequency hopping configuration `freqHopping`

`sym = nrPUCCH0( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 0 Modulation Symbols for Positive SR Transmission

Specify a transmission without HARQ-ACK and a positive SR.

```
ack = [];
sr = 1;
```

Specify the first symbol index in the PUCCH transmission slot as 11, the number of allocated PUCCH symbols as 2, and the slot number as 63.

```
symAllocation = [11 2];
nslot = 63;
```

Set the scrambling identity to 512 and the initial cyclic shift to 5.

```
nid = 512;
initialCS = 5;
```

Generate the symbols with normal cyclic prefix, intra-slot frequency hopping disabled, and group hopping enabled.

```
cp = 'normal';
freqHopping = 'disabled';
groupHopping = 'enable';
sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,freqHopping)
```

```
sym = 24×1 complex

   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Generate PUCCH Format 0 Modulation Symbols for HARQ-ACK Transmission

Specify two-bit HARQ-ACK transmission and a negative SR.

```
ack = [1;1];
sr = 0;
```

Specify the first symbol index in the PUCCH transmission slot as 10, the number of allocated PUCCH symbols as 2, and the slot number as 3.

```
symAllocation = [10 2];
nslot = 3;
```

Set the scrambling identity to 12 and the initial cyclic shift to 5.

```
nid = 12;
initialCS = 5;
```

Generate the symbols with extended cyclic prefix, intra-slot frequency hopping disabled, and group hopping enabled.

```
nid = 12;
initialCS = 5;
cp = 'extended';
freqHopping = 'disabled';
groupHopping = 'enable';
sym = nrPUCCH0(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,freqHopping)
```

```
sym = 24×1 complex

  -0.7071 - 0.7071i
  -0.9659 - 0.2588i
  -0.9659 + 0.2588i
```

```
-0.7071 - 0.7071i
 0.2588 - 0.9659i
-0.2588 - 0.9659i
-0.7071 + 0.7071i
 0.9659 + 0.2588i
 0.2588 + 0.9659i
-0.7071 - 0.7071i
    ⋮
```

Input Arguments

`ack` — HARQ-ACK bits

empty vector | binary column vector

HARQ-ACK bits, specified as an empty vector or a binary column vector with one or two rows. An empty vector indicates PUCCH transmission without HARQ-ACK. If specifying a binary column vector, the number of rows corresponds to the number of codewords. Vector element `1` denotes positive acknowledgement (ACK), and vector element `0` denotes negative acknowledgement (NACK).

Data Types: `double`

`sr` — SR bits

empty vector | `1` | `0`

SR bits, specified as an empty vector, `1`, or `0`. An empty vector indicates PUCCH transmission without SR. `1` denotes positive SR. `0` denotes negative SR. For negative SR without HARQ-ACK, the output `sym` is empty.

Data Types: `double`

`symAllocation` — PUCCH symbol allocation

two-element numeric vector

PUCCH symbol allocation, specified as a two-element numeric vector of the form [*S L*], where *S* and *L* are nonnegative integers.

- *S* is the first OFDM symbol index in the PUCCH transmission slot.
- *L* is the number of OFDM symbols allocated for PUCCH transmission. For PUCCH format 0, *L* is either 1 or 2.

Note *S* and *L* must satisfy these conditions.

- For extended control prefix, $S + L \leq 12$.
- For normal cyclic prefix, $S + L \leq 14$.

Data Types: `double`

`cp` — Cyclic prefix length

`'normal'` | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length only applies for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`nslot` — Radio frame slot number
integer from 0 to 159

Radio frame slot number, specified as an integer from 0 to 159. For normal cyclic prefix of different numerologies, specify an integer from 0 to 159. For extended cyclic prefix, specify an integer from 0 to 39. For more details, see TS 38.211 Section 4.3.2.

Data Types: `double`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *hoppingId*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information on these values, see TS 38.211 Section 6.3.2.2.1.

Data Types: `double`

`groupHopping` — Group hopping configuration
`'neither'` | `'enable'` | `'disable'`

Group hopping configuration, specified as `'neither'`, `'enable'`, or `'disable'`. The `groupHopping` argument is higher layer parameter *pucch-GroupHopping*.

Note When `groupHopping` is set to `'disable'`, the function enables sequence hopping. In this case, the selected sequence number might not be appropriate for short base sequences.

Data Types: `char` | `string`

`initialCS` — Initial cyclic shift
integer from 0 to 11

Initial cyclic shift, *m_0*, specified as an integer from 0 to 11. `initialCS` is higher layer parameter *initialCyclicShift*.

For more information, see TS 38.213 Section 9.2.1 [2].

Data Types: `double`

`freqHopping` — Intra-slot frequency hopping configuration
`'enabled'` | `'disabled'`

Intra-slot frequency hopping configuration, specified as `'enabled'` or `'disabled'`. The `freqHopping` argument is higher layer parameter *intraSlotFrequencyHopping*.

Data Types: `char` | `string`

datatype — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

sym — PUCCH format 0 modulation symbols
empty vector | complex column vector

PUCCH format 0 modulation symbols, returned as an empty vector or a complex column vector. `sym` is of length $12 \times L$, where L is the PUCCH symbol allocation length, specified by `symAllocation`. For negative SR without HARQ-ACK, `sym` is always empty.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrLowPAPRS` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4` | `nrPUCCHHoppingInfo`

Introduced in R2019a

nrPUCCH1

Generate PUCCH format 1 modulation symbols

Syntax

```
sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS,
freqHopping,occi)
sym = nrPUCCH1( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot,nid,groupHopping,initialCS, freqHopping,occi)` returns the physical uplink control channel (PUCCH) format 1 modulation symbols, as defined in TS 38.211 Section 6.3.2.4 [1], based on these input arguments:

- Hybrid automatic repeat-request acknowledgment (HARQ-ACK) `ack`
- Scheduling request (SR) `sr`
- PUCCH symbol allocation `symAllocation`
- Cyclic prefix `cp`
- Radio frame slot number `nslot`
- Scrambling identity `nid`
- Group hopping configuration `groupHopping`
- Initial cyclic shift `initialCS`
- Intra-slot frequency hopping configuration `freqHopping`
- Orthogonal cover code index `occi`

`sym = nrPUCCH1( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 1 Modulation Symbols for Two-Bit HARQ-ACK with Positive SR Transmission

Specify a transmission with two-bit HARQ-ACK and positive SR.

```
ack = [0;1];
sr = 1;
```

Specify the first symbol index in the PUCCH transmission slot as 0, the number of allocated PUCCH symbols as 14, and the slot number as 3.

```
symAllocation = [0 14];
nslot = 3;
```

Set the scrambling identity to 512 and the initial cyclic shift to 5.

```
nid = 512;
initialCS = 5;
```

Generate the symbols with normal cyclic prefix, intra-slot frequency hopping and group hopping enabled, and orthogonal cover code index 2.

```
cp = 'normal';
freqHopping = 'enabled';
groupHopping = 'enable';
occi = 2;
sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot, ...
    nid,groupHopping,initialCS,freqHopping,occi)
```

```
sym = 84×1 complex

  -1.0000 - 0.0000i
  -0.5000 + 0.8660i
  -0.8660 + 0.5000i
  -0.0000 - 1.0000i
  -0.8660 - 0.5000i
   0.8660 - 0.5000i
  -1.0000 + 0.0000i
   0.8660 + 0.5000i
  -0.8660 + 0.5000i
  -1.0000 + 0.0000i
      ⋮
```

Generate PUCCH Format 1 Modulation Symbols for One-Bit HARQ-ACK Transmission

Specify a transmission with one-bit HARQ-ACK and negative SR.

```
ack = 1;
sr = 0;
```

Specify the first symbol index in the PUCCH transmission slot as 3, the number of allocated PUCCH symbols as 9, and the slot number as 7.

```
symAllocation = [3 9];
nslot = 7;
```

Set the scrambling identity to 512 and the initial cyclic shift to 9.

```
nid = 512;
initialCS = 9;
```

Generate the symbols with extended cyclic prefix, intra-slot frequency hopping and group hopping enabled, and orthogonal cover code index 1.

```
cp = 'extended';
freqHopping = 'enabled';
groupHopping = 'enable';
occi = 1;
sym = nrPUCCH1(ack,sr,symAllocation,cp,nslot, ...
    nid,groupHopping,initialCS,freqHopping,occi)
```

```
sym = 48×1 complex

  -0.0000 + 1.0000i
  -0.8660 + 0.5000i
  -0.5000 + 0.8660i
   1.0000 - 0.0000i
   0.8660 - 0.5000i
   0.8660 + 0.5000i
  -0.0000 - 1.0000i
  -0.8660 + 0.5000i
   0.8660 + 0.5000i
   0.0000 + 1.0000i
      ⋮
```

Input Arguments

`ack` — HARQ-ACK bits

empty vector | binary column vector

HARQ-ACK bits, specified as an empty vector or a binary column vector with one or two rows. An empty vector indicates PUCCH transmission without HARQ-ACK. If specifying a binary column vector, the number of rows corresponds to the number of codewords. Vector element `1` denotes positive acknowledgement (ACK), and vector element `0` denotes negative acknowledgement (NACK).

Data Types: `double`

`sr` — SR bits

empty vector | `1` | `0`

SR bits, specified as an empty vector, `1`, or `0`. An empty vector indicates PUCCH transmission without SR. `1` denotes positive SR. `0` denotes negative SR. For either positive or negative SR with HARQ-ACK information bits, only HARQ-ACK transmission occurs. For negative SR without HARQ-ACK, the output `sym` is empty.

Data Types: `double`

`symAllocation` — PUCCH symbol allocation

two-element numeric vector

PUCCH symbol allocation, specified as a two-element numeric vector of the form [*S L*], where *S* and *L* are nonnegative integers.

- *S* is the first OFDM symbol index in the PUCCH transmission slot.
- *L* is the number of OFDM symbols allocated for PUCCH transmission. For PUCCH format 1, *L* is an integer from 4 or 14.

Note *S* and *L* must satisfy these conditions.

- For extended control prefix, $S + L \leq 12$.
- For normal cyclic prefix, $S + L \leq 14$.

Data Types: `double`

`cp` — Cyclic prefix length
`'normal'` | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length only applies for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`nslot` — Radio frame slot number
integer from 0 to 159

Radio frame slot number, specified as an integer from 0 to 159. For normal cyclic prefix of different numerologies, specify an integer from 0 to 159. For extended cyclic prefix, specify an integer from 0 to 39. For more details, see TS 38.211 Section 4.3.2.

Data Types: `double`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *hoppingId*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information on these values, see TS 38.211 Section 6.3.2.2.1.

Data Types: `double`

`groupHopping` — Group hopping configuration
`'neither'` | `'enable'` | `'disable'`

Group hopping configuration, specified as `'neither'`, `'enable'`, or `'disable'`. The `groupHopping` argument is higher layer parameter *pucch-GroupHopping*.

Note When `groupHopping` is set to `'disable'`, the function enables sequence hopping. In this case, the selected sequence number might not be appropriate for short base sequences.

Data Types: `char` | `string`

`initialCS` — Initial cyclic shift
integer from 0 to 11

Initial cyclic shift, *m_0*, specified as an integer from 0 to 11. `initialCS` is higher layer parameter *initialCyclicShift*.

For more information, see TS 38.213 Section 9.2.1 [2].

Data Types: `double`

`freqHopping` — Intra-slot frequency hopping configuration
`'enabled'` | `'disabled'`

Intra-slot frequency hopping configuration, specified as `'enabled'` or `'disabled'`. The `freqHopping` argument is higher layer parameter *intraSlotFrequencyHopping*.

Data Types: `char` | `string`

`occi` — Orthogonal cover code index
integer from 0 to 6

Orthogonal cover code index, specified as an integer from 0 to 6. This input argument corresponds to higher layer parameter *timeDomainOCC*. The valid range depends on the number of OFDM symbols that contain control information in a hop.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH format 1 modulation symbols
complex column vector | empty vector

PUCCH format 1 modulation symbols, returned as a complex column vector or an empty vector. `sym` is of length `12×floor(`L`/2)`, where L is the PUCCH symbol allocation length, specified by `symAllocation`. For negative SR without HARQ-ACK, the output `sym` is empty.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

nrLowPAPRS | nrPUCCH0 | nrPUCCH2 | nrPUCCH3 | nrPUCCH4 | nrPUCCHHoppingInfo

Introduced in R2019a

nrPUCCH2

Generate PUCCH format 2 modulation symbols

Syntax

```
sym = nrPUCCH2(uciCW,nid,rnti)
sym = nrPUCCH2(___,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH2(uciCW,nid,rnti)` returns the physical uplink control channel (PUCCH) format 2 modulation symbols, as defined in TS 38.211 Section 6.3.2.5 [1]. `uciCW` is the encoded uplink control information (UCI) codeword, as defined in TS 38.212 Section 6.3.1 [2]. The encoding consists of scrambling using scrambling identity `nid` and QPSK modulation. `rnti` specifies the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUCCH2(___,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 2 Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 100 bits.

```
uciCW = randi([0 1],100,1);
```

Generate PUCCH format 2 modulation symbols for the specified scrambling identity and RNTI.

```
nid = 148;
rnti = 160;
sym = nrPUCCH2(uciCW,nid,rnti)
```

```
sym = 50×1 complex

  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
       ⋮
```

Generate PUCCH Format 2 Modulation Symbols of Nondefault Data Type

Create a random sequence of binary values corresponding to a UCI codeword of 100 bits.

```
uciCW = randi([0 1],100,1);
```

Generate PUCCH format 2 modulation symbols of `single` data type for the specified scrambling identity and RNTI.

```
nid = 512;
rnti = 2563;
sym = nrPUCCH2(uciCW,nid,rnti,'OutputDataType','single')
```

```
sym = 50x1 single column vector

   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Input Arguments

`uciCW` — Encoded UCI codeword

logical column vector

Encoded UCI codeword, specified as a logical column vector. For more information, see TS 38.212 Section 6.3.1 [2].

Data Types: `int8` | `double` | `logical`

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Section 6.3.2.5.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

datatype — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

sym — PUCCH format 2 modulation symbols
complex column vector

PUCCH format 2 modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also
`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH3` | `nrPUCCH4` | `nrPUCCHPRBS` | `nrSymbolModulate`

Introduced in R2019a

nrPUCCH3

Generate PUCCH format 3 modulation symbols

Syntax

```
sym = nrPUCCH3(uciCW,mod,nid,rnti,mrb)
sym = nrPUCCH3(___,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH3(uciCW,mod,nid,rnti,mrb)` returns the physical uplink control channel (PUCCH) format 3 modulation symbols, as defined in TS 38.211 Section 6.3.2.6 [1]. `uciCW` is the encoded uplink control information (UCI) codeword, as defined in TS 38.212 Section 6.3.1 [2]. The encoding consists of scrambling using scrambling identity `nid`, symbol modulation using modulation scheme `mod`, and transform precoding based on the allocated number of resource blocks `mrb`.

`rnti` specifies the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUCCH3(___,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 3 QPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 96 bits.

```
uciCW = randi([0 1],96,1);
```

Using QPSK modulation, generate PUCCH format 3 modulation symbols for the specified scrambling identity, RNTI, and allocated bandwidth of two resource blocks.

```
modulation = 'QPSK';
nid = 148;
rnti = 160;
Mrb = 2;
sym = nrPUCCH3(uciCW,modulation,nid,rnti,Mrb)
```

```
sym = 48×1 complex

  -0.5774 - 0.2887i
  -0.0288 + 0.6273i
  -1.4717 + 0.3943i
   0.3237 - 0.3237i
   0.3660 + 0.5774i
  -0.3247 + 0.0149i
  -0.2887 + 1.1547i
  -1.0216 - 0.7397i
  -0.7113 - 0.2887i
  -0.6619 - 0.9010i
```

Generate PUCCH Format 3 Pi/2-BPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 96 bits.

```
uciCW = randi([0 1],96,1);
```

Using pi/2-BPSK modulation, generate PUCCH Format 3 modulation symbols of `single` data type for the specified scrambling identity, RNTI, and allocated bandwidth of two resource blocks.

```
modulation = 'pi/2-BPSK';
nid = 512;
rnti = 2563;
mrb = 2;
sym = nrPUCCH3(uciCW,modulation,nid,rnti,mrb,'OutputDataType','single')
```

```
sym = 96x1 single column vector

   1.1547 + 0.5774i
  -0.0197 - 0.1773i
   0.2887 + 0.2887i
   0.2887 - 0.1196i
   0.7113 + 0.2887i
  -0.4279 + 0.0475i
  -0.5774 - 0.5774i
   0.0475 - 0.4279i
   0.2887 + 0.7113i
  -0.1196 + 0.2887i
```

Input Arguments

`uciCW` — Encoded UCI codeword

logical column vector

Encoded UCI codeword, specified as a logical column vector. For more information, see TS 38.212 Section 6.3.1 [2].

Data Types: `int8` | `double` | `logical`

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'`

Modulation scheme, specified as `'pi/2-BPSK'` or `'QPSK'`. The modulation scheme determines the modulation type performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1

Modulation Scheme	Number of Bits Per Symbol
'QPSK'	2

Data Types: char | string

nid — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. nid is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, nid is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Section 6.3.2.6.1.

Data Types: double

rnti — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: double

mrb — Number of resource blocks
positive integer

Number of resource blocks associated with PUCCH format 3 transmission, specified as a positive integer. Preferred mrb values are 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, and 16.

Data Types: double

datatype — Data type of output symbols
'double' (default) | 'single'

Data type of the output symbols, specified as 'double' or 'single'.

Data Types: char | string

Output Arguments

sym — PUCCH format 3 modulation symbols
complex column vector

PUCCH format 3 modulation symbols, returned as a complex column vector.

Data Types: single | double
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH4` | `nrPUCCHPRBS` | `nrSymbolModulate` | `nrTransformPrecode`

Introduced in R2019a

nrPUCCH4

Generate PUCCH format 4 modulation symbols

Syntax

```
sym = nrPUCCH4(uciCW,mod,nid,rnti,sf,occi)
sym = nrPUCCH4( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUCCH4(uciCW,mod,nid,rnti,sf,occi)` returns the physical uplink control channel (PUCCH) format 4 modulation symbols for encoded uplink control information (UCI) codeword `uciCW`. The function implements TS 38.211 Section 6.3.2.6 [1]. The encoding consists of:

- Scrambling using scrambling identity `nid`.
- Symbol modulation using modulation scheme `mod`.
- Block-wise spreading using spreading factor `sf` and orthogonal cover code index `occi`.
- Transform precoding by considering 12 subcarriers associated with the PUCCH format 4 transmission.

`rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUCCH4( ___ ,'OutputDataType',datatype)` specifies the PUCCH symbol data type in addition to the input arguments in the previous syntax.

Examples

Generate PUCCH Format 4 QPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 96 bits.

```
uciCW = randi([0 1],96,1);
```

Using QPSK modulation, generate PUCCH format 4 modulation symbols for the specified scrambling identity, RNTI, spreading factor, and orthogonal cover code index.

```
modulation = 'QPSK';
nid = 148;
rnti = 160;
sf = 2;
occi = 1;
sym = nrPUCCH4(uciCW,modulation,nid,rnti,sf,occi)
```

```
sym = 96×1 complex

   0.0000 + 0.0000i
  -0.8165 + 0.8165i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
```

```
   0.0000 + 0.0000i
  -0.8165 + 0.8165i
   0.0000 + 0.0000i
  -1.4142 + 1.4142i
   0.0000 + 0.0000i
  -0.8165 + 0.8165i
      ⋮
```

Generate PUCCH Format 4 Pi/2-BPSK Modulation Symbols

Create a random sequence of binary values corresponding to a UCI codeword of 192 bits.

```
uciCW = randi([0 1],192,1);
```

Using pi/2-BPSK modulation, generate PUCCH format 4 modulation symbols of `single` data type for the specified scrambling identity, RNTI, spreading factor, and orthogonal cover code index.

```
modulation = 'pi/2-BPSK';
nid = 285;
rnti = 897;
sf = 4;
occi = 3;
sym = nrPUCCH4(uciCW,modulation,nid,rnti,sf,occi,'OutputDataType','single')
```

```
sym = 768x1 single column vector

   0.0000 + 0.0000i
  -1.6330 - 1.6330i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
  -1.6330 - 1.6330i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.0000 + 0.0000i
   0.8165 + 0.8165i
      ⋮
```

Input Arguments

`uciCW` — Encoded UCI codeword

logical column vector

Encoded UCI codeword, specified as a logical column vector. For pi/2-BPSK modulation, the length of `uciCW` must be a multiple of 12. For QPSK modulation, the length of `uciCW` must be a multiple of 24. For more information, see TS 38.212 Section 6.3.1 [2].

Data Types: `int8` | `double` | `logical`

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'`

Modulation scheme, specified as `'pi/2-BPSK'` or `'QPSK'`. The modulation scheme determines the modulation type performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2

Data Types: `char` | `string`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Section 6.3.2.6.1.

Data Types: `double`

`rnti` — RNTI of UE
integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`sf` — Spreading factor for PUCCH format 4
2 | 4

Spreading factor for PUCCH format 4, specified as `2` or `4`.

Data Types: `double`

`occi` — Orthogonal cover code index
nonnegative integer

Orthogonal cover code index, specified as a nonnegative integer. `occi` must be less than the spreading factor `sf`.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — PUCCH format 4 modulation symbols
complex column vector

PUCCH format 4 modulation symbols, returned as a complex column vector.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCHPRBS` | `nrSymbolModulate` | `nrTransformPrecode`

Introduced in R2019a

nrPUCCHHoppingInfo

Get PUCCH hopping information

Syntax

```
info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)
```

Description

`info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)` returns PUCCH sequence and cyclic shift hopping information. The function assumes intra-slot frequency hopping is enabled. The input arguments are:

- Cyclic prefix `cp`
- Radio frame slot number `nslot`
- Scrambling identity `nid`
- Group hopping configuration `groupHopping`
- Initial cyclic shift `initialCS`
- Sequence cyclic shift `seqCS`

Examples

Get PUCCH Hopping Information

Get PUCCH hopping information for the specified input arguments.

```
cp = 'normal';
nslot = 3;
nid = 512;
groupHopping = 'enable';
initialCS = 5;
seqCS = 0;
info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)
```

```
info = struct with fields:
          U: [13 22]
          V: [0 0]
      Alpha: [1x14 double]
        FGH: [11 20]
        FSS: 2
    Hopping: 'groupHopping'
        NCS: [239 107 223 6 24 2 3 66 238 125 209 145 44 233]
```

The output field `Alpha` provides cyclic shifts corresponding to all the symbols in a slot. Since symbol indices are zero-based, to obtain the cyclic shift value corresponding to a symbol index, you must increase the index value.

```
symInd = 0;
alpha = info.Alpha(symInd+1)
```

```
alpha = 2.0944
```

Get PUCCH Hopping Parameters When Intra-Slot Frequency Hopping Is Disabled

Get PUCCH hopping information for the specified input arguments.

```
cp = 'extended';
nslot = 7;
nid = 12;
groupHopping = 'enable';
initialCS = 9;
seqCS = 0;
info = nrPUCCHHoppingInfo(cp,nslot,nid,groupHopping,initialCS,seqCS)
```

```
info = struct with fields:
          U: [20 4]
          V: [0 0]
      Alpha: [1x12 double]
        FGH: [8 22]
        FSS: 12
    Hopping: 'groupHopping'
        NCS: [149 255 173 255 146 141 25 167 198 12 63 78]
```

To obtain the base sequence group number and base sequence number when intra-slot frequency hopping is disabled, consider only the first elements of `U` and `V`.

```
u = info.U(1)
```

```
u = 20
```

```
v = info.V(1)
```

```
v = 0
```

Input Arguments

cp — Cyclic prefix length

`'normal'` | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length only applies for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`nslot` — Radio frame slot number
integer from 0 to 159

Radio frame slot number, specified as an integer from 0 to 159. For normal cyclic prefix of different numerologies, the valid range is from 0 to 159. For extended cyclic prefix, the valid range is from 0 to 39. For more details, see TS 38.211 Section 4.3.2 [1].

Data Types: `double`

`nid` — Scrambling identity
integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *hoppingId*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information on these values, see TS 38.211 Section 6.3.2.2.1.

Data Types: `double`

`groupHopping` — Group hopping configuration
`'neither'` | `'enable'` | `'disable'`

Group hopping configuration, specified as `'neither'`, `'enable'`, or `'disable'`. The `groupHopping` argument is higher layer parameter *pucch-GroupHopping*.

Data Types: `char` | `string`

`initialCS` — Initial cyclic shift
integer from 0 to 11

Initial cyclic shift, referred to as *m_0* in TS 38.211 Section 6.3.2.2.2, specified as an integer from 0 to 11. For PUCCH formats 0 and 1, `initialCS` is higher layer parameter *initialCyclicShift*. For PUCCH format 3 demodulation reference signals (DMRS), `initialCS` must be 0. For PUCCH format 4 DMRS, `initialCS` must be 0, 3, 6, or 9. For more information, see TS 38.213 Section 9.2.1 [2].

Data Types: `double`

`seqCS` — Sequence cyclic shift
integer from 0 to 11

Sequence cyclic shift, referred to as *m_cs* in TS 38.211 Section 6.3.2.2.2, specified as an integer from 0 to 11. For PUCCH formats 1, 2, 3, and 4, `seqCS` must be 0.

Data Types: `double`

Output Arguments

`info` — Sequence and cyclic shift hopping information
structure

Sequence and cyclic shift hopping information, returned as a structure that contains these fields:

Parameter Field	Values	Description
U	1-by-2 integer vector	Base sequence group numbers, returned as a 1-by-2 integer vector with element values from 0 to 29. The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
V	1-by-2 logical vector	Base sequence numbers, returned a 1-by-2 logical vector. The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
Alpha	1-by-14 integer vector, 1-by-12 integer vector	Cyclic shifts of all symbols in a slot, returned as a 1-by-14 integer vector (for normal cyclic prefix) or 1-by-12 integer vector (for extended cyclic prefix). The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
FGH	1-by-2 integer vector	Sequence-group hopping pattern, returned as 1-by-2 integer vector with values from 0 to 29. The first vector element corresponds to the first hop in a slot. The second vector element corresponds to the second hop in a slot.
FSS	nonnegative integer	Sequence-group shift offset, returned as a nonnegative integer from 0 to 29.
Hopping	`'neither'`, `'groupHopping'`, `'sequenceHopping'`	Hopping configuration, returned as `'neither'`, `'groupHopping'`, or `'sequenceHopping'`. The hopping configuration is based on the input argument `groupHopping`.
NCS	1-by-14 integer vector, 1-by-12 integer vector	Hopping identity of cyclic shifts, referred to as *n_cs* in TS 38.211 Section 6.3.2.2.2, returned as a 1-by-14 integer vector (for normal cyclic prefix) or 1-by-12 integer vector (for extended cyclic prefix). A vector element at position *i* corresponds to the hopping identity of cyclic shift at symbol position *i* in a slot.

Data Types: `struct`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUCCH0` | `nrPUCCH1` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4`

Introduced in R2019a

nrPUCCHPRBS

Generate PUCCH scrambling sequence

Syntax

```
[seq,cinit] = nrPUCCHPRBS(nid,rnti,n)
[seq,cinit] = nrPUCCHPRBS(nid,rnti,n,Name,Value)
```

Description

`[seq,cinit] = nrPUCCHPRBS(nid,rnti,n)` returns the first `n` elements of the physical uplink control channel (PUCCH) scrambling sequence. The function also returns initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on scrambling identity `nid` and radio network temporary identifier (RNTI) of the user equipment (UE) `rnti`. The function implements TS 38.211 Section 6.3.2.5.1/6.3.2.6.1 [1].

`[seq,cinit] = nrPUCCHPRBS(nid,rnti,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified name-value pairs take their default values.

Examples

Generate PUCCH Scrambling Sequence

Generate the first 300 elements of the PUCCH scrambling sequence when initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 17;
rnti = 120;
n = 300;
seq = nrPUCCHPRBS(ncellid,rnti,n)
```

```
seq = 300x1 logical array

   0
   1
   1
   0
   1
   1
   0
   1
   0
   0
      ⋮
```

Input Arguments

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is the physical layer cell identity number *NCellID*, ranging from 0 to 1007.

For more information, see TS 38.211 Sections 6.3.2.5.1 and 6.3.2.6.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`n` — Number of elements in output sequence

nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MappingType','signed'` specifies nondefault sequence formatting.

`MappingType` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence

`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

seq — PUCCH scrambling sequence

logical column vector | numeric column vector

PUCCH scrambling sequence, returned as a logical or numeric column vector. `seq` contains the first `n` elements of the PUCCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the output data type is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

cinit — Initialization value for PRBS generator

nonnegative integer

Initialization value for PRBS generator, returned as a nonnegative integer.

Data Types: `double`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPRBS` | `nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4`

Introduced in R2019a

nrPUSCH

Generate PUSCH modulation symbols

Syntax

```
sym = nrPUSCH(cw,mod,nLayers,nid,rnti)
sym = nrPUSCH( ___ ,transformPrecode,mrb)
sym = nrPUSCH( ___ ,txScheme,nPorts,tpmi)
sym = nrPUSCH(carrier,pusch,cw)
sym = nrPUSCH( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUSCH(cw,mod,nLayers,nid,rnti)` returns physical uplink shared channel (PUSCH) modulation symbols, as defined in TS 38.211 Sections 6.3.1.1 to 6.3.1.5 [1]. The process consists of scrambling with scrambling identity `nid`, performing symbol modulation with modulation scheme `mod`, and layer mapping. `cw` specifies an uplink shared channel (UL-SCH) codeword, as described in TS 38.212 Section 6.2.7 [2]. `nLayers` specifies the number of transmission layers. `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

`sym = nrPUSCH( ___ ,transformPrecode,mrb)` specifies transform precoding as a logical value in addition to the input arguments in the first syntax. When `transformPrecode` is set to `true`, the function applies the transform precoding defined in TS 38.211 Section 6.3.1.4. `mrb` specifies the allocated number of PUSCH resource blocks.

`sym = nrPUSCH( ___ ,txScheme,nPorts,tpmi)` specifies the transmission scheme in addition to the input arguments in the second syntax. When `txScheme` is set to `'codebook'`, the function performs multi-input multi-output (MIMO) precoding based on the specified number of layers `nLayers`, number of antenna ports `nPorts`, and the transmitted precoding matrix indicator (TPMI) `tpmi`.

`sym = nrPUSCH(carrier,pusch,cw)` returns PUSCH modulation symbols for the specified carrier configuration `carrier` and PUSCH configuration `pusch`. The input `cw` specifies an uplink shared channel (UL-SCH) codeword.

`sym = nrPUSCH( ___ ,'OutputDataType',datatype)` specifies the PUSCH symbol data type in addition to the input arguments in any of the previous syntaxes.

Examples

Generate PUSCH Modulation Symbols

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using 16-QAM modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, and two transmission layers. By default, the function disables transform precoding and noncodebook-based transmission.

```
modulation = '16QAM';
nlayers = 2;
ncellid = 17;
rnti = 111;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti)
```

```
sym = 1008×2 complex

  -0.9487 - 0.9487i  -0.3162 + 0.3162i
   0.3162 + 0.3162i  -0.9487 - 0.3162i
   0.3162 + 0.3162i   0.3162 - 0.3162i
   0.9487 - 0.3162i  -0.3162 + 0.9487i
  -0.3162 - 0.9487i   0.3162 - 0.9487i
  -0.3162 + 0.9487i   0.3162 - 0.3162i
   0.3162 + 0.3162i   0.9487 - 0.9487i
  -0.9487 + 0.9487i  -0.3162 + 0.3162i
   0.9487 - 0.9487i  -0.9487 - 0.3162i
  -0.9487 - 0.9487i   0.3162 + 0.9487i
      ⋮
```

Generate PUSCH Symbols Using Codebook-Based Transmission

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using 256-QAM modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, bandwidth, and one transmission layer. Enable transform precoding and codebook-based transmission based on the specified TPMI and four antennas.

```
modulation = '256QAM';
ncellid = 17;
rnti = 111;
mrb = 6;
nlayers = 1;
transformPrecode = true;
txScheme = 'codebook';
tpmi = 1;
nports = 4;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti,transformPrecode,mrb,txScheme,nports,tpmi)
```

```
sym = 1008×4 complex

   0.0000 + 0.0000i   0.2169 + 0.2350i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.2296 + 0.3713i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0797 - 0.9008i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.4767 - 0.0143i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.4124 + 0.2638i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.1433 - 0.2366i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0885 - 0.1080i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.5507 - 0.1894i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.3039 - 0.9165i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.1498 + 0.3356i   0.0000 + 0.0000i   0.0000 + 0.0000i
      ⋮
```

Generate PUSCH Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to 30 kHz of subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with specified properties. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
pusch.NSizeBWP = 41;
pusch.Modulation = '16QAM';
pusch.NID = []; % Set NID equal to the NCellID property of carrier.
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
```

Generate PUSCH indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexOrientation','carrier')
```

```
ind = 864x1 uint32 column vector

   121
   122
   123
   124
   125
   126
   127
   128
   129
   130
      ⋮


info = struct with fields:
                 G: 3456
                Gd: 864
         NREPerPRB: 144
     DMRSSymbolSet: [2 7]
     PTRSSymbolSet: [1x0 double]
```

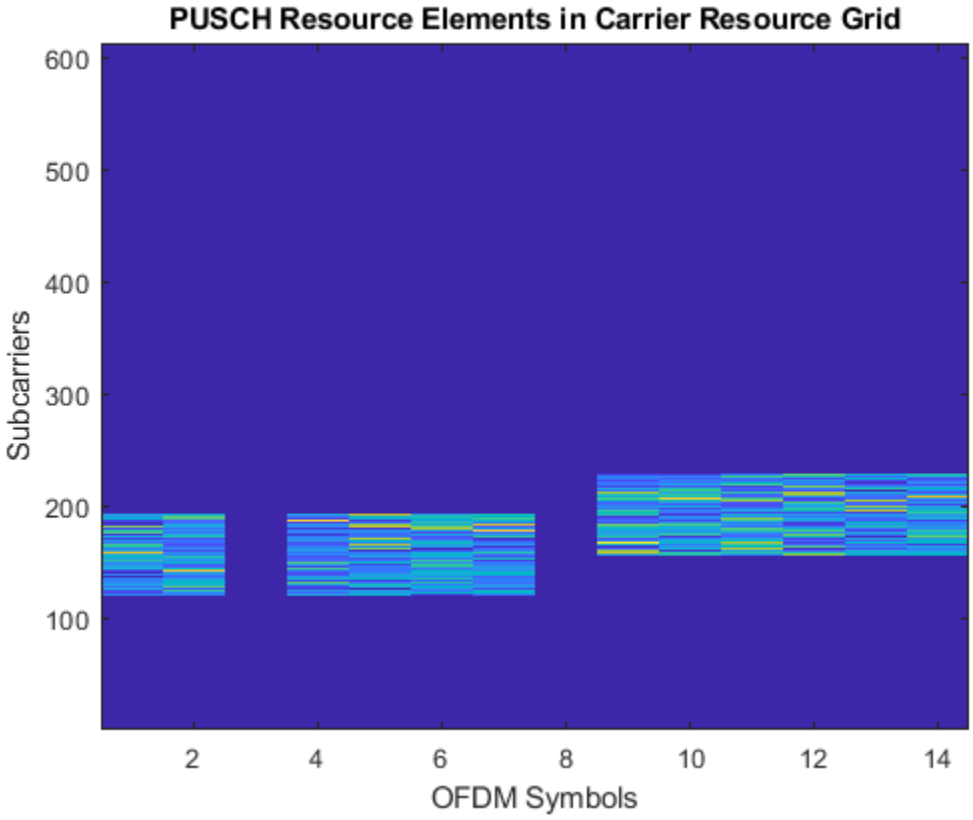
Generate PUSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPUSCH(carrier,pusch,cws,'OutputDataType','single')
```

```
sym = 864x1 single column vector

  -0.7454 + 0.2981i
   0.3406 - 0.2312i
  -0.1153 + 0.2756i
   1.1921 - 0.3658i
  -0.3968 - 0.0277i
  -0.8788 - 0.6493i
  -0.8737 + 0.8318i
  -0.5764 + 0.0269i
  -1.6638 + 0.0482i
  -1.0270 - 0.1347i
      ⋮
```

Plot the generated symbols and indices on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers]));
grid(ind) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH Resource Elements in Carrier Resource Grid');
```



Input Arguments

`cw` — UL-SCH codeword

column vector of integers from –2 to 1

UL-SCH codeword from TS 38.212 Section 6.2.7, specified as a column vector of integers from –2 to 1.

- 0 and 1 represent false and true bit values, respectively.
- –1 and –2 represent *x* and *y* placeholders in the uplink control information (UCI), respectively. For more details, see TS 38.212 Sections 5.3.3.1 and 5.3.3.2.

Data Types: `double` | `int8`

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers

integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`transformPrecode` — Transform precoding
`false` (default) | `true`

Transform precoding, specified as `false` or `true`. For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `double` | `logical`

`mrb` — Number of allocated PUSCH resource blocks
integer from 1 to 275

Number of allocated PUSCH resource blocks, specified as an integer from 1 to 275. For more information, see TS 38.214 Section 6.1.2.

Data Types: `double`

`txScheme` — Transmission scheme
`'nonCodebook'` (default) | `'codebook'`

Transmission scheme, specified as one of these values:

- `'nonCodebook'` — Use this option to disable MIMO precoding.
- `'codebook'` — Use this option for codebook-based transmission using MIMO precoding.

For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `char` | `string`

`tpmi` — Transmitted precoding matrix indicator
integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27. The valid range of `tpmi` depends on the specified number of transmission layers, `nLayers`, and number of antenna ports, `nPorts`. For more information, see TS 38.211 Tables 6.3.1.5-1 to 6.3.1.5-7.

Data Types: `double`

`nPorts` — Number of antenna ports
`1` | `2` | `4`

Number of antenna ports, specified as `1` , `2`, or `4`. For more information, see TS 38.211 Section 6.3.1.5.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only `NCellID` property of this `nrCarrierConfig` object.

Property Field	Values	Description
NCellID	1 (default), integer from 0 to 1007	Physical layer cell identity

pusch — PUSCH configuration parameters
nrPUSCHConfig object

PUSCH configuration parameters for a specific OFDM numerology, specified as an nrPUSCHConfig object. This function only uses these nrPUSCHConfig object properties.

Property Field	Values	Description
Modulation	'QPSK', '16QAM', '64QAM', or '256QAM', 'pi/2-BPSK', string scalar, or character array	Modulation scheme of codeword
NumLayers	1 (default), integer from 1 to 4	Number of transmission layers
NID	[] (default), integer from 0 to 1023	Scrambling identity, specified as an integer from 0 to 1023. Use [] value to allow this property to be equal to NCellID of carrier input.
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment
TransformPrecoding	0 (default), 1	Transform precoding flag, specified as one of these values. • 0 — The transform precoding is disabled and waveform type is CP-OFDM. • 1 — The transform precoding is enabled and waveform type is DFT-s-OFDM.
PRBSet	[0:51] (default), vector of nonnegative integers from 0 to 274	Physical resource blocks allocated for the shared channel within a BWP (0-based)
TransmissionScheme	'nonCodebook' (default), 'codebook'	PUSCH transmission scheme
NumAntennaPorts	1 (default), 2, or 4	Number of antenna ports
TPMI	0 (default), integer from 0 to 27	Transmitted precoding matrix indicator

Output Arguments

sym — PUSCH modulation symbols
complex matrix

PUSCH modulation symbols, returned as a complex matrix. If txScheme is set to 'codebook', the number of matrix columns is nPorts. If txScheme is set to 'nonCodebook', the number of matrix columns is nLayers.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPUSCHCodebook` | `nrPUSCHDecode` | `nrPUSCHScramble` | `nrULSCHInfo`

Objects

`nrCarrierConfig` | `nrPUSCHConfig`

Introduced in R2019a

nrPUSCHCodebook

Generate PUSCH precoding matrix

Syntax

```
w = nrPUSCHCodebook(nLayers,nPorts,tpmi)
w = nrPUSCHCodebook( ___ ,transformPrecode)
```

Description

`w = nrPUSCHCodebook(nLayers,nPorts,tpmi)` returns the physical uplink shared channel (PUSCH) precoding matrix for codebook-based transmission. `nLayers` is the number of layers, `nPorts` is the number of antenna ports, and `tpmi` is the transmitted precoding matrix indicator (TPMI). By default, this function disables transform precoding. The returned matrix, `w`, is the transpose of the precoding matrix defined in TS 38.211 Section 6.3.1.5 [1]. The matrix orientation of `w` allows the precoding operation to be performed by matrix multiplication on the output of the `nrLayerMap` function and `w`.

`w = nrPUSCHCodebook( ___ ,transformPrecode)` specifies transform precoding as a logical value in addition to the input arguments in the previous syntax. When `transformPrecode` is set to `true`, the function applies the transform precoding defined in TS 38.211 Section 6.3.1.4 [1].

Examples

Apply PUSCH Precoding Codebook

Modulate a random sequence of binary values of length 600 by using 64-QAM modulation. Split the modulated symbols into two layers.

```
modulation = '64QAM';
nlayers = 2;
in = randi([0 1],600,1);
data = nrSymbolModulate(in,modulation);
y = nrLayerMap(data,nlayers);
```

Generate the PUSCH precoding matrix for four antennas, two layers, and the specified TPMI.

```
nports = 4;
tpmi = 7;
w = nrPUSCHCodebook(nlayers,nports,tpmi)
```

```
w = 2×4 complex

   0.5000 + 0.0000i   0.0000 + 0.0000i   0.5000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.5000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.5000i
```

Precode the layered modulation symbols by using the codebook matrix.

```
z = y * w
```

```
z = 50×4 complex

  -0.0772 - 0.3858i   0.3858 - 0.5401i  -0.0772 - 0.3858i   0.5401 + 0.3858i
  -0.3858 + 0.0772i  -0.5401 - 0.0772i  -0.3858 + 0.0772i   0.0772 - 0.5401i
  -0.5401 - 0.2315i  -0.2315 + 0.0772i  -0.5401 - 0.2315i  -0.0772 - 0.2315i
  -0.3858 + 0.2315i  -0.2315 - 0.0772i  -0.3858 + 0.2315i   0.0772 - 0.2315i
  -0.0772 - 0.3858i   0.5401 + 0.2315i  -0.0772 - 0.3858i  -0.2315 + 0.5401i
  -0.5401 + 0.5401i  -0.0772 + 0.2315i  -0.5401 + 0.5401i  -0.2315 - 0.0772i
  -0.3858 + 0.2315i  -0.0772 + 0.5401i  -0.3858 + 0.2315i  -0.5401 - 0.0772i
  -0.3858 + 0.5401i   0.5401 + 0.3858i  -0.3858 + 0.5401i  -0.3858 + 0.5401i
  -0.2315 + 0.0772i   0.2315 - 0.5401i  -0.2315 + 0.0772i   0.5401 + 0.2315i
  -0.2315 - 0.0772i   0.3858 - 0.3858i  -0.2315 - 0.0772i   0.3858 + 0.3858i
      ⋮
```

Input Arguments

`nLayers` — Number of transmission layers

integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`nPorts` — Number of antenna ports

1 | 2 | 4

Number of antenna ports, specified as 1 , 2, or 4. For more information, see TS 38.211 Section 6.3.1.5.

Data Types: `double`

`tpmi` — Transmitted precoding matrix indicator

integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27. The valid range of `tpmi` depends on the specified number of transmission layers, `nLayers`, and number of antenna ports, `nPorts`. For more information, see TS 38.211 Tables 6.3.1.5-1 to 6.3.1.5-7.

Data Types: `double`

`transformPrecode` — Transform precoding

`false` (default) | `true`

Transform precoding, specified as `false` or `true`. For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `double` | `logical`

Output Arguments

`w` — PUSCH precoding codebook

complex matrix

PUSCH precoding codebook, returned as a complex matrix of size `nLayers`-by-`nPorts`. If `nLayers` and `nPorts` are both `1`, then `w` is `1`. Otherwise, the function returns the transpose of the matrix selected from Tables 6.3.1.5-1 to 6.3.1.5-7 in TS 38.211 [1].

Data Types: `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrLayerDemap` | `nrLayerMap`

Introduced in R2019a

nrPUSCHDecode

Decode PUSCH modulation symbols

Syntax

```
[cw,symbols] = nrPUSCHDecode(sym,mod,nid,rnti)
[cw,symbols] = nrPUSCHDecode(___,nVar)
[cw,symbols] = nrPUSCHDecode(___,transformPrecode,mrb)
[cw,symbols] = nrPUSCHDecode(___,txScheme,nLayers,tpmi)
[cw,symbols] = nrPUSCHDecode(carrier,pusch,sym,nVar)
```

Description

`[cw,symbols] = nrPUSCHDecode(sym,mod,nid,rnti)` returns soft bits `cw` and constellation symbols `symbols` resulting from the inverse operation of physical uplink shared channel (PUSCH) processing from TS 38.211 Section 6.3.1 [1]. The decoding consists of layer demapping, demodulation of symbols `sym` with modulation scheme `mod`, and descrambling with scrambling identity `nid`. The input `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE). This function performs data descrambling only. Because uplink control information (UCI) placeholder bit locations are unknown, the function cannot correctly descramble UCIs if present in the input.

`[cw,symbols] = nrPUSCHDecode(___,nVar)` specifies the noise variance scaling factor of the soft bits in the PUSCH demodulation in addition to the input arguments in the first syntax.

`[cw,symbols] = nrPUSCHDecode(___,transformPrecode,mrb)` specifies transform deprecoding as a logical value and the number of allocated PUSCH resource blocks. Specify these inputs in addition to the input arguments in the second syntax. When `transformPrecode` is set to `true`, the function applies the inverse of transform precoding defined in TS 38.211 Section 6.3.1.4. `mrb` specifies the allocated number of PUSCH resource blocks.

`[cw,symbols] = nrPUSCHDecode(___,txScheme,nLayers,tpmi)` specifies the transmission scheme in addition to the input arguments in the third syntax. When `txScheme` is set to `'codebook'`, the function performs multi-input multi-output (MIMO) deprecoding based on the specified number of transmission layers `nLayers` and transmitted precoding matrix indicator (TPMI) `tpmi`.

`[cw,symbols] = nrPUSCHDecode(carrier,pusch,sym,nVar)` returns soft bits `cw` and constellation symbols `symbols` for the specified carrier configuration `carrier` and PUSCH configuration `pusch`. The input `sym` is received PUSCH symbols to decode. The optional input `nVar` specifies the noise variance scaling factor of the soft bits.

Examples

Generate and Decode PUSCH Modulation Symbols

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using 256-QAM modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, and two transmission layers. By default, this function disables transform precoding and noncodebook-based transmission.

```
modulation = '256QAM';
nlayers = 2;
ncellid = 17;
rnti = 111;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti)
```

```
sym = 504×2 complex

  -0.9971 - 0.8437i   0.0767 + 0.2301i
   0.3835 + 0.2301i   0.9971 - 0.5369i
  -0.3835 - 1.1504i  -0.3835 + 0.9971i
   0.5369 + 0.0767i  -0.9971 + 0.8437i
   1.1504 - 0.9971i  -0.8437 - 0.6903i
  -0.6903 + 0.0767i   1.1504 - 0.3835i
   0.8437 + 0.6903i   1.1504 + 0.2301i
  -0.6903 - 0.2301i  -0.8437 + 1.1504i
   0.0767 + 0.8437i  -0.0767 + 0.6903i
   0.3835 - 0.8437i   0.3835 + 0.9971i
      ⋮
```

Decode the PUSCH modulation symbols.

```
demod = nrPUSCHDecode(sym,modulation,ncellid,rnti)
```

```
demod = 8064×1
10^10 ×

   -1.1529
   -0.8471
    0.2118
   -0.0941
   -0.0235
    0.0235
    0.0235
   -0.0235
   -0.0235
   -0.0941
      ⋮
```

Perform hard decision on the soft metric.

```
rxcw = double(demod<0)
```

```
rxcw = 8064×1

     1
     1
     0
     1
     1
     0
     0
     1
```

```
     1
     1
      ⋮
```

Compare the result with the original codeword.

```
isequal(cw,rxcw)
```

```
ans = logical
   1
```

Generate and Decode PUSCH Modulation Symbols for Codebook-Based Transmission

Specify a random sequence of binary values corresponding to a codeword of 8064 bits.

```
cw = randi([0 1],8064,1);
```

Using QPSK modulation, generate PUSCH modulation symbols for the specified physical layer cell identity number, RNTI, bandwidth, and one transmission layer. Enable transform precoding and codebook-based transmission based on the specified PUSCH bandwidth, TPMI, and four antennas.

```
modulation = 'QPSK';
ncellid = 17;
rnti = 111;
nlayers = 1;
transformPrecode = true;
txScheme = 'codebook';
mrb = 6;
tpmi = 1;
nports = 4;
sym = nrPUSCH(cw,modulation,nlayers,ncellid,rnti,transformPrecode,mrb,txScheme,nports,tpmi)
```

```
sym = 4032×4 complex

   0.0000 + 0.0000i  -0.1667 + 0.0833i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0632 - 0.2911i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.1519 - 0.0450i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.3677 + 0.3664i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.3079 - 0.5027i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.8082 - 0.1640i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0640 - 0.2388i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.3936 - 0.4160i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0851 - 0.4625i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0345 - 0.3333i   0.0000 + 0.0000i   0.0000 + 0.0000i
      ⋮
```

Decode the PUSCH modulation symbols assuming zero noise variance.

```
nVar = 0;
demod = nrPUSCHDecode(sym,modulation,ncellid,rnti,nVar,transformPrecode,mrb,txScheme,nlayers,tpmi
```

```
demod = 8064×1
10^10 ×
```

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

Perform hard decision on the soft metric.

```
rxcv = double(demod<0)
```

```
rxcv = 8064×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮
```

Compare the result with the original codeword.

```
isequal(cw,rxcv)
```

```
ans = logical
   1
```

Decode PUSCH Symbols using Configuration Parameters

Create a carrier configuration object with physical layer cell identity as 42.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
```

Create a PUSCH configuration object with these properties.

```
pusch = nrPUSCHConfig;
pusch.Modulation = '256QAM';
pusch.NumLayers = 2;
pusch.RNTI = 111;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'nonCodebook';
```

```
pusch.NID = []; % Use empty to be equal to NCellID of carrier
pusch.NSizeBWP = 25;
pusch.NStartBWP = 10;
pusch.PRBSet = 0:pusch.NSizeBWP-1; % Occupy entire bandwidth part
```

Generate PUSCH symbols for a single codeword of 8064 bits with the specified carrier configuration and uplink shared channel configuration.

```
cw = randi([0 1],8064,1);
sym = nrPUSCH(carrier,pusch,cw)
```

```
sym = 504×2 complex

  -0.3835 - 0.8437i  -1.1504 - 0.3835i
   0.6903 + 0.6903i   0.0767 + 0.3835i
  -1.1504 - 0.2301i  -1.1504 + 0.2301i
   0.9971 - 0.6903i  -0.2301 - 0.8437i
  -0.0767 + 0.2301i   0.2301 + 0.8437i
  -0.6903 + 1.1504i   0.8437 + 0.3835i
  -0.0767 + 0.8437i  -0.3835 + 0.5369i
   0.3835 + 0.5369i  -0.8437 - 1.1504i
   0.3835 + 0.2301i  -1.1504 - 0.8437i
   0.2301 - 0.0767i  -0.3835 + 1.1504i
      ⋮
```

Add an additive white Gaussian noise (AWGN) to the PUSCH symbols. Then demodulate to produce soft bit estimates.

```
SNR = 30; % SNR in dB
rxsym = awgn(sym,SNR);
demod = nrPUSCHDecode(carrier,pusch,rxsym)
```

```
demod = 8064×1
10^10 ×

   -0.2106
   -0.8118
    0.0949
   -0.0824
   -0.0231
    0.0294
    0.0239
   -0.0176
   -1.4404
   -0.1963
      ⋮
```

Perform hard decision on the soft metric.

```
rxcw = double(demod<0);
```

Compare the result with the original codeword.

```
isequal(cw,rxcw)
```

```
ans = logical
   1
```

Input Arguments

`sym` — Received PUSCH modulation symbols

complex matrix

Received PUSCH modulation symbols, specified as a complex matrix.

Data Types: `single` | `double`
Complex Number Support: Yes

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`nVar` — Noise variance

`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power.

Note The default value assumes the decoder and coder are connected back-to-back, where the noise variance is zero. To avoid `-Inf` or `+Inf` values in the output, the function uses `1e-10` as the default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

`transformPrecode` — Transform deprecoding
`false` (default) | `true`

Transform deprecoding, specified as `false` or `true`. For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `double` | `logical`

`mrb` — Number of allocated PUSCH resource blocks
integer from 1 to 275

Number of allocated PUSCH resource blocks, specified as an integer from 1 to 275. For more information, see TS 38.214 Section 6.1.2.

Data Types: `double`

`txScheme` — Transmission scheme
`'nonCodebook'` (default) | `'codebook'`

Transmission scheme, specified as one of these values:

- `'nonCodebook'` — Use this option to disable MIMO deprecoding.
- `'codebook'` — Use this option for codebook-based transmission using MIMO deprecoding.

For more information, see TS 38.211 Section 6.3.1.4.

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`tpmi` — Transmitted precoding matrix indicator
integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27. The valid range of `tpmi` depends on the specified number of transmission layers `nLayers` and the number of ports. For more information, see TS 38.211 Tables 6.3.1.5-1 to 6.3.1.5-7.

Data Types: `double`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only `NCellID` property of this `nrCarrierConfig` object.

Property Field	Values	Description
`NCellID`	1 (default), integer from 0 to 1007	Physical layer cell identity

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters for a specific OFDM numerology, specified as an `nrPUSCHConfig` object. This function only uses these `nrPUSCHConfig` object properties.

Property Field	Values	Description
`Modulation`	'QPSK', '16QAM', '64QAM', or '256QAM', 'pi/2-BPSK', string scalar, or character array	Modulation scheme of codeword
`NumLayers`	1 (default), integer from 1 to 4	Number of transmission layers
`NID`	`[]` (default), integer from 0 to 1023	Scrambling identity, specified as an integer from 0 to 1023. Use `[]` value to allow this property to be equal to `NCellID` of `carrier` input.
`RNTI`	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment
`TransformPrecoding`	0 (default), 1	Transform precoding flag, specified as one of these values. • `0` — The transform precoding is disabled and waveform type is CP-OFDM. • `1` — The transform precoding is enabled and waveform type is DFT-s-OFDM.
`PRBSet`	[0:51] (default), vector of nonnegative integers from 0 to 274	Physical resource blocks allocated for the shared channel within a BWP (0-based)
`TransmissionScheme`	`'nonCodebook'` (default), `'codebook'`	PUSCH transmission scheme
`TPMI`	0 (default), integer from 0 to 27	Transmitted precoding matrix indicator

Output Arguments

`cw` — Approximate LLR soft bits
real column vector

Approximate log-likelihood ratio (LLR) soft bits, returned as a real column vector. `cw` inherits the data type of `sym`. Sign represents hard bits.

Data Types: `double` | `single`

`symbols` — Constellation symbols
column vector of complex numbers

Constellation symbols for `cw`, returned as a column vector of complex numbers. `symbols` inherits the data type of `sym`.

Data Types: `double` | `single`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUSCH` | `nrPUSCHCodebook` | `nrPUSCHDescramble` | `nrULSCHInfo`

Objects
`nrCarrierConfig` | `nrPUSCHConfig`

Introduced in R2019a

nrPUSCHDescramble

Perform PUSCH descrambling

Syntax

```
cw = nrPUSCHDescramble(in,nid,rnti)
cw = nrPUSCHDescramble(in,nid,rnti,xInd,yInd)
```

Description

`cw = nrPUSCHDescramble(in,nid,rnti)` returns a column vector of soft bits resulting from the inverse operation of physical uplink shared channel (PUSCH) scrambling from TS 38.211 Section 6.3.1.1 [1]. `in` is a vector of scrambled soft bits, `nid` is the scrambling identity, and `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE). When you use this syntax, the function descrambles only the data bits because the placeholder bit locations for any uplink control information (UCI), if present, are unknown in this case.

`cw = nrPUSCHDescramble(in,nid,rnti,xInd,yInd)` returns a column vector of soft bits by taking into account the UCI placeholder X bit locations, `xInd`, and the UCI placeholder Y bit locations, `yInd`. The inputs `xInd` and `yInd` are 1-based column vectors within the codeword and indicate the respective placeholder locations.

Examples

Perform PUSCH Descrambling

Create a random sequence of binary values corresponding to a codeword containing 3000 bits. Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
cw = randi([0 1],3000,1);
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti);
```

Modulate the scrambled data using 16-QAM modulation. Demodulate the result.

```
modulation = '16QAM';
sym = nrSymbolModulate(scrambled,modulation);
demod = nrSymbolDemodulate(sym,modulation);
```

Perform PUSCH descrambling of the demodulated symbols.

```
descrambled = nrPUSCHDescramble(demod,ncellid,rnti)
```

```
descrambled = 3000×1
10^10 ×

   -1.6000
   -1.6000
```

```
    0.4000
   -0.4000
   -1.6000
    0.4000
    0.4000
   -0.4000
   -0.4000
   -0.4000
      ⋮
```

Perform hard decision on the soft metric.

```
rxcw = double(descrambled<0)
```

```
rxcw = 3000×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
      ⋮
```

Compare the result with the original codeword.

```
isequal(cw,rxcw)
```

```
ans = logical
   1
```

Perform PUSCH Descrambling with UCI Placeholder Indices

Create a codeword for 1 bit.

```
cw = [1 -2 -1 -1]';
```

Specify the scrambling identity as 100 and radio network temporary identifier as 65,350.

```
nid = 100;
rnti = 65350;
```

Perform PUSCH scrambling that is initialized with the UCI placeholder X and Y bit locations.

```
xind = find(cw == -1);
yind = find(cw == -2);
scrambled = nrPUSCHScramble(cw,nid,rnti);
```

Modulate the scrambled data using 16-QAM scheme. Demodulate the result.

```
modulation = '16QAM';
sym = nrSymbolModulate(scrambled,modulation);
demod = nrSymbolDemodulate(sym,modulation);
```

Perform PUSCH descrambling of the demodulated symbols.

```
descrambled = nrPUSCHDescramble(demod,nid,rnti,xind,yind)
```

```
descrambled = 4×1
10^10 ×

   -1.6000
   -1.6000
   -0.4000
   -0.4000
```

Compare the result with the original codeword.

```
isequal(descrambled(1)<0,cw(1))
```

```
ans = logical
   1
```

Input Arguments

`in` — Approximate LLR soft bits

real column vector

Approximate log-likelihood ratio (LLR) soft bits, specified as a real column vector. Sign represents scrambled hard bit.

Data Types: `double` | `single`

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`xInd` — X placeholder indices of UCI

column vector of positive values

X placeholder indices of the UCI, specified as a column vector of positive values. When you specify this input, the function descrambles the input codeword, `in`, at all locations except the X placeholder locations. A value of `[]` indicates no X placeholder indices.

Data Types: `double`

`yInd` — Y placeholder indices of UCI
column vector of positive values

Y placeholder indices of the UCI, specified as a column vector of positive values. The input codeword, `in`, at the placeholder locations, `yInd`, is descrambled with the previous values of the scrambling sequence. A value of `[]` indicates no Y placeholder indices.

Data Types: `double`

Output Arguments

`cw` — Descrambled approximate LLR soft bits
numeric column vector

Descrambled approximate LLR soft bits, returned as a numeric column vector. Sign represents descrambled hard bit.

Data Types: `double` | `single`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrPUSCHPRBS` | `nrPUSCHScramble`

Introduced in R2019a

nrPUSCHDMRS

Generate PUSCH DM-RS symbols

Syntax

```
sym = nrPUSCHDMRS(carrier,pusch)
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType',datatype)
```

Description

`sym = nrPUSCHDMRS(carrier,pusch)` returns a matrix containing demodulation reference signal (DM-RS) symbols of physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.1.1 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pusch` specifies the PUSCH configuration parameters.

`sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType',datatype)` specifies the data type for the DM-RS symbols.

Examples

Generate PUSCH DM-RS Symbols for CP-OFDM

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 1;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 2;
pusch.DMRS.NIDNSCID = 10;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS symbols associated with PUSCH of `single` data type.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 624x4 single matrix
```

```
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 + 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
      ⋮
```

Generate PUSCH DM-RS Symbols and Indices

Create a carrier configuration with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 9;
pusch.NStartBWP = 1;
pusch.PRBSet = 0:3;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 5;
```

Create a PUSCH demodulation reference signal (DM-RS) object with specified properties.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 3;
dmrs.GroupHopping = 1;
dmrs.SequenceHopping = 0;
dmrs.NRSID = 10;
```

Assign the PUSCH DM-RS configuration object to DMRS property of PUSCH configuration object.

```
pusch.DMRS = dmrs;
```

Generate PUSCH DM-RS symbols and indices for the specified carrier, PUSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 96x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
```

```
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexBase','0based','IndexOrientation','bwp')
```

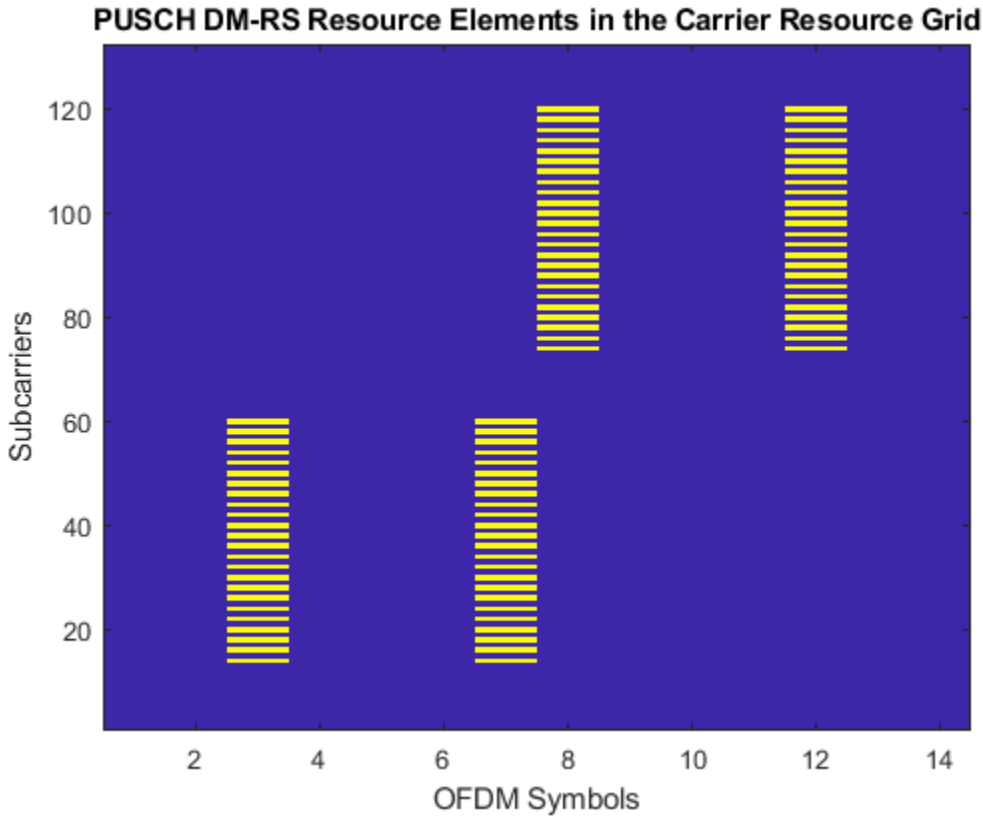
```
ind = 96x1 uint32 column vector

   217
   219
   221
   223
   225
   227
   229
   231
   233
   235
      ⋮
```

Create a bandwidth part (BWP) grid, and then map the DM-RS symbols on the grid.

```
bwp = complex(zeros([pusch.NSizeBWP*12 carrier.SymbolsPerSlot pusch.NumLayers]));
bwp(ind+1) = sym; % Map the DM-RS symbols
```

Map the BWP to the carrier resource grid, and then display the carrier grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers])); % Create ca
offset = pusch.NStartBWP-carrier.NStartGrid; % BWP start location in the carrier grid
grid(offset*12+1:(offset+pusch.NSizeBWP)*12,:,:) = bwp;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`NCellID`	1 (default), integer from 0 to 1007	Physical layer cell identity
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz

Property Field	Values	Description
CyclicPrefix	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these properties of the `nrPUSCHConfig` object.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
NumLayers	1 (default), integer from 1 to 4	Number of transmission layers
MappingType	'A' (default), 'B'	Mapping type of PUSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PUSCH
PRBSet	[0:51] (default), vector of nonnegative integers from 0 to 274	PRBs allocated for PUSCH within the BWP

Property Field	Values	Description
TransformPrecoding	0 (default), 1	Transform precoding flag, specified as one of these values. • `0` indicates that the transform precoding is enabled and waveform type is CP-OFDM. • `1` indicates that the transform precoding is disabled and waveform type is DFT-s-OFDM.
TransmissionScheme	`'nonCodebook'` (default), `'codebook'`	PUSCH transmission scheme
TPMI	0 (default), integer from 0 to 27	Transmitted precoding matrix indicator. This property is applicable when `TransmissionScheme` is set to `'codebook'`.
NumAntennaPorts	1 (default), 2, or 4	Number of antenna ports. The value must be greater than or equal to `NumLayers`. This property is applicable when `TransmissionScheme` is set to `'codebook'`.
FrequencyHopping	`'neither'` (default), `'intraSlot'`, or `'interSlot'`	Frequency hopping configuration for PUSCH
SecondHopStartPRB	1 (default), integer from 0 to 274	Starting PRB index of second hop relative to BWP

DMRS	`nrPUSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties.
		• DMRSConfigurationType
		• DMRSTypeAPosition
		• DMRSLength
		• DMRSAdditionalPosition
		• CustomSymbolSet
		• DMRSPortSet
		• These properties are applicable only when `TransformPrecoding` is set to zero.
		  • NIDNSCID
		  • NSCID
		• These properties are applicable only when `TransformPrecoding` is set to one.
		  • GroupHopping
		  • SequenceHopping
		  • NRSID
		For more information, see `nrPUSCHDMRSConfig`.

datatype — Data type for generated DM-RS symbols
`'double'` (default) | `'single'`

Data type for the generated DM-RS symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — DM-RS symbols

complex matrix

DM-RS symbols, returned as a complex matrix. The number of columns correspond to the number of antenna ports configured.

Data Types: `single` | `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrChannelEstimate` | `nrPUSCH` | `nrPUSCHDMRSIndices` | `nrPUSCHPTRS` | `nrTimingEstimate`

Objects

`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHDMRSConfig`

Introduced in R2020a

nrPUSCHDMRSIndices

Generate PUSCH DM-RS indices

Syntax

```
ind = nrPUSCHDMRSIndices(carrier,pusch)
ind = nrPUSCHDMRSIndices(carrier,pusch,Name,Value)
```

Description

`ind = nrPUSCHDMRSIndices(carrier,pusch)` returns a matrix containing demodulation reference signal (DM-RS) resource element (RE) indices of a physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.1.3 [1]. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology. `pusch` specifies the PUSCH configuration parameters. The returned indices are 1-based using linear indexing form.

`ind = nrPUSCHDMRSIndices(carrier,pusch,Name,Value)` specifies output formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PUSCH DM-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) object with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 2;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 3;
pusch.DMRS.NIDNSCID = 15;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS indices associated to PUSCH of subscript indexing form.

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 3744x3 uint32 matrix
```

```
     2     3     1
     4     3     1
     6     3     1
     8     3     1
    10     3     1
    12     3     1
    14     3     1
    16     3     1
    18     3     1
    20     3     1
     ⋮
```

Generate PUSCH DM-RS Symbols and Indices

Create a carrier configuration with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 9;
pusch.NStartBWP = 1;
pusch.PRBSet = 0:3;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 5;
```

Create a PUSCH demodulation reference signal (DM-RS) object with specified properties.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 3;
dmrs.GroupHopping = 1;
dmrs.SequenceHopping = 0;
dmrs.NRSID = 10;
```

Assign the PUSCH DM-RS configuration object to DMRS property of PUSCH configuration object.

```
pusch.DMRS = dmrs;
```

Generate PUSCH DM-RS symbols and indices for the specified carrier, PUSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 96x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
```

```
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexBase','0based','IndexOrientation','bwp')
```

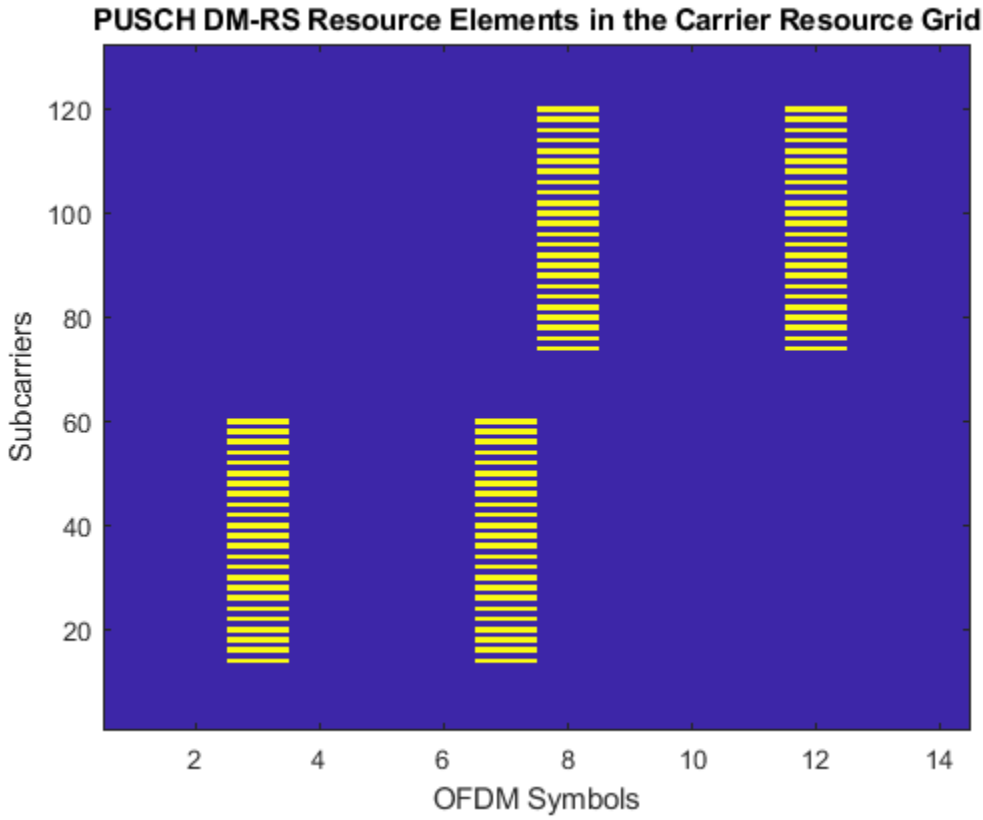
```
ind = 96x1 uint32 column vector

   217
   219
   221
   223
   225
   227
   229
   231
   233
   235
      ⋮
```

Create a bandwidth part (BWP) grid, and then map the DM-RS symbols on the grid.

```
bwp = complex(zeros([pusch.NSizeBWP*12 carrier.SymbolsPerSlot pusch.NumLayers]));
bwp(ind+1) = sym; % Map the DM-RS symbols
```

Map the BWP to the carrier resource grid, and then display the carrier grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers])); % Create ca
offset = pusch.NStartBWP-carrier.NStartGrid; % BWP start location in the carrier grid
grid(offset*12+1:(offset+pusch.NSizeBWP)*12,:,:) = bwp;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz
`CyclicPrefix`	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.

Property Field	Values	Description
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these properties of the `nrPUSCHConfig` object.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
NumLayers	1 (default), integer from 1 to 4	Number of transmission layers
MappingType	'A' (default), 'B'	Mapping type of PUSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PUSCH
PRBSet	[0:51] (default), vector of nonnegative integers from 0 to 274	PRBs allocated for PUSCH within the BWP
TransformPrecoding	0 (default), 1	Transform precoding flag, specified as one of these values. • `0` indicates that the transform precoding is enabled and waveform type is CP-OFDM. • `1` indicates that the transform precoding is disabled and waveform type is DFT-s-OFDM.
TransmissionScheme	`'nonCodebook'` (default), `'codebook'`	PUSCH transmission scheme
NumAntennaPorts	1 (default), 2, or 4	Number of antenna ports. The value must be greater than or equal to `NumLayers`. This property is applicable when `TransmissionScheme` is set to `'codebook'`.
FrequencyHopping	`'neither'` (default), `'intraSlot'`, or `'interSlot'`	Frequency hopping configuration for PUSCH

Property Field	Values	Description
SecondHopStartPRB	1 (default), integer from 0 to 274	Starting PRB index of second hop relative to BWP
DMRS	`nrPUSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet For more information, see `nrPUSCHDMRSConfig`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting properties.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Indexing orientation of resource elements

`'carrier'` (default) | `'bwp'`

Indexing orientation of resource elements, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to carrier grid.
- `'bwp'` — Indices are referenced with respect to bandwidth part.

Data Types: `char` | `string`

Output Arguments

`ind` — DM-RS resource element indices

N-by-*P* matrix | *M*-by-3 matrix

DM-RS resource element indices, returned as one of these values:

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`. The matrix columns correspond to the antenna ports configured.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrChannelEstimate` | `nrPUSCHDMRS` | `nrPUSCHIndices` | `nrPUSCHPTRSIndices` | `nrTimingEstimate`

Objects

`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHDMRSConfig`

Introduced in R2020a

nrPUSCHIndices

Generate PUSCH resource element indices

Syntax

```
[ind,info] = nrPUSCHIndices(carrier,pusch)
[ind,info] = nrPUSCHIndices(carrier,pusch,Name,Value)
```

Description

`[ind,info] = nrPUSCHIndices(carrier,pusch)` returns `ind` in matrix form, which contains 1-based physical uplink shared channel (PUSCH) resource element (RE) indices, as defined in TS 38.211 Sections 6.3.1.6 and 6.3.1.7 [1]. The number of columns in `ind` is equal to the number of configured antenna ports. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pusch` specifies the PUSCH configuration. The function also returns the structure `info`, which contains additional information about the associated physical reference signals, bit capacity, and symbol capacity.

`[ind,info] = nrPUSCHIndices(carrier,pusch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PUSCH Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission. Set the number of antenna ports to 4, modulation scheme to pi/2-BPSK, transmitted precoding matrix indicator to 10, and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.Modulation = 'pi/2-BPSK';
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 10;
```

Generate PUSCH indices in subscript form.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 32448x3 uint32 matrix

    1    1    1
    2    1    1
```

```
     3     1     1
     4     1     1
     5     1     1
     6     1     1
     7     1     1
     8     1     1
     9     1     1
    10     1     1
       ⋮
```

```
info = struct with fields:
                 G: 8112
                Gd: 8112
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PUSCH Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to 30 kHz of subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with specified properties. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
pusch.NSizeBWP = 41;
pusch.Modulation = '16QAM';
pusch.NID = []; % Set NID equal to the NCellID property of carrier.
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
```

Generate PUSCH indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexOrientation','carrier')
```

```
ind = 864x1 uint32 column vector

   121
   122
   123
   124
   125
   126
   127
   128
```

```
   129
   130
      ⋮
```

```
info = struct with fields:
                 G: 3456
                Gd: 864
         NREPerPRB: 144
     DMRSSymbolSet: [2 7]
     PTRSSymbolSet: [1x0 double]
```

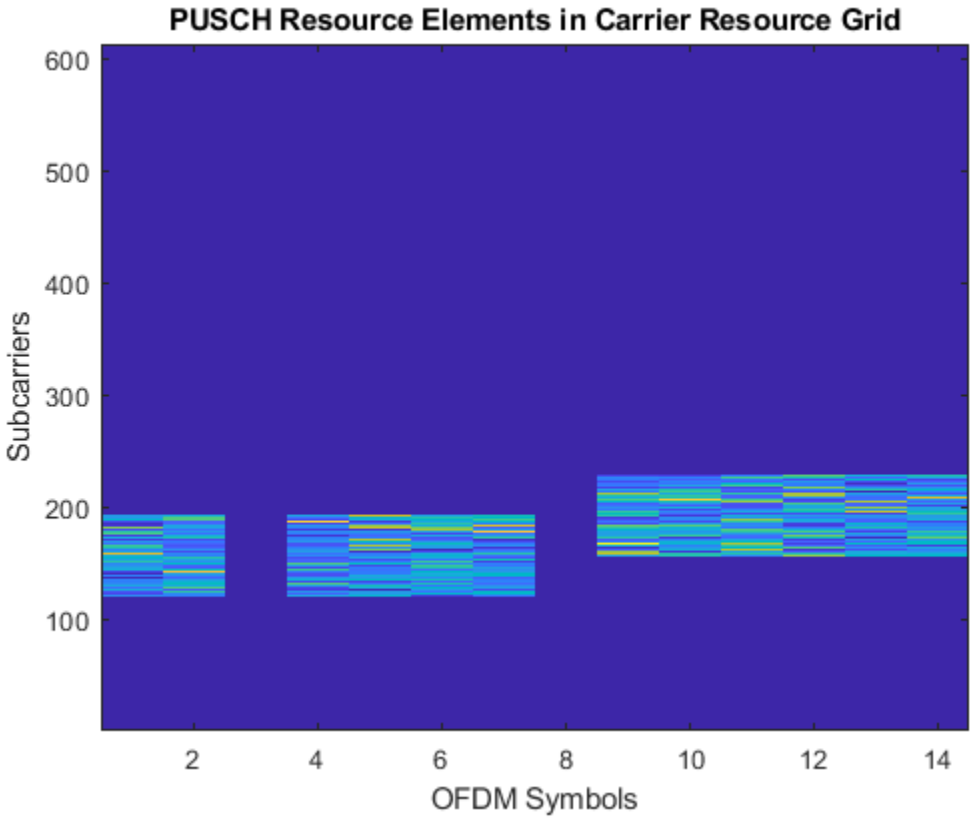
Generate PUSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPUSCH(carrier,pusch,cws,'OutputDataType','single')
```

```
sym = 864x1 single column vector

  -0.7454 + 0.2981i
   0.3406 - 0.2312i
  -0.1153 + 0.2756i
   1.1921 - 0.3658i
  -0.3968 - 0.0277i
  -0.8788 - 0.6493i
  -0.8737 + 0.8318i
  -0.5764 + 0.0269i
  -1.6638 + 0.0482i
  -1.0270 - 0.1347i
      ⋮
```

Plot the generated symbols and indices on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers]));
grid(ind) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH Resource Elements in Carrier Resource Grid');
```



Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz
`CyclicPrefix`	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.

Property Field	Values	Description
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function only uses these `nrPUSCHConfig` object properties.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
Modulation	'QPSK', '16QAM', '64QAM', or '256QAM', 'pi/2-BPSK', string scalar, or character array	Modulation scheme of codeword
NumLayers	1 (default), integer from 1 to 4	Number of transmission layers
MappingType	'A' (default), 'B'	Mapping type of PUSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PUSCH
PRBSet	[0:51] (default), vector of nonnegative integers from 0 to 274	PRBs allocated for PUSCH within the BWP
TransformPrecoding	0 (default), 1	Transform precoding flag, specified as one of these values. • `0` — The transform precoding is disabled and waveform type is CP-OFDM. • `1` — The transform precoding is enabled and waveform type is DFT-s-OFDM.
TransmissionScheme	`'nonCodebook'` (default), `'codebook'`	PUSCH transmission scheme

Property Field	Values	Description
NumAntennaPorts	1 (default), 2, or 4	Number of antenna ports. The value must be greater than or equal to `NumLayers`. This property is applicable only when `TransmissionScheme` is set to `'codebook'`.
FrequencyHopping	`'neither'` (default), `'intraSlot'`, or `'interSlot'`	Frequency hopping mode for PUSCH
SecondHopStartPRB	1 (default), integer from 0 to 274	Starting PRB index of second hop relative to `NStartBWP`
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment
DMRS	`nrPUSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet • NumCDMGroupsWithoutData For more information, see `nrPUSCHDMRSConfig`.
EnablePTRS	0 (default), 1	PT-RS configuration, specified as one of these values. • `0` — Disable PT-RS configuration. • `1` — Enable PT-RS configuration.

PTRS	`nrPUSCHPTRSConfig` configuration object	`PTRS` configuration object uses these properties.
		• TimeDensity
		• These properties are applicable only when `TransformPrecoding` is set to zero.
		  • FrequencyDensity
		  • REOffset
		  • PTRSPortSet
		• These properties are applicable only when `TransformPrecoding` is set to one.
		  • NumPTRSSamples
		  • NumPTRSGroups
		For more information, see `nrPUSCHPTRSConfig`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Resource element indexing orientation

`'carrier'` (default) | `'bwp'`

Resource element indexing orientation, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to the carrier grid.
- `'bwp'` — Indices are referenced with respect to the BWP.

Dependencies

This property is applicable only when `TransformPrecoding` property of `nrPUSCHConfig` object is set to `0`.

Data Types: `char` | `string`

Output Arguments

`ind` — PUSCH resource element indices

N-by-*P* matrix | *M*-by-3 matrix

PUSCH resource element indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`.

- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices.

Data Types: `uint32`

`info` — PUSCH resource information
structure

PUSCH resource information, returned as a structure containing these fields.

Field	Description
`G`	Bit capacity of the PUSCH. This value must be equal to the length of the codeword from the uplink shared channel (UL-SCH) transport channel.
`Gd`	Number of REs per layer or port
`DMRSSymbolSet`	The OFDM symbol locations in a slot containing the demodulation reference signal (DM-RS). The symbols are 0-based.
`NREPerPRB`	Number of REs per PRB allocated to the PUSCH
`PTRSSymbolSet`	The OFDM symbol locations in a slot containing the phase tracking reference signal (PT-RS). The symbols are 0-based.

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUSCH` | `nrPUSCHDMRSIndices` | `nrPUSCHDecode` | `nrPUSCHPTRSIndices`

Objects
`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHDMRSConfig` | `nrPUSCHPTRSConfig`

Introduced in R2020a

nrPUSCHPRBS

Generate PUSCH scrambling sequence

Syntax

```
[seq,cinit] = nrPUSCHPRBS(nid,rnti,n)
[seq,cinit] = nrPUSCHPRBS(nid,rnti,n,Name,Value)
```

Description

`[seq,cinit] = nrPUSCHPRBS(nid,rnti,n)` returns the first `n` elements of the physical uplink shared channel (PUSCH) scrambling sequence. The function also returns initialization value `cinit` of the pseudorandom binary sequence (PRBS) generator. The initialization value depends on scrambling identity `nid` and radio network temporary identifier (RNTI) of the user equipment (UE) `rnti`. The function implements TS 38.211 Section 6.3.1.1 [1].

`[seq,cinit] = nrPUSCHPRBS(nid,rnti,n,Name,Value)` specifies additional output formatting options by using one or more name-value pair arguments. Unspecified name-value pairs take their default values.

Examples

Generate PUSCH Scrambling Sequence

Generate the first 300 elements of the PUSCH scrambling sequence when initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 17;
rnti = 120;
n = 300;
seq = nrPUSCHPRBS(ncellid,rnti,n)
```

```
seq = 300x1 logical array

   0
   1
   1
   0
   1
   1
   0
   1
   0
   0
      ⋮
```

Input Arguments

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

`n` — Number of elements in output sequence

nonnegative integer

Number of elements in output sequence, specified as a nonnegative integer.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'MappingType','signed'` specifies nondefault sequence formatting.

`MappingType` — Output sequence formatting

`'binary'` (default) | `'signed'`

Output sequence formatting, specified as the comma-separated pair consisting of `'MappingType'` and one of these values:

- `'binary'` — This value maps `true` to `1` and `false` to `0`. The data type of the output sequence is `logical`.
- `'signed'` — This value maps `true` to `–1` and `false` to `1`. The data type of the output sequence is `double`. To specify `single` data type, use the `'OutputDataType'` name-value pair.

Data Types: `char` | `string`

`OutputDataType` — Data type of output sequence

`'double'` (default) | `'single'`

Data type of output sequence, specified as the comma-separated pair consisting of `'OutputDataType'` and `'double'` or `'single'`. This name-value pair applies only when `'MappingType'` is set to `'signed'`.

Data Types: `char` | `string`

Output Arguments

`seq` — PUSCH scrambling sequence

logical column vector | numeric column vector

PUSCH scrambling sequence, returned as a logical or numeric column vector. `seq` contains the first `n` elements of the PDSCH scrambling sequence. If you set `'MappingType'` to `'signed'`, the output data type is either `double` or `single`. If you set `'MappingType'` to `'binary'`, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator

nonnegative integer

Initialization value for PRBS generator, returned as a nonnegative integer.

Data Types: `double`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPRBS` | `nrPUSCHDescramble` | `nrPUSCHScramble`

Introduced in R2019a

nrPUSCHPTRS

Generate PUSCH PT-RS symbols

Syntax

```
sym = nrPUSCHPTRS(carrier,pusch)
sym = nrPUSCHPTRS( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPUSCHPTRS(carrier,pusch)` returns `sym` in matrix form, which contains phase tracking reference signal (PT-RS) symbols of physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.2.1 [1]. The number of columns in `sym` depends on the transmission scheme and transform precoding. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pusch` specifies the PUSCH configuration parameters.

`sym = nrPUSCHPTRS( ___ ,'OutputDataType',datatype)` specifies the data type of output PT-RS symbols `sym`, in addition to the input arguments in the previous syntax.

Examples

Generate PUSCH PT-RS Symbols for CP-OFDM

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with 'codebook' based transmission and enable the PT-RS configuration. Set the number of antenna ports to 4, transmitted precoding matrix indicator to 5, frequency density to 4, and resource element offset to '11'. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 5;
pusch.EnablePTRS = 1;
pusch.PTRS.FrequencyDensity = 4;
pusch.PTRS.REOffset = '11';
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 169x4 single matrix

  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.3536 - 0.3536i   0.0000 + 0.0000i
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.3536 - 0.3536i   0.0000 + 0.0000i
```

```
   0.3536 + 0.3536i   0.0000 + 0.0000i  -0.3536 - 0.3536i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.3536 + 0.3536i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.3536 + 0.3536i   0.0000 + 0.0000i
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.3536 - 0.3536i   0.0000 + 0.0000i
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.3536 - 0.3536i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.3536 + 0.3536i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i  -0.3536 + 0.3536i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i  -0.3536 + 0.3536i   0.0000 + 0.0000i
      ⋮
```

Generate PUSCH PT-RS Symbols and Indices

Create a carrier configuration object with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a PUSCH configuration object with intraslot frequency hopping and enable the PT-RS configuration. Set the transform precoding to 1, starting physical resource blocks (PRB) index of the second hop to 3 and PRB set to 0:5. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 192x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
```

```
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
       ⋮
```

Generate PUSCH PT-RS indices in subscript form.

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 192x3 uint32 matrix

    1     1     1
    2     1     1
    3     1     1
    4     1     1
   12     1     1
   13     1     1
   14     1     1
   15     1     1
   21     1     1
   22     1     1
       ⋮
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`NCellID`	1 (default), integer from 0 to 1007	Physical layer cell identity
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz
`CyclicPrefix`	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.

Property Field	Values	Description
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
NumLayers	1 (default), integer from 1 to 4	Number of transmission layers
MappingType	'A' (default), 'B'	Mapping type of PUSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PUSCH
PRBSet	[0:51] (default), vector of nonnegative integers from 0 to 274	PRBs allocated for PUSCH within the BWP
TransformPrecoding	0 (default), 1	Transform precoding flag, specified as one of these values. • `0` — The transform precoding is disabled and waveform type is CP-OFDM. • `1` — The transform precoding is enabled and waveform type is DFT-s-OFDM.
TransmissionScheme	`'nonCodebook'` (default), `'codebook'`	PUSCH transmission scheme
NumAntennaPorts	1 (default), 2, or 4	Number of antenna ports. The value must be greater than or equal to `NumLayers`. This property is applicable only, when `TransmissionScheme` is set to `'codebook'`.
TPMI	0 (default), integer from 0 to 27	Transmitted precoding matrix indicator. This property is applicable only when `TransmissionScheme` is set to `'codebook'`.

Property Field	Values	Description
`FrequencyHopping`	`'neither'` (default), `'intraSlot'`, or `'interSlot'`	Frequency hopping mode for PUSCH
`SecondHopStartPRB`	1 (default), integer from 0 to 274	Starting PRB index of second hop relative to `NStartBWP`
`RNTI`	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment

`DMRS`	`nrPUSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties.
		• DMRSConfigurationType
		• DMRSTypeAPosition
		• DMRSLength
		• DMRSAdditionalPosition
		• CustomSymbolSet
		• DMRSPortSet
		• These properties are applicable only when `TransformPrecoding` is set to zero.
		  • NIDNSCID
		  • NSCID
		• This property are applicable only when `TransformPrecoding` is set to one.
		  • NRSID
		For more information, see `nrPUSCHDMRSConfig`.

`EnablePTRS`	0 (default), 1	PT-RS configuration, specified as one of these values. • `0` — Disable PT-RS configuration. • `1` — Enable PT-RS configuration.

Property Field	Values	Description

PTRS	`nrPUSCHPTRSConfig` configuration object	PTRS configuration object uses these properties.
		• TimeDensity
		• These properties are applicable only when `TransformPrecoding` is set to zero.
		  • FrequencyDensity
		  • REOffset
		  • PTRSPortSet
		• These properties are applicable only when `TransformPrecoding` is set to one.
		  • NumPTRSSamples
		  • NumPTRSGroups
		  • NID
		For more information, see `nrPUSCHPTRSConfig`.

`datatype` — Data type of generated PT-RS symbols
`'double'` (default) | `'single'`

Data type for the generated PT-RS symbols, specified as 'double' or 'single'.

Data Types: `char` | `string`

Output Arguments

`sym` — PT-RS symbols
complex matrix

PT-RS symbols, returned as a complex matrix. The number of columns depends on the `TransmissionScheme` and `TransformPrecoding` properties of `nrPUSCHConfig` object.

The number of columns in `sym` is returned as one these values.

- Number of PT-RS antenna ports configured — When transform precoding is disabled and transmission scheme is non-codebook.
- Number of antenna ports configured — When transform precoding is disabled and transmission scheme is codebook.
- Number of transmission layers — When transform precoding is enabled.

Data Types: `double` | `single`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrPUSCH` | `nrPUSCHDMRS` | `nrPUSCHDecode` | `nrPUSCHPTRSIndices`

Objects

`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHDMRSConfig` | `nrPUSCHPTRSConfig`

Introduced in R2020a

nrPUSCHPTRSIndices

Generate PUSCH PT-RS Indices

Syntax

```
ind = nrPUSCHPTRSIndices(carrier,pusch)
ind = nrPUSCHPTRSIndices(carrier,pusch,Name,Value)
```

Description

`ind = nrPUSCHPTRSIndices(carrier,pusch)` returns `ind` in matrix form, which contains phase tracking reference signal (PT-RS) resource elements (RE) of the physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.2.2 [1]. The number of columns in `ind` depends on the transmission scheme and transform precoding. `carrier` specifies the carrier configuration parameters for a specific OFDM numerology and `pusch` specifies the PUSCH configuration parameters. When you enable transform precoding, the indices are generated relative to the start of the PUSCH allocation.

`ind = nrPUSCHPTRSIndices(carrier,pusch,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate PUSCH PT-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission and enable the PT-RS configuration. Set the number of antenna ports to 4 and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '11';
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS indices in subscript form

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 312x3 uint32 matrix

    21     1     1
    69     1     1
   117     1     1
   165     1     1
   213     1     1
   261     1     1
   309     1     1
   357     1     1
   405     1     1
   453     1     1
     ⋮
```

Generate PUSCH PT-RS Symbols and Indices

Create a carrier configuration object with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a PUSCH configuration object with intraslot frequency hopping and enable the PT-RS configuration. Set the transform precoding to 1, starting physical resource blocks (PRB) index of the second hop to 3 and PRB set to 0:5. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 192x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
       ⋮
```

Generate PUSCH PT-RS indices in subscript form.

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 192x3 uint32 matrix

    1     1     1
    2     1     1
    3     1     1
    4     1     1
   12     1     1
   13     1     1
   14     1     1
   15     1     1
   21     1     1
   22     1     1
       ⋮
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. This function uses only these properties of the `nrCarrierConfig` object.

Property Field	Values	Description
`SubcarrierSpacing`	15 (default), 30, 60, 120, 240	Subcarrier spacing in kHz

Property Field	Values	Description
CyclicPrefix	'normal' (default), 'extended'	Cyclic prefix length, specified as one of these options. • `'normal'` corresponds to 14 OFDM symbols in a slot. • `'extended'` corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies only for 60 kHz subcarrier spacing.
NSizeGrid	52 (default), integer from 1 to 275	Number of resource blocks in the carrier resource grid
NStartGrid	0 (default), integer from 0 to 2199	Start of carrier resource grid relative to common resource block 0 (CRB 0)
NSlot	0 (default), nonnegative integer scalar	Slot number. You can set `NSlot` to a value larger than the number of slots per frame.

`pusch` — PUSCH configuration parameters
`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

Property Field	Values	Description
NSizeBWP	`[]` (default), integer from 1 to 275	Size of the bandwidth part (BWP) in terms of number of physical resource blocks (PRBs). The default value of `[]` implies that the value is equal to `NSizeGrid` of `carrier` input argument.
NStartBWP	`[]` (default), integer from 0 to 2473	Starting PRB index of BWP relative to CRB 0. The default value of `[]` implies that the value is equal to `NStartGrid` of `carrier` input argument.
NumLayers	1 (default), integer from 1 to 4	Number of transmission layers
MappingType	'A' (default), 'B'	Mapping type of PUSCH
SymbolAllocation	[0 14] (default), two-element vector of nonnegative integers	OFDM symbols allocated for PUSCH
PRBSet	[0:51] (default), vector of nonnegative integers from 0 to 274	PRBs allocated for PUSCH within the BWP

Property Field	Values	Description
TransformPrecoding	0 (default), 1	Transform precoding flag, specified as one of these values. • `0` — The transform precoding is disabled and waveform type is CP-OFDM. • `1` — The transform precoding is enabled and waveform type is DFT-s-OFDM.
TransmissionScheme	`'nonCodebook'` (default), `'codebook'`	PUSCH transmission scheme
NumAntennaPorts	1 (default), 2, or 4	Number of antenna ports. The value must be greater than or equal to `NumLayers`. This property is applicable only, when `TransmissionScheme` is set to `'codebook'`.
FrequencyHopping	`'neither'` (default), `'intraSlot'`, or `'interSlot'`	Frequency hopping mode for PUSCH
SecondHopStartPRB	1 (default), integer from 0 to 274	Starting PRB index of second hop relative to `NStartBWP`
RNTI	1 (default), integer from 0 to 65,535	Radio network temporary identifier of the user equipment
DMRS	`nrPUSCHDMRSConfig` configuration object	`DMRS` configuration object only uses these properties. • DMRSConfigurationType • DMRSTypeAPosition • DMRSLength • DMRSAdditionalPosition • CustomSymbolSet • DMRSPortSet For more information, see `nrPUSCHDMRSConfig`.
EnablePTRS	0 (default), 1	PT-RS configuration, specified as one of these values. • `0` — Disable PT-RS configuration. • `1` — Enable PT-RS configuration.

Property Field	Values	Description

PTRS	`nrPUSCHPTRSConfig` configuration object	PTRS configuration object uses these properties.
		• TimeDensity
		• These properties are applicable only when `TransformPrecoding` is set to zero.
		  • FrequencyDensity
		  • REOffset
		  • PTRSPortSet
		• These properties are applicable only when `TransformPrecoding` is set to one.
		  • NumPTRSSamples
		  • NumPTRSGroups
		For more information, see `nrPUSCHPTRSConfig`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the RE indexing form and base, respectively, of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base

`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

`IndexOrientation` — Resource element indexing orientation

`'carrier'` (default) | `'bwp'`

Resource element indexing orientation, specified as the comma-separated pair consisting of `'IndexOrientation'` and one of these values:

- `'carrier'` — Indices are referenced with respect to carrier grid.
- `'bwp'` — Indices are referenced with respect to bandwidth part.

Dependencies

This property is applicable only when `TransformPrecoding` is set to `0`.

Data Types: `char` | `string`

Output Arguments

`ind` — PT-RS resource element indices

N-by-*P* matrix | *M*-by-3 matrix

PT-RS resource element indices, returned as one of these values.

- *N*-by-*P* matrix — The function returns this type of value when `'IndexStyle'` is set to `'index'`. The number of columns depends on the `TransmissionScheme` and `TransformPrecoding` properties of `nrPUSCHConfig` object and returned as one these values.
  - Number of PT-RS antenna ports configured — When transform precoding is disabled and transmission scheme is non-codebook.
  - Number of antenna ports configured — When transform precoding is disabled and transmission scheme is codebook.
  - Number of transmission layers — When transform precoding is enabled.
- *M*-by-3 matrix — The function returns this type of value when `'IndexStyle'` is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

Depending on the value of `'IndexBase'`, the function returns either 1-based or 0-based indices. Depending on the value of `'IndexOrientation'`, the function returns either carrier oriented indices or BWP oriented indices. This index orientation is applicable only when `TransformPrecoding` is set to `0`. For DFT-s-OFDM, the indices are relative to the shared channel allocation.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPUSCH` | `nrPUSCHDMRSIndices` | `nrPUSCHDecode` | `nrPUSCHPTRS`

Objects
`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHDMRSConfig` | `nrPUSCHPTRSConfig`

Introduced in R2020a

nrPUSCHScramble

Perform PUSCH scrambling

Syntax

```
scrambled = nrPUSCHScramble(cw,nid,rnti)
```

Description

`scrambled = nrPUSCHScramble(cw,nid,rnti)` returns a column vector resulting from physical uplink shared channel (PUSCH) scrambling, as defined in TS 38.211 Section 6.3.1.1 [1]. `cw` is an uplink shared channel (UL-SCH) codeword, as described in TS 38.212 Section 6.2.7 [2]. `nid` is the scrambling identity, and `rnti` is the radio network temporary identifier (RNTI) of the user equipment (UE).

Examples

Perform PUSCH Scrambling

Create a random sequence of binary values corresponding to a codeword containing 5000 bits.

```
cw = randi([0 1],5000,1)
```

```
cw = 5000×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
     ⋮
```

Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti)
```

```
scrambled = 5000x1 logical array

   0
   1
   1
   1
```

```
   1
   0
   1
   0
   0
   1
      ⋮
```

Input Arguments

`cw` — UL-SCH codeword

column vector of integers from –2 to 1

UL-SCH codeword from TS 38.212 Section 6.2.7, specified as a column vector of integers from –2 to 1.

- 0 and 1 represent false and true bit values, respectively.
- –1 and –2 represent *x* and *y* placeholders in the uplink control information (UCI), respectively. For more details, see TS 38.212 Sections 5.3.3.1 and 5.3.3.2.

Data Types: `double` | `int8`

`nid` — Scrambling identity

integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. `nid` is higher layer parameter *dataScramblingIdentityPUSCH*, ranging from 0 to 1023, if the higher layer parameter is configured. Otherwise, `nid` is physical layer cell identity number *NCellID*, ranging from 0 to 1007. For more information, see TS 38.211 Section 6.3.1.1.

Data Types: `double`

`rnti` — RNTI of UE

integer from 0 to 65,535

RNTI of the UE, specified as an integer from 0 to 65,535.

Data Types: `double`

Output Arguments

`scrambled` — Scrambled UL-SCH codeword

logical column vector

Scrambled UL-SCH codeword, returned as a logical column vector.

Data Types: `logical`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPUSCHDescramble` | `nrPUSCHPRBS`

Introduced in R2019a

nrRateMatchLDPC

Low-density parity-check (LDPC) rate matching

Syntax

```
out = nrRateMatchLDPC(in,outlen,rv,mod,nLayers)
out = nrRateMatchLDPC( ___ ,Nref)
```

Description

`out = nrRateMatchLDPC(in,outlen,rv,mod,nLayers)` returns the rate-matched output of length `outlen` for input data matrix `in`. The input `rv` is the redundancy version, `mod` is the modulation type, and `nLayers` is the number of transmission layers. The internal buffer used for the soft input has no size limits.

`nrRateMatchLDPC` includes the stages of bit selection and interleaving defined for LDPC-encoded data and code block concatenation, as specified in TS 38.212 Sections 5.4.2 and 5.5 [1].

`out = nrRateMatchLDPC( ___ ,Nref)` returns the rate-matched output for a limited soft buffer size `Nref`, in addition to the input arguments in the previous syntax. `Nref` is defined in TS 38.212 Section 5.4.2.1 [1].

Examples

Perform LDPC Rate Matching

Create input data corresponding to two LDPC-encoded code blocks of length 3960.

```
encoded = ones(3960,2);
```

Perform LDPC rate matching of the two code blocks to a vector of length 8000. Use single transmission layer with QPSK modulation and zero redundancy version.

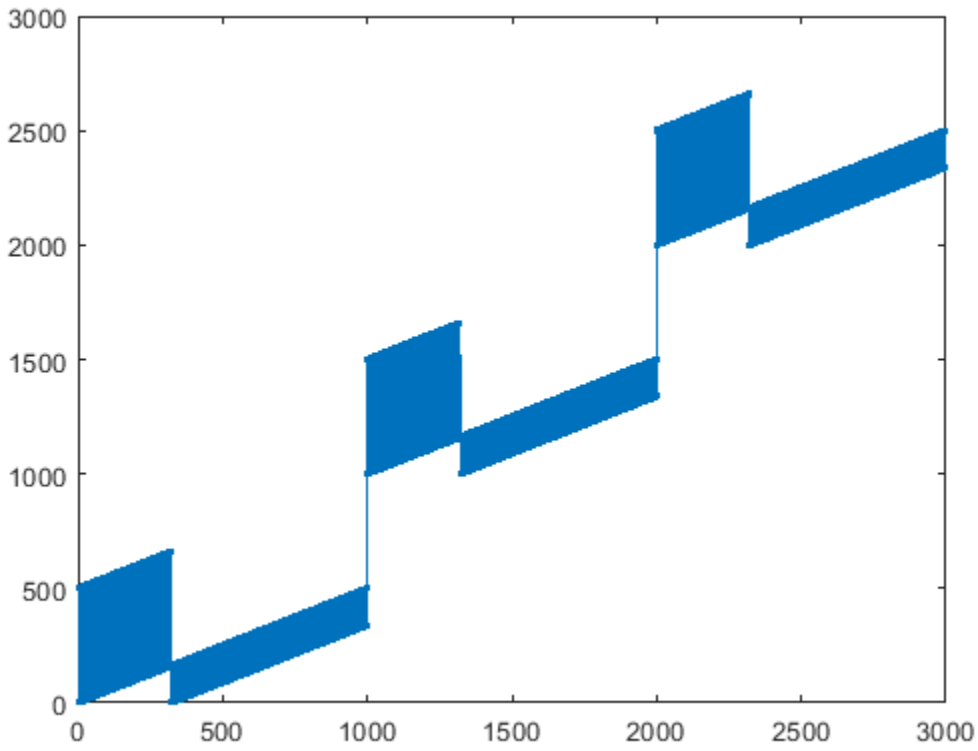
```
rv = 0;
mod = 'QPSK';
nLayers = 1;
outlen = 8000;
ratematched = nrRateMatchLDPC(encoded,outlen,rv,mod,nLayers);
size(ratematched)
```

```
ans = 1×2

        8000           1
```

Plot Output Locations in LDPC Rate-Matched Code Blocks

Create LDPC-encoded input data consisting of integer ramps in separate code blocks. Perform LDPC rate matching of the code blocks to a vector of length 3000. Use single transmission layer with QPSK modulation and zero redundancy version. Plot the locations of the rate-matched output data.

```
encoded = [0 1000 2000] + (1:66*10)';
rv = 0;
mod = 'QPSK';
nLayers = 1;
outlen = 3000;
out = nrRateMatchLDPC(encoded,outlen,rv,mod,nLayers);
plot(out,'.-')
```



Input Arguments

`in` — LDPC-encoded input data

matrix

LDPC-encoded input data, specified as a matrix. Each column of `in` is a codeword. The number of columns in the input argument `in` is equal to the number of scheduled code blocks in a transport block. Each column is rate-matched separately, and the results are concatenated in `out`.

Data Types: `double` | `int8`

`outlen` — Length of output vector

positive integer

Length of the rate-matched and concatenated output vector, specified as a positive integer. `outlen` is the number of coded bits available for transmission in the transport block, as specified in TS 38.212 Section 5.4.2.1 [1].

The modulation scheme `mod` determines the modulation order Q_m(number of bits used per modulation symbol). If `outlen` is not a multiple of `nLayers` × Q_m, the function sets the length of the output vector to the next multiple of `nLayers` × Q_m.

Data Types: `double`

`rv` — Redundancy version

integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type of the codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers

integer from 1 to 4

Number of transmission layers associated with the transport block, specified as an integer from 1 to 4.

Data Types: `double`

`Nref` — Limited buffer rate matching

positive integer

Limited buffer rate matching, specified as a positive integer. `Nref` is defined in TS 38.212 Section 5.4.2.1.

Data Types: `double`

Output Arguments

out — Rate-matched and concatenated code blocks for transport block

vector

Rate-matched and concatenated code blocks for a transport block, returned as a vector of length `outlen`.

Data Types: `double` | `int8`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCRCEncode` | `nrCodeBlockSegmentLDPC` | `nrLDPCEncode` | `nrRateRecoverLDPC`

Introduced in R2018b

nrRateMatchPolar

Polar rate matching

Syntax

```
rm = nrRateMatchPolar(enc,K,E)
rm = nrRateMatchPolar(enc,K,E,ibil)
```

Description

`rm = nrRateMatchPolar(enc,K,E)` returns the rate-matched output of length `E` for the polar-encoded input `enc` and information block length `K`, as specified in TS 38.212 Section 5.4.1 [1]. In this syntax, coded-bit interleaving is disabled. Use this syntax for downlink (DL) configuration.

`rm = nrRateMatchPolar(enc,K,E,ibil)` controls coded-bit interleaving. To enable coded-bit interleaving, set `ibil` to `true`. Use this syntax for uplink (UL) configuration with coded-bit interleaving enabled.

Examples

Perform Polar Rate Matching

Create a polar encoded random block of 512 bits and perform polar rate matching. Specify an information block of 56 bits and a rate-matched output of 864 bits.

```
N = 2^9;
K = 56;
E = 864;
in = randi([0 1],N,1);
out = nrRateMatchPolar(in,K,E)
```

```
out = 864×1

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1
     ⋮
```

Input Arguments

`enc` — Polar-encoded message

column vector of binary values

Polar-encoded message, specified as a column vector of binary values.

The length of the polar-encoded message, N, is a power of two. For more information, see TS 38.212 Section 5.3.1.

- For DL configuration, $N \le 512$.
- For UL configuration, $N \le 1024$.

Data Types: `double` | `int8`

`K` — Length of information block in bits

positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable.

Data Types: `double`

`E` — Rate-matched output length in bits

positive integer

Rate-matched output length in bits, specified as a positive integer.

- If $18 \le K \le 25$, `E` must be in the range $K + 3 < E \le 8192$.
- If $K > 30$, `E` must be in the range $K < E \le 8192$.

Data Types: `double`

`ibil` — Coded-bit interleaving

`false` (default) | `true`

Coded-bit interleaving, specified as `false` or `true`.

- For DL configuration, specify `false`.
- For UL configuration, specify `true`.

Data Types: `logical`

Output Arguments

`rm` — Rate-matched output data

column vector of binary values

Rate-matched output data, returned as an `E`-by-1 column vector of binary values. `rm` inherits its data type from the encoded message `enc`.

Data Types: `double` | `int8`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCRCEncode` | `nrDCIEncode` | `nrPolarEncode` | `nrRateRecoverPolar` | `nrUCIEncode`

Topics

“5G New Radio Polar Coding”

Introduced in R2018b

nrRateRecoverLDPC

Low-density parity-check (LDPC) rate recovery

Syntax

```
out = nrRateRecoverLDPC(in,trblklen,R,rv,mod,nLayers)
out = nrRateRecoverLDPC( ___ ,numCB)
out = nrRateRecoverLDPC( ___ ,numCB,Nref)
```

Description

`out = nrRateRecoverLDPC(in,trblklen,R,rv,mod,nLayers)` returns the rate-recovered output representing the LDPC-encoded code blocks for input data vector `in`. The input `trblklen` is the transport block length, `R` is the target code rate, `rv` is the redundancy version, `mod` is the modulation type, and `nLayers` is the number of transmission layers. The internal buffer used for the soft input has no size limits, and the output contains the total number of code blocks.

`nrRateRecoverLDPC` is the inverse of `nrRateMatchLDPC` and performs the inverse of the code block concatenation, bit interleaving, and bit selection stages at the receiver end.

`out = nrRateRecoverLDPC( ___ ,numCB)` specifies the number of code blocks `numCB` to be recovered, in addition to the input arguments in the previous syntax.

`out = nrRateRecoverLDPC( ___ ,numCB,Nref)` returns the rate-recovered output for a limited soft buffer size `Nref` with the specified number of code blocks `numCB` to recover, in addition to the input arguments in the first syntax. `Nref` is defined in TS 38.212 Section 5.4.2.1 [1].

Examples

Perform LDPC Rate Recovery

Create input data of length 4500 corresponding to soft bits. The length of the original transport block is 4000. Perform LDPC rate recovery of the input to one code block. Use single transmission layer with QPSK modulation and zero redundancy version.

```
sbits = ones(4500,1);
trblklen = 4000;
R = 0.5;
rv = 0;
mod = 'QPSK';
nlayers = 1;
numCB = 1;
raterec = nrRateRecoverLDPC(sbits,trblklen,R,rv,mod,nlayers,numCB);
size(raterec)
```

```
ans = 1×2

       12672           1
```

Input Arguments

`in` — Received soft bits before code block desegmentation

vector

Received soft bits before code block desegmentation, specified as a vector.

Data Types: `double` | `single`

`trblklen` — Original transport block length

nonnegative integer

Original transport block length, specified as a nonnegative integer.

Data Types: `double`

`R` — Target code rate

real scalar in the range (0,1)

Target code rate, specified as a real scalar in the range (0,1).

Data Types: `double`

`rv` — Redundancy version

integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type of the codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers

integer from 1 to 4

Number of transmission layers associated with the transport block, specified as an integer from 1 to 4.

Data Types: `double`

numCB — Number of scheduled code block segments
positive integer

Number of scheduled code block segments, specified as a positive integer. `numCB` is less than or equal to the number of code block segments for a transport block.

Data Types: `double`

Nref — Limited buffer rate matching
positive integer

Limited buffer rate matching, specified as a positive integer. `Nref` is defined in TS 38.212 Section 5.4.2.1.

Data Types: `double`

Output Arguments

out — Rate-recovered scheduled code block segments
matrix

Rate-recovered scheduled code segments, returned as a matrix. The number of rows in `out` is calculated from `trblklen` and `R`. The number of columns in `out` is equal to `numCB`, or the total number of code blocks for a transport block. Filler bits are set to `Inf` to correspond to zeros used during their encoding.

Data Types: `double` | `single`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`nrCRCDecode` | `nrCodeBlockSegmentLDPC` | `nrLDPCDecode` | `nrRateMatchLDPC`

Introduced in R2018b

nrRateRecoverPolar

Polar rate recovering

Syntax

```
rec = nrRateRecoverPolar(llr,K,N)
rec = nrRateRecoverPolar(llr,K,N,ibil)
```

Description

`rec = nrRateRecoverPolar(llr,K,N)` returns the rate-recovered output of length `N` for the soft input `llr` and information block length `K`, as specified in TS 38.212 Section 5.4.1 [1]. In this syntax, coded-bit deinterleaving is disabled. Use this syntax for downlink (DL) configuration.

`rec = nrRateRecoverPolar(llr,K,N,ibil)` controls coded-bit deinterleaving. To enable coded-bit deinterleaving, set `ibil` to `true`. Use this syntax for uplink (UL) configuration with coded-bit deinterleaving enabled.

Examples

Perform Polar Rate Recovery

Create a polar encoded random block of 512 bits and perform polar rate matching using `nrRateMatchPolar`. Perform polar rate recovery. Verify the results are identical to the original polar encoded input.

Specify an information block of 56 bits and an output of 864 bits for rate matching.

```
N = 512;
K = 56;
E = 864;
in = randi([0 1],N,1);
rateMatched = nrRateMatchPolar(in,K,E);
```

Perform rate recovery of the rate-matched data and information block of 56 bits. The length of the rate-recovered output, `N`, is the same as the length of the original polar encoded message.

```
rateRecovered = nrRateRecoverPolar(rateMatched,K,N);
```

Verify that the rate recovered output is identical to the original polar encoded input `in`.

```
isequal(rateRecovered,in)
```

```
ans = logical
   1
```

Input Arguments

`llr` — Log-likelihood ratio value input

column vector of real values

Log-likelihood ratio value input, specified as a column vector of real values. `llr` is the soft-demodulated input of length E, the same length as the rate-matched data vector before modulation.

Data Types: `single` | `double`

`K` — Length of information block in bits

positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable.

Data Types: `double`

`N` — Length of polar-encoded message in bits

power of two

Length of polar-encoded message in bits, specified as a power of two.

- $N \leq 512$ for DL configuration.
- $N \leq 1024$ for UL configuration.

For more details, see TS 38.212 Section 5.3.1 [1].

Data Types: `double`

`ibil` — Coded-bit deinterleaving

`false` for DL (default) | `true` for UL

Coded-bit deinterleaving, specified as `false` or `true`.

- For DL configuration, specify `false`.
- For UL configuration, specify `true`.

Data Types: `logical`

Output Arguments

`rec` — Rate-recovered output

column vector of real numbers

Rate-recovered output, returned as an `N`-by-1 column vector of real numbers.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCRCDecode` | `nrDCIDecode` | `nrPolarDecode` | `nrRateMatchPolar` | `nrUCIDecode`

Topics

“5G New Radio Polar Coding”

Introduced in R2018b

nrResourceGrid

Generate empty carrier slot resource grid

Syntax

```
grid = nrResourceGrid(carrier)
grid = nrResourceGrid(carrier,p)
grid = nrResourceGrid( ___ ,'OutputDataType',type)
```

Description

`grid = nrResourceGrid(carrier)` generates an empty carrier slot resource grid for one antenna and the specified carrier configuration parameters.

`grid = nrResourceGrid(carrier,p)` generates an empty carrier slot resource grid for the specified number of antennas.

`grid = nrResourceGrid( ___ ,'OutputDataType',type)` specifies the data type of the generated grid in addition to any combination of input arguments from the previous syntaxes.

Examples

Generate OFDM Modulated Waveform

Generate a waveform by performing OFDM modulation of a resource array that contains sounding reference signals (SRSs). The resource array spans an entire frame.

Set carrier configuration parameters, specifying a subcarrier spacing of 30 kHz and 24 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',24);
```

Configure SRS parameters, setting the slot periodicity to 2 and the offset to zero.

```
srs = nrSRSConfig('SRSPeriod',[2 0]);
```

Get OFDM information for the specified carrier configuration.

```
info = nrOFDMInfo(carrier);
```

Produce the frame resource array by creating and concatenating individual slot resource arrays.

```
grid = [];
for nslot = 0:(info.SlotsPerFrame - 1)
    carrier.NSlot = nslot;
    slotGrid = nrResourceGrid(carrier);
    ind = nrSRSIndices(carrier,srs);
    sym = nrSRS(carrier,srs);
    slotGrid(ind) = sym;
    grid = [grid slotGrid];
end
```

Perform OFDM modulation on the resource array for the specified carrier configuration.

```
[waveform,info] = nrOFDMModulate(carrier,grid);
```

Demodulate OFDM Waveform

Recover a transmitted carrier resource array by demodulating an OFDM waveform.

Set carrier configuration parameters, specifying 106 resource blocks (RBs) in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Generate physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) symbols and indices.

```
p = 2;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array containing the PDSCH DM-RS symbols.

```
txGrid = nrResourceGrid(carrier,p);
txGrid(ind) = sym;
```

Generate OFDM modulated waveform.

```
[txWaveform,~] = nrOFDMModulate(carrier,txGrid);
```

Pass the waveform through a simple 2-by-1 channel.

```
H = [0.6; 0.4];
waveform = txWaveform*H;
```

Recover the carrier resource array by demodulating the received OFDM waveform.

```
grid = nrOFDMDemodulate(carrier,waveform);
```

Generate OFDM Modulated Waveform for Specified Sample Rate

Generate a waveform by performing OFDM modulation of a resource array that contains PDSCH DM-RS symbols.

Set carrier configuration parameters, specifying 106 RBs in the carrier resource array.

```
carrier = nrCarrierConfig('NSizeGrid',106);
```

Configure PDSCH and generate the corresponding symbols and indices.

```
p = 4;
pdsch = nrPDSCHConfig('NumLayers',p);
sym = nrPDSCHDMRS(carrier,pdsch);
ind = nrPDSCHDMRSIndices(carrier,pdsch);
```

Create a carrier resource array and map the PDSCH symbols.

```
grid = nrResourceGrid(carrier,p,'OutputDataType','single');
grid(ind) = sym;
```

Generate OFDM modulated waveform, specifying the sample rate.

```
sr = 1e8;
[waveform,info] = nrOFDMModulate(carrier,grid,'SampleRate',sr);
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid

52 (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of 52 corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`p` — Number of antennas

positive integer

Number of antennas, specified as a positive integer.

Data Types: `double`

`type` — Data type of generated grid

`'double'` (default) | `'single'`

Output data type of generated grid, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`grid` — Empty carrier slot resource grid

complex-valued array

Empty carrier slot resource array, returned as a complex-valued array of size *K*-by-*L*-by-`p`.

- *K* is the number of subcarriers.
- *L* is the number of OFDM symbols.

Data Types: `single` | `double`
Complex Number Support: Yes

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrOFDMDemodulate` | `nrOFDMInfo` | `nrOFDMModulate`

Objects

`nrCarrierConfig`

Introduced in R2019b

nrSRS

Generate uplink SRS symbols

Syntax

```
[sym,info] = nrSRS(carrier,srs)
[sym,info] = nrSRS(carrier,srs,'OutputDataType',datatype)
```

Description

`[sym,info] = nrSRS(carrier,srs)` returns uplink sounding reference signal (SRS) symbols, as defined in TS 38.211 section 6.4.1.4.2 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `srs` specifies SRS configuration parameters. The function also returns the structure `info`, which contains information about the SRS generation process.

`[sym,info] = nrSRS(carrier,srs,'OutputDataType',datatype)` specifies the data type of the SRS symbols.

Examples

Generate SRS Symbols for Two-Port Transmission

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure a two-port SRS transmission of 4 OFDM symbols.

```
srs = nrSRSConfig;
srs.NumSRSPorts = 2;
srs.NumSRSSymbols = 4;
```

The SRS must be located in the last six symbols of the slot. Set the time-domain starting position of the SRS to 8 and the bandwidth configuration index to 5.

```
srs.SymbolStart = 8;
srs.CSRS = 5;
```

Generate SRS symbols for the specified carrier and SRS configuration parameters.

```
[sym,info] = nrSRS(carrier,srs);
```

Verify that the symbols vector contains two columns corresponding to the two-port transmission.

```
size(sym)
```

```
ans = 1×2

   480     2
```

Verify the number of SRS symbols per port.

```
isequal(info.SeqLength*srs.NumSRSSymbols,size(sym,1))
```

```
ans = logical
   1
```

Generate and Map SRS Symbols to Carrier Grid

Configure the SRS and the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
srs = nrSRSConfig;
```

Generate SRS symbols and indices using the specified carrier and SRS configuration parameters.

```
srsSym = nrSRS(carrier,srs);
srsInd = nrSRSIndices(carrier,srs);
```

Create a carrier grid corresponding to the number of subcarriers, OFDM symbols, and number of antenna ports specified in the configuration objects.

```
K = carrier.NSizeGrid*12;       % Number of subcarriers
L = carrier.SymbolsPerSlot;     % Number of OFDM symbols per slot
P = srs.NumSRSPorts;  % Number of antenna ports
gridSize = [K L P];
```

Initialize the carrier grid for one slot with all zeros.

```
slotGrid = complex(zeros(gridSize));
```

Map the SRS symbols to the carrier grid using the indices.

```
slotGrid(srsInd) = srsSym;
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`srs` — SRS configuration parameters

`nrSRSonfig` object

SRS configuration parameters, specified as an `nrSRSConfig` object.

`datatype` — Data type of output symbols

`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — SRS symbols

complex column vector | complex matrix | `[]`

SRS symbols, returned as a complex column vector, complex matrix, or empty array. The number of transmission antenna ports specified by the `NumSRSPorts` property of the `srs` input determines the number of columns. The symbols in a column correspond to one antenna port.

The function returns an empty array when the slot number specified by `carrier.NSlot` is not a candidate slot, as defined in TS 38.211 Section 6.4.1.4.4, or when the `srs.SRSPeriod` property is set to `'off'`.

Data Types: `single` | `double`

`info` — Information about SRS generation

structure

Information about the SRS generation, returned as a structure containing these fields:

Fields	Description
`SeqGroup`	Base sequence group number per OFDM symbol (parameter u in TS 38.211 Section 6.4.1.4.2)
`NSeq`	Base sequence number per OFDM symbol (parameter v in TS 38.211 Section 6.4.1.4.2)
`Alpha`	Reference signal cyclic shift per port (parameter α_i in TS 38.211 Section 6.4.1.4.2)
`SeqLength`	Zadoff-Chu sequence length (parameter $M_{sc,b}^{RS}$ in TS 38.211 Section 6.4.1.4.2)

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrSRSIndices`

Objects
`nrCarrierConfig` | `nrSRSConfig`

Topics
“NR SRS Configuration”

Introduced in R2020a

nrSRSIndices

Generate uplink SRS resource element indices

Syntax

```
[ind,info] = nrSRSIndices(carrier,srs)
[ind,info] = nrSRSIndices(carrier,srs,Name,Value)
```

Description

`[ind,info] = nrSRSIndices(carrier,srs)` returns resource element indices `ind` for the uplink sounding reference signal (SRS), as defined in TS 38.211 section 6.4.1.4.3 [1]. The input `carrier` specifies carrier configuration parameters for a specific OFDM numerology. The input `srs` specifies SRS configuration parameters.

`[ind,info] = nrSRSIndices(carrier,srs,Name,Value)` specifies output formatting options using one or more name-value pair arguments. Unspecified options take default values.

Examples

Generate SRS Indices for Two-Port Transmission

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure a two-port SRS transmission of 4 OFDM symbols.

```
srs = nrSRSConfig;
srs.NumSRSPorts = 2;
srs.NumSRSSymbols = 4;
```

Set the time-domain starting position of the SRS to 8 and the bandwidth configuration index to 5.

```
srs.SymbolStart = 8;
srs.CSRS = 5;
```

Generate SRS resource element indices for the specified carrier and SRS configuration parameters.

```
ind = nrSRSIndices(carrier,srs,'IndexStyle','subscript');
```

Verify that the index matrix has three columns corresponding to the [subcarrier, symbol, antenna] subscripts.

```
size(ind)
```

```
ans = 1×2

   960     3
```

Generate and Map SRS Symbols to Carrier Grid

Configure the SRS and the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
srs = nrSRSConfig;
```

Generate SRS symbols and indices using the specified carrier and SRS configuration parameters.

```
srsSym = nrSRS(carrier,srs);
srsInd = nrSRSIndices(carrier,srs);
```

Create a carrier grid corresponding to the number of subcarriers, OFDM symbols, and number of antenna ports specified in the configuration objects.

```
K = carrier.NSizeGrid*12;       % Number of subcarriers
L = carrier.SymbolsPerSlot;     % Number of OFDM symbols per slot
P = srs.NumSRSPorts;  % Number of antenna ports
gridSize = [K L P];
```

Initialize the carrier grid for one slot with all zeros.

```
slotGrid = complex(zeros(gridSize));
```

Map the SRS symbols to the carrier grid using the indices.

```
slotGrid(srsInd) = srsSym;
```

Input Arguments

`carrier` — Carrier configuration parameters

`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object.

`srs` — SRS configuration parameters

`nrSRSonfig` object

SRS configuration parameters, specified as an `nrSRSConfig` object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies the indexing style and indexing base of the output.

`IndexStyle` — Resource element indexing form

`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

Output Arguments

`ind` — SRS resource element indices
N-by-*P* matrix (default) | *M*-by-3 matrix

SRS resource element indices, returned as one of these values:

- *N*-by-*P* matrix — When '`IndexStyle`' is set to `'index'` and where *P* is the number of antenna ports.
- *M*-by-3 matrix — When '`IndexStyle`' is set to `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers, OFDM symbols, and number of antennas, respectively.

The number of rows depends on the SRS configuration specified by `srs`. The `NumSRSPorts` property of `srs` determines the number of antenna ports. Depending on '`IndexBase`', the indices are either 1-based or 0-based.

Data Types: `uint32`

`info` — Information about SRS index generation
structure

Information about the SRS index generation, returned as a structure containing these fields:

Fields	Values	Description
`SubcarrierOffset`	`srs`.`NumSRSPorts`-by-`srs`.`NumSRSSymbols` integer matrix	Frequency starting position per antenna port and OFDM symbol (parameter k_0 in TS 38.211 Section 6.4.1.4.3)
`FreqIndex`	(`srs`.BSRS + 1)-by-`srs`.`NumSRSSymbols` integer matrix	Frequency position index per OFDM symbol (parameter n_b in TS 38.211 Section 6.4.1.4.3, where b is an integer from 0 to `srs`.BSRS)

Fields	Values	Description
`HoppingOffset`	(`srs`.BSRS – `srs`.BHop)-by-`srs`.`NumSRSSymbols` integer matrix	Hopping offset per OFDM symbol (parameter F_b in TS 38.211 Section 6.4.1.4.3, where b is an integer from `srs`.BHop + 1 to `srs`.BSRS)
`PRBSet`	`srs`.`NRBPerTransmission` -by-`srs`.`NumSRSSymbols` integer matrix	Resource blocks allocated for SRS per OFDM symbol

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrSRS`

Objects

`nrCarrierConfig` | `nrSRSConfig`

Topics

“NR SRS Configuration”

Introduced in R2020a

nrSSS

Generate SSS symbols

Syntax

```
sym = nrSSS(ncellid)
sym = nrSSS(ncellid,'OutputDataType',datatype)
```

Description

`sym = nrSSS(ncellid)` returns the secondary synchronization signal (SSS) symbols for the physical layer cell identity number `ncellid`. The function implements TS 38.211 Section 7.4.2.3 [1].

`sym = nrSSS(ncellid,'OutputDataType',datatype)` specifies the data type of the SSS symbol.

Examples

Generate SSS Symbols

Generate the sequence of 127 SSS binary phase shift keying (BPSK) modulation symbols for a given cell identity. The SSS is transmitted in the third symbol of a Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block.

```
ncellid = 17;
sss = nrSSS(ncellid)
```

```
sss = 127×1

    -1
     1
    -1
    -1
    -1
     1
    -1
     1
    -1
     1
      ⋮
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

Physical layer cell identity number, specified as an integer from 0 to 1007.

Data Types: `double`

`datatype` — Data type of output symbols
`'double'` (default) | `'single'`

Data type of the output symbols, specified as `'double'` or `'single'`.

Data Types: `char` | `string`

Output Arguments

`sym` — SSS symbols
column vector of real numbers

SSS symbols, returned as a column vector of real numbers.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify `single` data type for the output, include `{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCH` | `nrPBCHDMRS` | `nrPSS` | `nrSSSIndices`

Introduced in R2018b

nrSSSIndices

Generate SSS resource element indices

Syntax

```
ind = nrSSSIndices
ind = nrSSSIndices(Name,Value)
```

Description

`ind = nrSSSIndices` returns the resource element indices for the secondary synchronization signal (SSS), as defined in TS 38.211 Section 7.4.3.1 [1]. The returned indices are one-based using linear indexing form. This indexing form can directly index the elements of a 240-by-4 matrix corresponding to the Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block. The order of the indices indicates how the SSS modulation symbols are mapped.

`ind = nrSSSIndices(Name,Value)` specifies index formatting options by using one or more name-value pair arguments. Unspecified options take default values.

Examples

Get SSS Resource Element Indices

Generate the 127 resource element indices associated with the SSS within a single SS/PBCH block.

```
ind = nrSSSIndices
```

```
ind = 127x1 uint32 column vector

   537
   538
   539
   540
   541
   542
   543
   544
   545
   546
      ⋮
```

Input Arguments

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'IndexStyle','subscript','IndexBase','0based'` specifies nondefault resource element index formatting options.

`IndexStyle` — Resource element indexing form
`'index'` (default) | `'subscript'`

Resource element indexing form, specified as the comma-separated pair consisting of `'IndexStyle'` and one of these values:

- `'index'` — The indices are in linear index form.
- `'subscript'` — The indices are in [subcarrier, symbol, antenna] subscript row form.

Data Types: `char` | `string`

`IndexBase` — Resource element indexing base
`'1based'` (default) | `'0based'`

Resource element indexing base, specified as the comma-separated pair consisting of `'IndexBase'` and one of these values:

- `'1based'` — The index counting starts from one.
- `'0based'` — The index counting starts from zero.

Data Types: `char` | `string`

Output Arguments

`ind` — SSS resource element indices
column vector (default) | *M*-by-3 matrix

SSS resource element indices, returned as one of these values:

- Column vector — When `'IndexStyle'` is `'index'`.
- *M*-by-3 matrix — When `'IndexStyle'` is `'subscript'`. The matrix rows correspond to the [subcarrier, symbol, antenna] subscripts based on the number of subcarriers and OFDM symbols in an SS/PBCH block, and the number of antennas, respectively.

Depending on `'IndexBase'`, the indices are either one-based or zero-based.

Data Types: `uint32`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify linear indexing form, include `{coder.Constant('IndexStyle'),coder.Constant('index')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions
`nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPSSIndices` | `nrSSS`

Introduced in R2018b

nrSymbolDemodulate

Demodulate and convert symbols to bits

Syntax

```
out = nrSymbolDemodulate(in,mod)
out = nrSymbolDemodulate(in,mod,nVar)
out = nrSymbolDemodulate(in,mod,'DecisionType',decision)
```

Description

`out = nrSymbolDemodulate(in,mod)` demodulates complex symbols in codeword `in` to soft bits using modulation scheme `mod`. The function implements the inverse of TS 38.211 Section 5.1 [1].

`out = nrSymbolDemodulate(in,mod,nVar)` specifies the noise variance scaling factor for the soft bits.

`out = nrSymbolDemodulate(in,mod,'DecisionType',decision)` specifies the demodulation decision mode by using a name-value pair argument.

Examples

QPSK Demodulation with Soft Decision Mode

Generate a random sequence of binary values of length 40.

```
data = randi([0 1],40,1);
```

Generate modulated symbols using QPSK modulation.

```
modsymb = nrSymbolModulate(data,'QPSK');
```

Perform QPSK demodulation in soft decision mode for a noise variance of 0.1.

```
nVar = 0.1;
recsymb = awgn(modsymb,1/nVar,1,'linear');
out = nrSymbolDemodulate(recsymb,'QPSK',0.1);
```

16-QAM Demodulation with Hard Decision Mode

Generate a random sequence of binary values of length 100.

```
data = randi([0 1],100,1,'int8');
```

Generate modulated symbols using 16-QAM modulation.

```
modsymb = nrSymbolModulate(data,'16QAM');
```

Add a noise to the modulated symbols corresponding to an SNR of 15 dB.

```
recsymb = awgn(modsymb,15);
```

Perform 16-QAM demodulation in hard decision mode.

```
demodbits = nrSymbolDemodulate(recsymb,'16QAM','DecisionType','Hard');
```

Check for bit errors.

```
numErr = biterr(data,demodbits)
```

```
numErr = 1
```

Input Arguments

`in` — Codeword to demodulate

complex column vector

Codeword to demodulate, specified as a complex column vector.

Data Types: `double` | `single`
Complex Number Support: Yes

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type to be performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'` `'BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nVar` — Noise variance

`1e-10` (default) | nonnegative numeric scalar

Noise variance, specified as a nonnegative numeric scalar. The soft bits are scaled with the variance of additive white Gaussian noise (AWGN). The default value corresponds to an SNR of 100 dB, assuming unit signal power. This argument applies only for soft decision mode.

Note The default value assumes the modulator and demodulator are connected back-to-back where the noise variance is zero. To avoid `+/-Inf` values in the output, the function uses `1e-10` as default value for noise variance. To get appropriate results when the signal is transmitted through a noisy channel, adjust the noise variance accordingly.

Data Types: `double`

`decision` — Decision mode
`'soft'` (default) | `'hard'`

Decision mode, specified as `'soft'` or `'hard'`. The decision mode controls the demodulation type performed on the received symbols.

- `'soft'` — Soft decision mode results in a numeric output containing the bitwise approximation to the log-likelihood ratios of the demodulated bits. The output `out` inherits its data type from the input `in`.
- `'hard'`— Hard decision mode results in a binary output containing groups of bits corresponding to the closest constellation points to the input `in`. The output `out` is type-cast to `int8`.

Data Types: `char` | `string`

Output Arguments

`out` — Demodulated output bits
numeric column vector | binary column vector

Demodulated output bits, returned as a numeric column vector or binary column vector. Demodulation is performed assuming the input constellation power normalization defined in TS 38.211 section 5.1 [1].

Modulation Scheme	Constellation Power Normalization Factor

`'pi/2-BPSK'`
`'BPSK'`
`'QPSK'`
	`1/sqrt(2)`

`'16QAM'`	`1/sqrt(10)`
`'64QAM'`	`1/sqrt(42)`
`'256QAM'`	`1/sqrt(170)`

Each demodulated symbol is mapped to a group of bits corresponding to the number of bits per symbol in the modulation scheme `mod`. The first bit represents the most significant bit, and the last bit represents the least significant bit. The length of `out` is the length of the input `in` multiplied by the number of bits per symbol. The `decision` mode controls the content and the data type of the demodulated output bits.

Data Types: `double` | `single` | `int8`

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify soft decision type, include `{coder.Constant('DecisionType'),coder.Constant('soft')}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions

`nrLayerDemap` | `nrPBCHDecode` | `nrPDCCHDecode` | `nrPDSCHDecode` | `nrPRBS` | `nrSymbolModulate`

Introduced in R2018b

nrSymbolModulate

Generate modulated symbols

Syntax

```
out = nrSymbolModulate(in,mod)
out = nrSymbolModulate(in,mod,'OutputDataType',datatype)
```

Description

`out = nrSymbolModulate(in,mod)` maps the bit sequence in codeword `in` to complex modulation symbols using modulation scheme `mod` and returns modulated symbols. The function implements TS 38.211 Section 5.1 [1].

`out = nrSymbolModulate(in,mod,'OutputDataType',datatype)` specifies the data type of the modulated output symbols by using a name-value pair argument. The function uses the specified data type for intermediate computations.

Examples

Generate 16-QAM Modulated Symbols

Generate a random sequence of binary values of length 40. Generate modulated symbols using 16-QAM modulation.

```
data = randi([0 1],40,1);
sym = nrSymbolModulate(data,'16QAM');
```

Generate QPSK-Modulated Symbols

Generate a random sequence of binary values of length 20. Generate modulated symbols using QPSK modulation and specify single-precision data type for the output.

```
data = randi([0 1],20,1,'int8');
sym = nrSymbolModulate(data,'QPSK','OutputDataType','single');
```

Input Arguments

`in` — Codeword to modulate

column vector of binary values

Codeword to modulate, specified as a column vector of binary values. The codeword length must be a multiple of the number of bits per symbol, specified by the modulation scheme `mod`.

Data Types: `double` | `int8` | `logical`

mod — Modulation scheme

'pi/2-BPSK' | 'BPSK' | 'QPSK' | '16QAM' | '64QAM' | '256QAM'

Modulation scheme, specified as 'pi/2-BPSK', 'BPSK', 'QPSK', '16QAM', '64QAM', or '256QAM'. This modulation scheme determines the modulation type to be performed on the input codeword and the number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK' 'BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string

datatype — Data type of modulated output symbols

'double' (default) | 'single'

Data type of modulated output symbols, specified as 'double' or 'single'. The input argument datatype determines the data type of the modulated output symbols and the data type that the function uses for intermediate computations.

Data Types: char | string

Output Arguments

out — Modulated output symbols

complex column vector

Modulated output symbols, returned as a complex column vector. The length of out is the length of the codeword in divided by the number of bits per symbol, specified by the modulation scheme mod.

Data Types: double | single
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify single data type for the output, include

`{coder.Constant('OutputDataType'),coder.Constant('single')}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

See Also

Functions

`nrLayerMap` | `nrPBCH` | `nrPDCCH` | `nrPDSCH` | `nrPRBS` | `nrSymbolDemodulate`

Introduced in R2018b

nrTBS

Get transport block size

Syntax

```
tbs = nrTBS(mod,nlayers,nPRB,NREPerPRB,tcr)
tbs = nrTBS(___,xOh)
tbs = nrTBS(___,tbScaling)
```

Description

`tbs = nrTBS(mod,nlayers,nPRB,NREPerPRB,tcr)` returns `tbs`, the transport block size (TBS), associated with each codeword for a shared channel transmission, as defined in TS 38.214 Sections 5.1.3.2 and 6.1.4.2. `modulation` is the modulation scheme for each codeword and `nlayers` is the number of transmission layers. `nPRB` is the number of physical resource blocks (PRBs) allocated for the physical shared channel. `NREPerPRB` is the number of resource elements (REs) allocated for the data transmission in the shared channel within one PRB for one slot (without accounting for the additional overhead). `tcr` is the target code rate for each codeword. The additional overhead and scaling factor used for TBS calculation are 0 and 1, respectively.

`tbs = nrTBS(___,xOh)` specifies the additional overhead in addition to the input arguments of the previous syntax. The additional overhead accounts for the overhead from channel state information reference signal (CSI-RS) and control resource set (CORESET). `xOh` controls the number of REs available for the data transmission in the shared channel within one PRB for one slot. The scaling factor used for TBS calculation is 1.

`tbs = nrTBS(___,tbScaling)` specifies the scaling factor in addition to the input arguments of the previous syntax. The function uses `tbScaling` to calculate the intermediate number of information bits, *N_info*, as defined in TS 38.214 Section 5.1.3.2.

Examples

Get TBS for One Codeword

Specify the modulation scheme for one codeword as 16-QAM, the number of transmission layers as 4, and the number of PRBs allocated for the shared channel as 52. Specify the number of REs allocated for the shared channel within one PRB for one slot (without accounting for the additional overhead) as 120. Set the target code rate to 0.48.

```
modulation = '16QAM';
nlayers = 4;
nPRB = 52;
NREPerPRB = 120;
tcr = 0.48;
```

Get the TBS associated with a data transmission having the additional overhead of 6 and scaling factor of 0.25.

```
xOh = 6;
tbScaling = 0.25;
tbs = nrTBS(modulation,nlayers,nPRB,NREPerPRB,tcr,xOh,tbScaling)
```

```
tbs = 11272
```

Get TBS for Two Codewords

Specify the modulation schemes for two codewords as QPSK and 64-QAM. Set the number of transmission layers to 8, and the number of PRBs allocated for the shared channel as 106. Specify the number of REs allocated for the shared channel within one PRB for one slot (without accounting for the additional overhead) as 100.

```
modulation = {'QPSK','64QAM'};
nlayers = 8;
nPRB = 106;
NREPerPRB = 100;
```

Specify the target code rates for two codewords as 0.3701 and 0.4277. Get the payload size of each transport block for a shared channel transmission.

```
tcr = [0.3701 0.4277];
tbs = nrTBS(modulation,nlayers,nPRB,NREPerPRB,tcr)
```

```
tbs = 1×2

       31240      108552
```

Input Arguments

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar | string array | cell array of character vectors

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, a string scalar, a string array, or a cell array of character vectors. The modulation scheme for a single codeword is specified as a character vector or a string scalar. If two codewords are present, a single modulation scheme can be applied to both codewords. Alternatively, you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string` | `cell`

`nlayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For one codeword, use an integer from 1 to 4. For two codewords, use an integer from 5 to 8.

Data Types: `double`

`nPRB` — Number of PRBs
nonnegative integer

Number of PRBs allocated for the physical shared channel, specified as a nonnegative integer. The nominal value of this argument is in the range of 0 to 275.

Data Types: `double`

`NREPerPRB` — Number of REs
nonnegative integer

Number of REs allocated for the data transmission in the physical shared channel within one PRB for one slot, specified as a nonnegative integer. This value excludes any additional overhead.

Data Types: `double`

`tcr` — Target code rate
scalar between 0 and 1 | two-element vector of values between 0 and 1

Target code rate for each codeword, specified as a scalar between 0 and 1 or a two-element vector of values between 0 and 1. Configure two codewords with different target code rates by specifying a two-element vector. Configure two codewords with the same target code rate by specifying a scalar.

Data Types: `double`

`xOh` — Additional overhead
nonnegative integer

Additional overhead, specified as a nonnegative integer. The additional overhead controls the number of REs available for the data transmission in the physical shared channel within one PRB for one slot. The additional overhead accounts for the overhead from channel state information reference signal (CSI-RS) and control resource set (CORESET). The nominal value of the additional overhead is `0`, `6`, `12`, or `18`, provided by the higher-layer parameter *xOverhead* in PDSCH-ServingCellConfig IE or PUSCH-ServingCellConfig IE.

Data Types: `double`

`tbScaling` — Scaling factor
scalar in the range (0, 1] | two-element vector of values in the range (0, 1]

Scaling factor, specified as a scalar in the range (0, 1] or a two-element vector of values in the range (0, 1]. The function uses this value in calculating the intermediate number of information bits, *N_info*, as defined in TS 38.214 Section 5.1.3.2. Configure two codewords with different scaling factors by specifying a two-element vector. Configure two codewords with the same scaling factor by specifying a scalar.

The nominal value of the scaling factor is `0.25`, `0.5`, or `1`, as defined in TS 38.214 Table 5.1.3.2-2.

Data Types: `double`

Output Arguments

`tbs` — Transport block size

nonnegative integer | two-element vector of nonnegative integers

Transport block size associated with each codeword in the shared channel transmission, returned as a nonnegative integer or a two-element vector of nonnegative integers.

The value of `tbs` is `0` in any of these cases.

- When the `NREPerPRB` input is `0`
- When the `nPRB` input is `0`
- When the `NREPerPRB` input is less than the `xOh` input

Data Types: `double`

References

[1] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`nrDLSCH` | `nrULSCH`

Introduced in R2020b

nrTimingEstimate

Practical timing estimation

Syntax

```
[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialSlot,refInd,refSym)
[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialSlot,refGrid)

[offset,mag] = nrTimingEstimate(carrier,waveform,refInd,refSym)
[offset,mag] = nrTimingEstimate(carrier,waveform,refGrid)

[offset,mag] = nrTimingEstimate( ___ ,'CyclicPrefix',cpl)
[offset,mag] = nrTimingEstimate( ___ ,Name,Value)
```

Description

`[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialSlot,refInd,refSym)` performs practical timing estimation by cross-correlating the input waveform `waveform` with a reference waveform. The function obtains the reference waveform by modulating a reference resource grid containing reference symbols `refSym` at locations `refInd` using orthogonal frequency division multiplexing (OFDM). The OFDM modulation spans `nrb` resource blocks at subcarrier spacing `scs` and initial slot number `initialSlot`. The function returns the estimated timing offset `offset` and the estimated impulse response magnitude `mag` for each receive antenna in the input waveform.

`[offset,mag] = nrTimingEstimate(waveform,nrb,scs,initialSlot,refGrid)` specifies a predefined reference resource grid `refGrid`.

`[offset,mag] = nrTimingEstimate(carrier,waveform,refInd,refSym)` specifies carrier configuration parameters.

`[offset,mag] = nrTimingEstimate(carrier,waveform,refGrid)` specifies a carrier resource grid.

`[offset,mag] = nrTimingEstimate( ___ ,'CyclicPrefix',cpl)` specifies the cyclic prefix length for the OFDM modulation in addition to the input arguments in any of the previous syntaxes that do not include the `carrier` input.

`[offset,mag] = nrTimingEstimate( ___ ,Name,Value)` specifies options by using one or more name-value pair arguments in addition to any combination of input arguments from the previous syntaxes.

Examples

Estimate Timing Offset for TDL-C Channel Transmission

Generate primary synchronization signal (PSS) symbols for physical layer cell identity number 42.

```
ncellid = 42;
pssSym = nrPSS(ncellid);
```

Obtain resource element indices for the PSS.

```
pssInd = nrPSSIndices();
```

Create a resource grid containing the generated PSS symbols.

```
nrb = 20;
scs = 15;
carrier = nrCarrierConfig('NSizeGrid',nrb,'SubcarrierSpacing',scs);
txGrid = nrResourceGrid(carrier);
txGrid(pssInd) = pssSym;
```

OFDM modulate the resource grid.

```
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Transmit the waveform through a TDL-C channel model by using a sample rate of 7.68 MHz.

```
ofdmInfo = nrOFDMInfo(carrier);
channel = nrTDLChannel;
channel.SampleRate = ofdmInfo.SampleRate;
channel.DelayProfile = 'TDL-C';
rxWaveform = channel(txWaveform);
```

Estimate timing offset for the transmission by using the PSS symbols as reference symbols. The OFDM modulation of the reference symbols uses initial slot number 0.

```
initialSlot = 0;
offset = nrTimingEstimate(rxWaveform,nrb,scs,initialSlot,pssInd,pssSym);
```

Input Arguments

`waveform` — Received waveform

T-by-N_R complex matrix

Received waveform, specified as a *T*-by-N_R complex matrix.

- *T* is the number of time-domain samples.
- N_R is the number of receive antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

`nrb` — Number of resource blocks

integer from 1 to 275

Number of resource blocks, specified as an integer from 1 to 275.

Data Types: `double`

`scs` — Subcarrier spacing in kHz

`15` | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`initialSlot` — Zero-based initial slot number
nonnegative integer

Zero-based initial slot number, specified as a nonnegative integer. The function selects the appropriate cyclic prefix length for the OFDM modulation based on the value of `initialSlot` modulo the number of slots per subframe.

Data Types: `double`

`refInd` — Reference symbol indices
integer matrix

Reference symbol indices, specified as an integer matrix. The number of rows equals the number of resource elements. You can specify all indices in a single column or distribute them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

The elements of `refInd` are one-based linear indices addressing a *K*-by-*L*-by-*P* resource array.

- *K* is the number of subcarriers equal to `nrb` × 12.
- *L* is the number of OFDM symbols in a slot. *L* is 12 or 14, depending on the cyclic prefix length specified in the `cpl` input or the `CyclicPrefix` property of the `carrier` input.
- *P* is the number of reference signal ports, inferred from the range of values in `refInd`.

Data Types: `double`

`carrier` — Carrier configuration parameters
`nrCarrierConfig` object

Carrier configuration parameters for a specific OFDM numerology, specified as an `nrCarrierConfig` object. The function uses only these properties of this input.

`NSizeGrid` — Number of RBs in carrier resource grid
`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSlot` — Slot number
`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB

simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`refSym` — Reference symbols
complex matrix

Reference symbols, specified as a complex matrix. The number of rows equals the number of resource elements. You can specify all symbols in a single column or distribute them across several columns. The number of elements in `refInd` and `refSym` must be the same but their dimensionality can differ. The function reshapes `refInd` and `refSym` into column vectors before mapping them into a reference grid: `refGrid(refInd(:)) = refSym(:)`.

Data Types: `single` | `double`
Complex Number Support: Yes

`refGrid` — Predefined reference grid
K-by-*N*-by-*P* complex array

Predefined reference grid, specified as a *K*-by-*N*-by-*P* complex array. `refGrid` can span multiple slots.

- *K* is the number of subcarriers equal to `nrb` × 12.
- *N* is the number of OFDM symbols in the reference grid.
- *P* is the number of reference signal ports.

Data Types: `single` | `double`
Complex Number Support: Yes

`cpl` — Cyclic prefix length
`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options:

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, the extended cyclic prefix length only applies to 60 kHz subcarrier spacing.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'SampleRate','1e9'` specifies a sample rate of 1×10^9 Hz.

`Nfft` — Number of FFT points

integer greater than 127 (default depends on other input values) | `[]`

Number of fast Fourier transform (FFT) points, specified as the comma-separated pair consisting of `'Nfft'` and either a nonnegative integer greater than 127, or `[]`. The value you specify must result in integer-valued cyclic prefix lengths and a maximum occupancy, defined as the value of $(12 \times N_{\mathrm{RB}})/$ `Nfft`, where N_{RB} is the number of resource blocks, of 100%.

If you do not specify this input, or if you specify `'Nfft',[]`, the function sets a default value satisfying these conditions.

- The value of this input is an integer power of 2.
- The maximum occupancy is 85%.
- The minimum value of this input is `128`.

Data Types: `double`

`SampleRate` — Waveform sample rate

positive scalar (default depends on other input values) | `[]`

Waveform sample rate, specified as the comma-separated pair consisting of `'SampleRate'` and either a positive scalar or `[]`.

If you do not specify this input, or if you specify `'SampleRate',[]`, then the function sets this input to the value of $N_{\mathrm{fft}} \times SCS$.

- N_{fft} is the value of the `'Nfft'` input.
- SCS is the subcarrier spacing specified in the `SubcarrierSpacing` property of the `config` input for the first function syntax, or the `scs` input for the other syntaxes.

Data Types: `double`

`CarrierFrequency` — Carrier frequency

`0` (default) | nonnegative scalar

Carrier frequency, in Hz, specified as the comma-separated pair consisting of `'CarrierFrequency'` and a nonnegative scalar. This input corresponds to f_0, defined in Section 5.4 of [1].

Data Types: `double`

Output Arguments

`offset` — Estimated timing offset in samples

nonnegative integer

Estimated timing offset in samples, returned as a nonnegative integer. The number of samples is relative to the first sample of the input waveform `waveform`.

Data Types: `double`

`mag` — Estimated impulse response magnitude
T-by-N_R real matrix

Estimated impulse response magnitude, for each receive antenna in the input waveform `waveform`, returned as a *T*-by-N_R real matrix.

- *T* is the number of time-domain samples.
- N_R is the number of receive antennas.

`mag` inherits the data type of the input `waveform`.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

The `nrb` and `scs` inputs must be constant when you specify any name-value pair arguments. For example, to specify a subcarrier spacing of 30 kHz, include `{coder.Constant(30)}` in the `-args` value of the `codegen` function. For more information, see the `coder.Constant` class.

The `'SampleRate'` name-value-pair argument cannot be used with function syntaxes that use the `carrier` input.

See Also

Functions
`nrChannelEstimate` | `nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

Objects
`nrCarrierConfig`

Introduced in R2019b

nrTransformDeprecode

Recover transform deprecoded symbols

Syntax

```
tdpSym = nrTransformDeprecode(modSym,mrb)
```

Description

`tdpSym = nrTransformDeprecode(modSym,mrb)` recovers transform deprecoded symbols for modulation symbols `modSym`, corresponding to the inverse operation of transform precoding from TS 38.211 Section 6.3.1.4 and 6.3.2.6.4 [1]. `mrb` is the number of resource blocks allocated for the physical uplink shared channel (PUSCH), physical uplink control channel (PUCCH) format 3, or PUCCH format 4 transmission. `mrb` determines the length of the subblocks in `modSym` which are transform deprecoded separately.

In the NR uplink, transform deprecoding is used together with CP-OFDM demodulation to demodulate an SC-FDMA (DFT-s-OFDM) waveform. Transform deprecoding applies to only these transmissions:

- After MIMO deprecoding in the PUSCH with single-layer transmission.
- Before symbol demodulation in the PUCCH format 3 transmission.
- Before block-wise despreading in the PUCCH format 4 transmission.

Examples

Recover Transform Deprecoded PUSCH Symbols

Generate a random sequence of binary values corresponding to a PUSCH codeword of 960 bits.

```
cw = randi([0 1],960,1);
```

Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti);
```

Modulate the scrambled PUSCH codeword by using modulation scheme 16-QAM.

```
modulation = '16QAM';
modSym = nrSymbolModulate(scrambled,modulation);
```

Perform layer mapping using a single transmission layer.

```
layeredSym = nrLayerMap(modSym,1);
```

Generate transform precoded symbols by using an allocated PUSCH bandwidth of 2 resource blocks.

```
tpSym = nrTransformPrecode(layeredSym,2);
```

Recover the corresponding transform deprecoded symbols.

```
tdpSym = nrTransformDeprecode(tpSym,2);
```

Input Arguments

`modSym` — Modulation symbols

complex matrix

Modulation symbols, specified as a complex matrix. The number of rows in `modSym` must be a multiple of `mrb`×12. Typically, `modSym` is specified as a column vector, corresponding to single-layer transmission. If `modSym` is a matrix, the `nrTransformDeprecode` function processes each column separately and returns a matrix.

Data Types: `double`

`mrb` — Number of resource blocks

positive integer

Number of resource blocks allocated for the PUSCH, PUCCH format 3, or PUCCH format 4 transmission, specified as a positive integer. `mrb` determines the length of the subblocks in `modSym` which are transform deprecoded separately. Preferred `mrb` values are of the form $2^{\alpha_2} \times 3^{\alpha_3} \times 5^{\alpha_5}$, where α_2, α_3, and α_5 are nonnegative integers, as specified in the standard.

Data Types: `double`

Output Arguments

`tdpSym` — Transform deprecoded symbols

complex matrix

Transform deprecoded symbols, returned as a complex matrix. `tdpSym` inherits the dimensionality of the input `modSym`.

Data Types: `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrTransformPrecode`

Introduced in R2019a

nrTransformPrecode

Generate transform precoded symbols

Syntax

```
tpSym = nrTransformPrecode(modSym,mrb)
```

Description

`tpSym = nrTransformPrecode(modSym,mrb)` generates transform precoded symbols for modulation symbols `modSym`, as defined in TS 38.211 Section 6.3.1.4 and 6.3.2.6.4 [1]. `mrb` is the number of resource blocks allocated for the physical uplink shared channel (PUSCH), physical uplink control channel (PUCCH) format 3, or PUCCH format 4 transmission. `mrb` determines the length of the subblocks in `modSym` which are transform precoded separately.

In the NR uplink, transform precoding is used together with CP-OFDM modulation to create an SC-FDMA (DFT-s-OFDM) waveform. Transform precoding applies to only these transmissions:

- Before MIMO precoding in the PUSCH with single-layer transmission.
- After symbol modulation in the PUCCH format 3 transmission.
- After block-wise spreading in the PUCCH format 4 transmission.

Examples

Generate Transform Precoded PUSCH Symbols

Generate a random sequence of binary values corresponding to a PUSCH codeword of 960 bits.

```
cw = randi([0 1],960,1);
```

Perform PUSCH scrambling initialized with the specified physical layer cell identity number and RNTI.

```
ncellid = 42;
rnti = 101;
scrambled = nrPUSCHScramble(cw,ncellid,rnti);
```

Modulate the scrambled PUSCH codeword by using modulation scheme 16-QAM.

```
modulation = '16QAM';
modSym = nrSymbolModulate(scrambled,modulation);
```

Perform layer mapping using a single transmission layer.

```
layeredSym = nrLayerMap(modSym,1);
```

Generate transform precoded symbols by using an allocated PUSCH bandwidth of 2 resource blocks.

```
tpSym = nrTransformPrecode(layeredSym,2);
```

Input Arguments

`modSym` — Modulation symbols

complex matrix

Modulation symbols, specified as a complex matrix. The number of rows in `modSym` must be a multiple of `mrb`×12. Typically, `modSym` is specified as a column vector, corresponding to single-layer transmission. If `modSym` is a matrix, the `nrTransformPrecode` function processes each column separately and returns a matrix.

Data Types: `double`

`mrb` — Number of resource blocks

positive integer

Number of resource blocks allocated for the PUSCH, PUCCH format 3, or PUCCH format 4 transmission, specified as a positive integer. `mrb` determines the length of the subblocks in `modSym` which are transform precoded separately. Preferred `mrb` values are of the form $2^{\alpha_2} \times 3^{\alpha_3} \times 5^{\alpha_5}$, where α_2, α_3, and α_5 are nonnegative integers, as specified in the standard.

Data Types: `double`

Output Arguments

`tpSym` — Transform precoded symbols

complex matrix

Transform precoded symbols, returned as a complex matrix. `tpSym` inherits the dimensionality of the input `modSym`.

Data Types: `double`
Complex Number Support: Yes

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrTransformDeprecode`

Introduced in R2019a

nrUCIDecode

Decode uplink control information (UCI)

Syntax

```
ucibits = nrUCIDecode(softbits,A)
ucibits = nrUCIDecode(softbits,A,mod)
ucibits = nrUCIDecode( ___ ,'ListLength',L)
[ucibits,err] = nrUCIDecode( ___ )
```

Description

`ucibits = nrUCIDecode(softbits,A)` decodes the input `softbits` and returns the decoded UCI bits of length `A`. The function implements the inverse of the encoding process specified in TS 38.212 Sections 6.3.1.2–6.3.1.5 for the physical uplink control channel (PUCCH) and in Sections 6.3.2.2–6.3.2.5 for the physical uplink shared channel (PUSCH) [1]. The decoding consists of rate recovery, channel decoding, and cyclic redundancy check (CRC) decoding per code block. The particular decoding scheme that the function implements depends on the decoded UCI message length, `A`. For more details, see “Algorithms” on page 1-406.

`ucibits = nrUCIDecode(softbits,A,mod)` also specifies the modulation scheme for the decoding. The specified modulation scheme applies only when the length of `ucibits` is 1 or 2. When not specified, the modulation scheme defaults to QPSK.

`ucibits = nrUCIDecode( ___ ,'ListLength',L)` specifies the list length for polar decoding in addition to the input arguments in any of the previous syntaxes. The specified list length applies only for the successive cancellation list (SCL) decoding when $A \geq 12$. When not specified, the list length defaults to 8.

`[ucibits,err] = nrUCIDecode( ___ )` also returns an error flag. Use the input arguments in any of the previous syntaxes. A value of `1` in `err` indicates that an error occurred during code block decoding. The `err` output applies only for CRC-based decoding schemes. For more information, see “Algorithms” on page 1-406.

Examples

Decode UCI Codeword

Create a random sequence of binary values corresponding to a UCI message of 32 bits. Encode the message based on the specified length of the rate-matched UCI codeword.

```
A = 32;
E = 120;
uciBits = randi([0 1],A,1);
ucicw   = nrUCIEncode(uciBits,E);
```

Decode the soft bits representing UCI codeword `ucicw`. Set the length of the polar decoding list to 4. The error flag in the output indicates that no errors occurred during code block decoding.

```
L = 4;
[recBits,err] = nrUCIDecode(1-2*ucicw,A,'ListLength',L)
```

```
recBits = 32x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮


err = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(recBits,uciBits)
```

```
ans = logical
   1
```

Decode UCI Codeword with 16-QAM Modulation Scheme and AWGN

Create a random sequence of binary values corresponding to a two-bit UCI message.

```
K = 2;
uci = randi([0 1],K,1,'int8');
```

Encode the message for the specified length of the rate-matched output and 16-QAM modulation scheme.

```
mod = '16QAM';
E = 4*3;
encUCI = nrUCIEncode(uci,E,mod);
```

Replace placeholders -1 and -2 in the output through scrambling.

```
encUCI(encUCI==-1) = 1;
encUCI(encUCI==-2) = encUCI(find(encUCI==-2)-1);
```

Modulate the encoded UCI message.

```
modOut = nrSymbolModulate(encUCI,mod);
```

Add white Gaussian noise (AWGN) to the modulated symbols using a signal-to-noise ratio of 0 dB.

```
snrdB = 0;
rxSig = awgn(modOut,snrdB);
```

Demodulate the received signal.

```
rxSoftBits = nrSymbolDemodulate(rxSig,mod);
```

Decode the soft bits representing the demodulated UCI codeword.

```
decBits = nrUCIDecode(rxSoftBits,K,mod);
```

Verify that the transmitted and received message bits are identical.

```
isequal(decBits,uci)
```

```
ans = logical
   1
```

Input Arguments

`softbits` — Approximate LLR soft bits

real column vector

Approximate log-likelihood ratio (LLR) soft bits corresponding to encoded UCI bits, specified as a real column vector.

Data Types: `double` | `single`

`A` — Length of decoded UCI message bits

integer from 1 to 1706

Length of decoded UCI message bits, specified as an integer from 1 to 1706.

Data Types: `double`

`mod` — Modulation scheme

`'QPSK'` (default) | `'pi/2-BPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol, as shown in this table.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

This input argument applies only when the input `A` is `1` or `2`.

Data Types: `char` | `string`

`L` — Length of polar decoding list

8 (default) | power of two

Length of polar decoding list, specified as `8` or a power of two.

Data Types: `double`

Output Arguments

`ucibits` — Decoded UCI message bits

A-by-1 column vector of binary values

Decoded UCI message bits, returned as an `A`-by-1 column vector of binary values.

Data Types: `int8`

`err` — Result of UCI code block decoding

logical scalar | logical vector

Result of UCI code block decoding for each code block, returned as a logical scalar or logical vector of length 2. `1` in `err` indicates that an error has occurred during code block decoding.

Data Types: `logical`

Algorithms

The particular UCI decoding scheme that `nrUCIDecode` implements depends on the specified output length `A`.

A	Deconcatenation	Decoding	CRC Bits
1–11	N/A	Maximum likelihood	N/A
12–19	N/A	CRC-aided SCL	6
20–1706	Depends on `A` and the length of `softbits`	CRC-aided SCL	11

Compatibility Considerations

Polar decoding metric update

Behavior changed in R2020a

In releases R2019b and before, polar decoding uses the exact form of the expression $\log(1 + e^x)$ for internal metric evaluation. Starting in release R2020a, because the exact form leads to numerical instability for high SNR ranges, polar decoding approximates $\log(1 + e^x)$ as 0 for $x < 0$ and as x for $x \geq 0$. This approximation affects the results of the `nrUCIDecode` function, resulting in a marginal degradation of the BLER performance in a link-level simulation.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Names and values in name-value pair arguments must be compile-time constants. For example, to specify the list length, include `{coder.Constant('ListLength'),coder.Constant(L)}` in the `-args` value of `codegen`. For more information, see `coder.Constant`.

See Also

Functions

`nrCRCDecode` | `nrPolarDecode` | `nrRateRecoverPolar` | `nrUCIEncode`

Introduced in R2019a

nrUCIEncode

Encode uplink control information (UCI)

Syntax

```
codeduci = nrUCIEncode(ucibits,E)
codeduci = nrUCIEncode(ucibits,E,mod)
```

Description

`codeduci = nrUCIEncode(ucibits,E)` encodes UCI message bits `ucibits` and returns concatenated, rate-matched, and encoded UCI blocks of length `E`. The returned blocks can be mapped to either the physical uplink control channel (PUCCH) or the physical uplink shared channel (PUSCH). The function implements TS 38.212 Sections 6.3.1.2–6.3.1.5 for PUCCH and Sections 6.3.2.2–6.3.2.5 for PUSCH [1]. The encoding consists of code block segmentation, cyclic redundancy check (CRC) attachment, channel coding, rate matching, and code block concatenation. The function supports polar encoding and small block lengths. The particular encoding scheme that the function implements depends on the input UCI message length. For more details, see “Algorithms” on page 1-410.

`codeduci = nrUCIEncode(ucibits,E,mod)` also specifies the modulation scheme for the encoding. The specified modulation scheme applies only when the length of `ucibits` is 1 or 2. When not specified, the modulation scheme defaults to QPSK. In the output, –1 and –2 represent the *x* and *y* placeholders, respectively, in Tables 5.3.3.1-1 and 5.3.3.2-1.

Examples

Encode UCI Message Bits

Create a random sequence of binary values corresponding to a UCI message of 32 bits.

```
ucibits = randi([0 1],32,1,'int8');
```

Encode the message for the specified rate-matched output length.

```
E = 120;
codeduci = nrUCIEncode(ucibits,E)
```

```
codeduci = 120x1 int8 column vector

   1
   1
   1
   0
   1
   0
   1
   0
   0
   0
```

```
    ⋮
```

Encode UCI Message Bits for 16-QAM Modulation Scheme

Create a random sequence of binary values corresponding to a two-bit UCI message.

```
ucibits = randi([0 1],2,1,'int8');
```

Encode the message for the specified rate-matched output length and 16-QAM modulation scheme.

```
E = 12;
codeduci = nrUCIEncode(ucibits,E,'16QAM')
```

```
codeduci = 12x1 int8 column vector

    1
    1
   -1
   -1
    0
    1
   -1
   -1
    1
    0
    ⋮
```

Input Arguments

`ucibits` — UCI message bits

column vector of binary values

UCI message bits, specified as a column vector of binary values. `ucibits` is the information bits encoded before transmission on the PUCCH or PUSCH.

Data Types: `double` | `int8`

E — Length of rate-matched UCI codeword

positive integer

Length of rate-matched UCI codeword, in bits, specified as a positive integer. The valid range of E depends on A, the length of the input `ucibits`, as shown in this table.

A	Valid Range of E
1	$E \geq A$
2–11	$E > A$
12–19	$E > A + 9$
20–1706	$E > A + 11$

Data Types: `double`

mod — Modulation scheme
'QPSK' (default) | 'pi/2-BPSK' | '16QAM' | '64QAM' | '256QAM'

Modulation scheme, specified as 'QPSK', 'pi/2-BPSK', '16QAM', '64QAM', or '256QAM'. This modulation scheme determines the modulation type and number of bits used per modulation symbol, as shown in this table.

Modulation Scheme	Number of Bits Per Symbol
'pi/2-BPSK'	1
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

This input argument applies only when the input ucibits is one or two bits.

Data Types: char | string

Output Arguments

codeduci — Coded UCI bits
E-by-1 column vector of integers from –2 to 1

Coded UCI bits, returned as an E-by-1 column vector of integers from –2 to 1. codeduci inherits its data type from the input ucibits. Element values –1 and –2 represent the *x* and *y* placeholders, respectively, in Tables 5.3.3.1-1 and 5.3.3.2-1.

Data Types: double | int8

Algorithms

The UCI encoding consists of code block segmentation, cyclic redundancy check (CRC) attachment, channel coding, rate matching, and code block concatenation. The particular UCI encoding scheme that nrUCIEncode implements depends on *A*, the length of input ucibits.

A	Code Block Segmentation	CRC Bits	Encoding
1	N/A	N/A	Repetition
2	N/A	N/A	Simplex
3–11	N/A	N/A	Reed-Muller
12–19	N/A	6	Parity-check Polar
20–1706	Occurs only when $A \geq 1013$ or when $A \geq 360$ and $E \geq 1088$	11	Polar

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPUCCH2` | `nrPUCCH3` | `nrPUCCH4` | `nrPUSCH` | `nrPolarEncode` | `nrRateMatchPolar` | `nrUCIDecode`

Introduced in R2019a

nrULSCHDemultiplex

Perform UL-SCH data and control demultiplexing

Syntax

```
[culsch,cack,ccsi1,ccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1,
ocsi2,cw)
```

Description

`[culsch,cack,ccsi1,ccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1, ocsi2,cw)` returns demultiplexed encoded data vector `culsch`, and encoded uplink control information (UCI) vectors `cack`, `ccsi1`, and `ccsi2` post the demultiplexing of received codeword `cw`. This demultiplexing reverses the processing defined in TS 38.212 Section 6.2.7 [1]. `pusch` is the physical uplink shared channel (PUSCH) configuration. `tcr` is the target code rate. `tbs` is the transport block size for the uplink shared channel (UL-SCH) transmission. `oack` is number of the hybrid automatic repeat request acknowledgment (HARQ-ACK) payload bits. `ocsi1` is the number of channel state information (CSI) part 1 payload bits, and `ocsi2` is the number of CSI part 2 payload bits. `cw` is a column vector of the received log-likelihood ratio (LLR) soft bits.

Examples

Perform UL-SCH Multiplexing and Demultiplexing

Create a PUSCH configuration object with a pi/2-BPSK modulation scheme and no frequency hopping. Set the beta offset factor for the HARQ-ACK to 20, and the beta offset factor for CSI part 1 and CSI part 2 to 6.25 each. Specify the scaling factor as 1, which limits the number of resource elements (REs) assigned for the UCI.

```
pusch = nrPUSCHConfig;
pusch.Modulation = 'pi/2-BPSK';
pusch.FrequencyHopping = 'neither';
pusch.BetaOffsetACK = 20;
pusch.BetaOffsetCSI1 = 6.25;
pusch.BetaOffsetCSI2 = 6.25;
pusch.UCIScaling = 1;
```

Set the target code rate, payload lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.5;   % Target code rate
tbs = 3848;  % Payload length of UL-SCH data (transport block size)
oack = 8;    % Payload length of HARQ-ACK
ocsi1 = 88;  % Payload length of CSI part 1
ocsi2 = 100; % Payload length of CSI part 2
```

Get the rate matched lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the randomly coded bits of the UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2.

```
culsch = randi([0 1],rmInfo.GULSCH,1);
cack = randi([0 1],rmInfo.GACK,1);
ccsi1 = randi([0 1],rmInfo.GCSI1,1);
ccsi2 = randi([0 1],rmInfo.GCSI2,1);
```

Get the codeword from the randomly coded bits of the UL-SCH and coded bits of UCI types.

```
cw = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);
```

Get the demultiplexed UL-SCH and UCI bits from the codeword.

```
[rxculsch,rxcack,rxccsi1,rxccsi2] = nrULSCHDemultiplex(pusch,tcr,tbs,oack,ocsi1,ocsi2,1-2*cw);
```

Verify that the randomly coded bits and demultiplexed coded bits of the UL-SCH and the coded UCI types are identical.

```
isequal(rxculsch<0,culsch)
```

```
ans = logical
   1
```

```
isequal(rxcack<0,cack)
```

```
ans = logical
   1
```

```
isequal(rxccsi1<0,ccsi1)
```

```
ans = logical
   1
```

```
isequal(rxccsi2<0,ccsi2)
```

```
ans = logical
   1
```

Input Arguments

`pusch` — PUSCH configuration parameters

`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

`Modulation` — Modulation scheme

`'QPSK'` (default) | `'pi/2-BPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, or a character array.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1

Modulation Scheme	Number of Bits Per Symbol
'QPSK'	2
'16QAM'	4
'64QAM'	6
'256QAM'	8

Data Types: char | string

NumLayers — Number of transmission layers
1 (default) | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4.

Data Types: double

MappingType — Mapping type
'A' (default) | 'B'

Mapping type of the physical shared channel, specified as 'A' or 'B'.

Data Types: char | string

SymbolAllocation — OFDM symbol allocation
[0 14] (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When this property is set to [] or second element in this two-element vector of nonnegative integers is 0, the symbol allocation is empty.

Data Types: double

PRBSet — PRB allocation
[0:51] (default) | vector of nonnegative integers from 0 to 274

Physical resource block (PRB) allocation of PUSCH within the BWP, specified as a vector of nonnegative integers from 0 to 274.

Data Types: double

TransformPrecoding — Transform precoding flag
0 (default) | 1

Transform precoding flag, specified as one of these values.

- 0 — The transform precoding is disabled and waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).
- 1 — The transform precoding is enabled and waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

Data Types: double | logical

`FrequencyHopping` — Frequency hopping
`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping for the physical uplink shared channel, specified as one of these options.

- `'neither'`
- `'intraSlot'`
- `'interSlot'`

If you specify intra-slot frequency hopping and the input coded bits doesn't satisfy the equations of TS 38.212 Section 6.2.7, the function returns the codeword assuming no frequency hopping is present.

Data Types: `char` | `string`

`BetaOffsetACK` — Beta offset factor of HARQ-ACK
`20` (default) | positive integer

Beta offset factor of the hybrid automatic repeat request acknowledgment (HARQ-ACK), specified as a positive integer. This property is used to determine the number of resources for multiplexing HARQ-ACK. The nominal value is one of the entry from the Table 9.3-1 of TS 38.213.

Data Types: `double`

`BetaOffsetCSI1` — Beta offset factor of CSI part 1
`6.25` (default) | positive integer

Beta offset factor of the channel state information (CSI) part 1, specified as a positive integer. This property is used to determine the number of resources for multiplexing CSI part 1. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

`BetaOffsetCSI2` — Beta offset factor of CSI part 2
`6.25` (default) | positive integer

Beta offset factor of the CSI part 2, specified as a positive integer. This property is used to determine the number of resources for multiplexing CSI part 2. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

`UCIScaling` — Scaling factor
`1` (default) | scalar in the range (0, 1)

Scaling factor to limit the number of the resource elements allocated for the uplink channel information (UCI) on the PUSCH, specified as a scalar in the range (0, 1). The nominal value is 0.5, 0.65, 0.8, or 1.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

DMRS — PUSCH DM-RS configuration parameters
`nrPUSCHDMRSConfig` object with default properties (default) | `nrPUSCHDMRSConfig` object

PUSCH demodulation reference signal (DM-RS) configuration parameters, specified as a `nrPUSCHDMRSConfig` configuration object. This property relates to the demodulation reference signal configuration and contains all properties of the specified `nrPUSCHDMRSConfig` object.

EnablePTRS — Enable PT-RS
`0` (default) | `1`

Enable PT-RS, specified as one of these values.

- `0` — Disable PT-RS configuration.
- `1` — Enable PT-RS configuration.

Data Types: `double` | `logical`

PTRS — PUSCH PT-RS configuration parameters
`nrPUSCHPTRSConfig` object with default properties (default) | `nrPUSCHPTRSConfig` object

PUSCH phase tracking reference signal (PT-RS) configuration, specified as a `nrPUSCHPTRSConfig` configuration object. This property relates to the phase tracking reference signal configuration and contains all properties of the specified `nrPUSCHPTRSConfig` object.

tcr — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: `double`

tbs — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of `0` indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: `double`

oack — Payload length of HARQ-ACK bits
nonnegative integer

Payload length of the HARQ-ACK bits, specified as a nonnegative integer. A value of `0` indicates no HARQ-ACK transmission.

Data Types: `double`

ocsi1 — Payload length of CSI part 1 bits
nonnegative integer

Payload length of the CSI part 1 bits, specified as a nonnegative integer. A value of `0` indicates no CSI part 1 transmission.

Data Types: `double`

`ocsi2` — Payload length of CSI part 2 bits
nonnegative integer

Payload length of the CSI part 2 bits, specified as a nonnegative integer. A value of `0` indicates no CSI part 2 transmission. Nominally, the CSI part 2 is present only when CSI part 1 is present.

Data Types: `double`

`cw` — Received LLR soft bits
real-valued column vector | `[]`

Received log likelihood ratio (LLR) soft bits, returned as a real-valued column vector or `[]`. The length of `cw` must be equal to the bit capacity of the PUSCH.

Data Types: `single` | `double`

Output Arguments

`culsch` — Coded UL-SCH LLR soft bits
real-valued column vector

Coded UL-SCH LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `culsch` is also empty. The output data type of `culsch` matches that of the input argument `cw`.

Data Types: `single` | `double`

`cack` — Coded HARQ-ACK LLR soft bits
real-valued column vector

Coded HARQ-ACK LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `cack` is also empty. The output data type of `cack` matches that of the input argument `cw`.

Data Types: `single` | `double`

`ccsi1` — Coded CSI part 1 LLR soft bits
real-valued column vector

Coded CSI part 1 LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `ccsi1` is also empty. The output data type of `ccsi1` matches that of the input argument `cw`.

Data Types: `single` | `double`

`ccsi2` — Coded CSI part 2 LLR soft bits
real-valued column vector

Coded CSI part 2 LLR soft bits, specified as a real-valued column vector. If the input argument `cw` is empty, then the output argument `ccsi2` is also empty. The output data type of `ccsi2` matches that of the input argument `cw`.

Data Types: `single` | `double`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`nrPUSCHConfig` | `nrULSCHDecoder`

Functions

`nrPUSCHDecode` | `nrPUSCHDescramble` | `nrRateRecoverLDPC` | `nrRateRecoverPolar` | `nrUCIDecode` | `nrULSCHInfo` | `nrULSCHMultiplex`

Introduced in R2020b

nrULSCHInfo

Get uplink shared channel (UL-SCH) information

Syntax

```
info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2)
info = nrULSCHInfo(tbs,tcr)
```

Description

`info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2)` returns a structure, which contains the uplink shared transmission (UL-SCH) information related to the encoding process and uplink control information (UCI) multiplexing, for physical uplink shared channel (PUSCH) configuration `pusch`, target code rate `tcr`, and transport block size `tbs`. The `oack` input is the hybrid automatic repeat request acknowledgment (HARQ-ACK) payload length. The `ocsi1` input is the channel state information (CSI) part 1 payload length. The `ocsi2` input is the CSI part 2 payload length.

The function performs the multiplexing process on one of these options.

- UL-SCH data and UCI data (HARQ-ACK, CSI part 1, and CSI part 2)
- UCI data (HARQ-ACK, CSI part 1, and CSI part 2) only

`info = nrULSCHInfo(tbs,tcr)` returns a structure containing UL-SCH information for an input transport block size `tbs` and target code rate `tcr`. The UL-SCH information includes the cyclic redundancy check (CRC) attachment, code block segmentation (CBS), and channel coding. When you use this syntax, the function provides the UL-SCH coding information and does not handle UCI multiplexing on PUSCH, because the information of PUSCH resources is not known.

Examples

Get Bit Capacity Information for UL-SCH Data and UCI

Create a PUSCH object with a default configuration.

```
pusch = nrPUSCHConfig;
```

Set the beta offsets of the UCI types in the PUSCH configuration. Set the UCI scaling factor.

```
pusch.BetaOffsetACK = 10;  % Beta offset factor for HARQ-ACK
pusch.BetaOffsetCSI1 = 10; % Beta offset factor for CSI part 1
pusch.BetaOffsetCSI2 = 10; % Beta offset factor for CSI part 2
pusch.UCIScaling = 1;      % Scaling factor
```

Set the target code rate for the shared channel transmission.

```
tcr = 517/1024; % Target code rate
```

Set the payload lengths of the data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tbs = 8456; % Payload length of UL-SCH data (transport block size)
oack = 6;   % Payload length of HARQ-ACK
```

```
ocsi1 = 40; % Payload length of CSI part 1
ocsi2 = 10; % Payload length of CSI part 2
```

Obtain the bit capacity information of the data and UCI.

```
info = nrULSCHInfo(pusch,tcr,tbs,ocsi1,ocsi2,oack)
```

```
info = struct with fields:
        CRC: '24A'
          L: 24
        BGN: 1
          C: 2
        Lcb: 24
          F: 312
         Zc: 208
          K: 4576
          N: 13728
     GULSCH: 15032
       GACK: 906
      GCSI1: 178
      GCSI2: 108
    GACKRvd: 0
      QdACK: 453
     QdCSI1: 89
     QdCSI2: 54
```

Get UL-SCH Information

Show UL-SCH information before rate matching for an input transport block of length 8456 and target code rate 517/1024. The displayed UL-SCH information shows:

- The transport block has 312 <NULL> filler bits per code block.
- The number of bits per code block, after CBS, is 4576.
- The number of bits per code block, after low-density parity-check (LDPC) coding, is 13,728.

```
tBlkLen = 8456;
targetCodeRate = 517/1024;
nrULSCHInfo(tBlkLen,targetCodeRate)
```

```
ans = struct with fields:
        CRC: '24A'
          L: 24
        BGN: 1
          C: 2
        Lcb: 24
          F: 312
         Zc: 208
          K: 4576
          N: 13728
     GULSCH: []
       GACK: 0
      GCSI1: 0
      GCSI2: 0
    GACKRvd: 0
```

```
 QdACK: 0
QdCSI1: 0
QdCSI2: 0
```

Input Arguments

`pusch` — PUSCH configuration parameters

`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

`Modulation` — Modulation scheme

`'QPSK'` (default) | `'pi/2-BPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, or a character array.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`NumLayers` — Number of transmission layers

`1` (default) | `2` | `3` | `4`

Number of transmission layers, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`MappingType` — Mapping type

`'A'` (default) | `'B'`

Mapping type of the physical shared channel, specified as `'A'` or `'B'`.

Data Types: `char` | `string`

`SymbolAllocation` — OFDM symbol allocation

`[0 14]` (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When this property is set to `[]` or second element in this two-element vector of nonnegative integers is `0`, the symbol allocation is empty.

Data Types: `double`

PRBSet — PRB allocation
`[0:51]` (default) | vector of nonnegative integers from 0 to 274

Physical resource block (PRB) allocation of PUSCH within the BWP, specified as a vector of nonnegative integers from 0 to 274.

Data Types: `double`

TransformPrecoding — Transform precoding flag
`0` (default) | `1`

Transform precoding flag, specified as one of these values.

- `0` — The transform precoding is disabled and waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).
- `1` — The transform precoding is enabled and waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

Data Types: `double` | `logical`

FrequencyHopping — Frequency hopping
`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping for the physical uplink shared channel, specified as `'neither'`, `'intraSlot'`, or `'interSlot'`.

Data Types: `char` | `string`

BetaOffsetACK — Beta offset factor of HARQ-ACK
`20` (default) | positive integer

Beta offset factor of the hybrid automatic repeat request acknowledgment (HARQ-ACK), specified as a positive integer. This property is used to determine the number of resources for multiplexing HARQ-ACK. The nominal value is one of the entry from the Table 9.3-1 of TS 38.213.

Data Types: `double`

BetaOffsetCSI1 — Beta offset factor of CSI part 1
`6.25` (default) | positive integer

Beta offset factor of the channel state information (CSI) part 1, specified as a positive integer. This property is used to determine the number of resources for multiplexing CSI part 1. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

BetaOffsetCSI2 — Beta offset factor of CSI part 2
`6.25` (default) | positive integer

Beta offset factor of the CSI part 2, specified as a positive integer. This property is used to determine the number of resources for multiplexing CSI part 2. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

UCIScaling — Scaling factor
`1` (default) | scalar in the range (0, 1)

Scaling factor to limit the number of the resource elements allocated for the uplink channel information (UCI) on the PUSCH, specified as a scalar in the range (0, 1). The nominal value is 0.5, 0.65, 0.8, or 1.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`DMRS` — PUSCH DM-RS configuration parameters
`nrPUSCHDMRSConfig` object with default properties (default) | `nrPUSCHDMRSConfig` object

PUSCH demodulation reference signal (DM-RS) configuration parameters, specified as a `nrPUSCHDMRSConfig` configuration object. This property relates to the demodulation reference signal configuration and contains all properties of the specified `nrPUSCHDMRSConfig` object.

`EnablePTRS` — Enable PT-RS
`0` (default) | `1`

Enable PT-RS, specified as one of these values.

- `0` — Disable PT-RS configuration.
- `1` — Enable PT-RS configuration.

Data Types: `double` | `logical`

`PTRS` — PUSCH PT-RS configuration parameters
`nrPUSCHPTRSConfig` object with default properties (default) | `nrPUSCHPTRSConfig` object

PUSCH phase tracking reference signal (PT-RS) configuration, specified as a `nrPUSCHPTRSConfig` configuration object. This property relates to the phase tracking reference signal configuration and contains all properties of the specified `nrPUSCHPTRSConfig` object.

`tcr` — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: `double`

`tbs` — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of `0` indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: `double`

`oack` — Payload length of HARQ-ACK bits
nonnegative integer

Payload length of the HARQ-ACK bits, specified as a nonnegative integer. A value of `0` indicates no HARQ-ACK transmission.

Data Types: `double`

`ocsi1` — Payload length of CSI part 1 bits

nonnegative integer

Payload length of the CSI part 1 bits, specified as a nonnegative integer. A value of `0` indicates no CSI part 1 transmission.

Data Types: `double`

`ocsi2` — Payload length of CSI part 2 bits

nonnegative integer

Payload length of the CSI part 2 bits, specified as a nonnegative integer. A value of `0` indicates no CSI part 2 transmission. Nominally, the CSI part 2 is present only when CSI part 1 is present.

Data Types: `double`

Output Arguments

`info` — UL-SCH information

structure

UL-SCH information, returned as a structure containing these fields.

Fields	Values	Description
`CRC`	`'16'`, `'24A'`	CRC polynomial selection
`L`	`0`, 16, 24	Number of CRC bits
`BGN`	1, 2	LDPC base graph selection
`C`	Positive integer	Number of code blocks
`Lcb`	`0`, 24	Number of parity bits per code block
`F`	Nonnegative integer	Number of <NULL> filler bits per code block
`Zc`	Positive integer	Lifting size selection
`K`	Nonnegative integer	Number of bits per code block after CBS
`N`	Nonnegative integer	Number of bits per code block after LDPC coding
`GULSCH`	Nonnegative integer	Number of coded and rate matched UL-SCH data bits
`GACK`	Nonnegative integer	Number of coded and rate matched HARQ-ACK bits
`GCSI1`	Nonnegative integer	Number of coded and rate matched CSI part 1 bits
`GCSI2`	Nonnegative integer	Number of coded and rate matched CSI part 2 bits
`GACKRvd`	Nonnegative integer	Number of reserved bits for HARQ-ACK
`QdACK`	Nonnegative integer	Number of coded HARQ-ACK symbols per layer (Q'_{ACK})
`QdCSI1`	Nonnegative integer	Number of coded CSI part 1 symbols per layer (Q'_{CSI1})
`QdCSI2`	Nonnegative integer	Number of coded CSI part 2 symbols per layer (Q'_{CSI2})

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`nrPUSCHConfig` | `nrULSCH` | `nrULSCHDecoder`

Functions

`nrPUSCH` | `nrPUSCHDecode` | `nrULSCHDemultiplex` | `nrULSCHMultiplex`

Introduced in R2019a

nrULSCHMultiplex

Perform UL-SCH data and control multiplexing

Syntax

```
[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)
```

Description

`[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)` returns codeword `cw` by performing uplink shared channel (UL-SCH) multiplexing on the encoded UL-SCH data and the encoded uplink control information (UCI), as defined in TS 38.212 Section 6.2.7 [1]. `pusch` is the physical uplink shared channel configuration (PUSCH). `tcr` is the target code rate. `tbs` is the transport block size for the UL-SCH transmission. `culsch` is the encoded UL-SCH data. `cack`, `ccsi1`, and `ccsi2` are the encoded UCI types.

The function internally calculates the number of reserved bits for the hybrid automatic repeat request acknowledgment (HARQ-ACK) transmission, `GACKRvd` and then compares against the lengths of the coded inputs. This comparison determines the processing of the HARQ-ACK for rate-matching or puncturing.

The length of `cw` equals the bit capacity of the PUSCH. `cw` contains the encoded information up to the bit capacity of PUSCH and ignores any other additional information in the inputs. The output `cw` contains zeros when not enough encoded UL-SCH and encoded UCI (HARQ-ACK, channel state information (CSI) 1, or CSI part 2) data is present to achieve the bit capacity. The function also returns the structure `info`, which contains information about the 1-based locations of each type in the codeword.

Examples

Generate Codeword with Predefined Coded UL-SCH Data and Coded UCI Types

Create a default PUSCH configuration object. Allocate the first 21 resource blocks of the bandwidth part to the PUSCH.

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:20;
```

Set the target code rate, payload lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.5;  % Target code rate
tbs = 100;  % Payload length of UL-SCH data (transport block size)
oack = 3;   % Payload length of HARQ-ACK
ocsi1 = 10; % Payload length of CSI part 1
ocsi2 = 10; % Payload length of CSI part 2
```

Get the rate matched lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the predefined coded bits of the UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2 for the rate-matched output length obtained from the `rmInfo` structure.

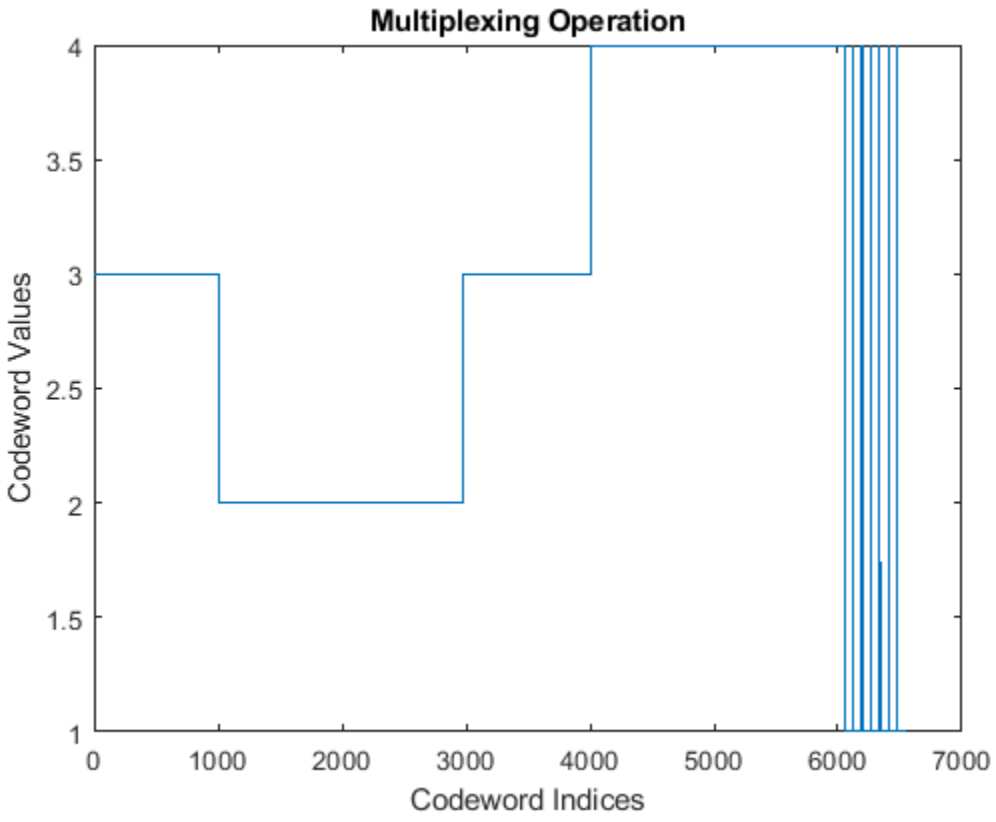
```
culsch = ones(rmInfo.GULSCH,1);
cack = 2*ones(rmInfo.GACK,1);
ccsi1 = 3*ones(rmInfo.GCSI1,1);
ccsi2 = 4*ones(rmInfo.GCSI2,1);
```

Get the codeword from the predefined coded UL-SCH data and coded UCI types.

```
cw = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);
```

To see the multiplexing operation, plot the codeword. The codeword starts with the elements of CSI part 1, followed by HARQ-ACK, CSI part 1, CSI part 2, and the mix of UL-SCH data and CSI part 2.

```
plot(cw)
xlabel('Codeword Indices')
ylabel('Codeword Values')
title('Multiplexing Operation')
```



Generate Codeword with Coded UL-SCH Data and Coded UCI

Create a PUSCH configuration object with a pi/2-BPSK modulation scheme and no frequency hopping. Set the beta offset factor for HARQ-ACK to 20, and the beta offset factor for CSI part 1 and

CSI part 2 to 6.25 each. Specify the scaling factor as 0.8, which limits the number of resource elements (REs) assigned for the UCI.

```
pusch = nrPUSCHConfig;
pusch.Modulation = 'pi/2-BPSK';
pusch.FrequencyHopping = 'neither';
pusch.BetaOffsetACK = 20;
pusch.BetaOffsetCSI1 = 6.25;
pusch.BetaOffsetCSI2 = 6.25;
pusch.UCIScaling = 0.8;
```

Set the target code rate, payload lengths of the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
tcr = 0.5;   % Target code rate
tbs = 1032;  % Payload length of UL-SCH data (transport block size)
oack = 8;    % Payload length of HARQ-ACK
ocsi1 = 88;  % Payload length of CSI part 1
ocsi2 = 720; % Payload length of CSI part 2
```

Get the rate matched lengths of the data, HARQ-ACK, CSI part 1, and CSI part 2.

```
rmInfo = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);
```

Create the random payload bits for the UL-SCH data, HARQ-ACK, CSI part 1, and CSI part 2.

```
data = randi([0 1],tbs,1);
ack  = randi([0 1],oack,1);
csi1 = randi([0 1],ocsi1,1);
csi2 = randi([0 1],ocsi2,1);
```

Create a UL-SCH encoder System object.

```
encUL = nrULSCH;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,data);
```

Get the coded bits of length `rmInfo.GULSCH` by calling the encoder.

```
rv = 0; % Redundancy version is 0
culsch = encUL(pusch.Modulation,pusch.NumLayers,rmInfo.GULSCH,rv);
```

Encode the random payload of the HARQ-ACK, CSI part 1, and CSI part 2 for the rate-matched output lengths obtained from the `rmInfo` structure.

```
cack  = nrUCIEncode(ack,rmInfo.GACK,pusch.Modulation);
ccsi1 = nrUCIEncode(csi1,rmInfo.GCSI1,pusch.Modulation);
ccsi2 = nrUCIEncode(csi2,rmInfo.GCSI2,pusch.Modulation);
```

Get the codeword from the coded bits of the UL-SCH and the coded bits of UCI types.

```
[cw,info] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2)
```

```
cw = 8112x1 int8 column vector

   1
   0
   1
```

```
   0
   0
   0
   1
   1
   1
   0
      ⋮

info = struct with fields:
    ULSCHIndices: [1622x1 uint32]
      ACKIndices: [1159x1 uint32]
     CSI1Indices: [4482x1 uint32]
     CSI2Indices: [849x1 uint32]
     UCIXIndices: [0x1 uint32]
     UCIYIndices: [0x1 uint32]
```

Input Arguments

`pusch` — PUSCH configuration parameters

`nrPUSCHConfig` object

PUSCH configuration parameters, specified as an `nrPUSCHConfig` object. This function uses only these `nrPUSCHConfig` object properties.

`Modulation` — Modulation scheme

`'QPSK'` (default) | `'pi/2-BPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, or a character array.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`NumLayers` — Number of transmission layers

1 (default) | 2 | 3 | 4

Number of transmission layers, specified as 1, 2, 3, or 4.

Data Types: `double`

`MappingType` — Mapping type

`'A'` (default) | `'B'`

Mapping type of the physical shared channel, specified as `'A'` or `'B'`.

Data Types: `char` | `string`

`SymbolAllocation` — OFDM symbol allocation
`[0 14]` (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When this property is set to `[]` or second element in this two-element vector of nonnegative integers is `0`, the symbol allocation is empty.

Data Types: `double`

`PRBSet` — PRB allocation
`[0:51]` (default) | vector of nonnegative integers from 0 to 274

Physical resource block (PRB) allocation of PUSCH within the BWP, specified as a vector of nonnegative integers from 0 to 274.

Data Types: `double`

`TransformPrecoding` — Transform precoding flag
`0` (default) | `1`

Transform precoding flag, specified as one of these values.

- `0` — The transform precoding is disabled and waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).
- `1` — The transform precoding is enabled and waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

Data Types: `double` | `logical`

`FrequencyHopping` — Frequency hopping
`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping for the physical uplink shared channel, specified as one of these options.

- `'neither'`
- `'intraSlot'`
- `'interSlot'`

If you specify intra-slot frequency hopping and the input coded bits doesn't satisfy the equations of TS 38.212 Section 6.2.7, the function returns the codeword assuming no frequency hopping is present.

Data Types: `char` | `string`

`BetaOffsetACK` — Beta offset factor of HARQ-ACK
`20` (default) | positive integer

Beta offset factor of the hybrid automatic repeat request acknowledgment (HARQ-ACK), specified as a positive integer. This property is used to determine the number of resources for multiplexing HARQ-ACK. The nominal value is one of the entry from the Table 9.3-1 of TS 38.213.

Data Types: `double`

`UCIScaling` — Scaling factor
`1` (default) | scalar in the range (0, 1)

Scaling factor to limit the number of the resource elements allocated for the uplink channel information (UCI) on the PUSCH, specified as a scalar in the range (0, 1). The nominal value is 0.5, 0.65, 0.8, or 1.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`DMRS` — PUSCH DM-RS configuration parameters
`nrPUSCHDMRSConfig` object with default properties (default) | `nrPUSCHDMRSConfig` object

PUSCH demodulation reference signal (DM-RS) configuration parameters, specified as a `nrPUSCHDMRSConfig` configuration object. This property relates to the demodulation reference signal configuration and contains all properties of the specified `nrPUSCHDMRSConfig` object.

`EnablePTRS` — Enable PT-RS
`0` (default) | `1`

Enable PT-RS, specified as one of these values.

- `0` — Disable PT-RS configuration.
- `1` — Enable PT-RS configuration.

Data Types: `double` | `logical`

`PTRS` — PUSCH PT-RS configuration parameters
`nrPUSCHPTRSConfig` object with default properties (default) | `nrPUSCHPTRSConfig` object

PUSCH phase tracking reference signal (PT-RS) configuration, specified as a `nrPUSCHPTRSConfig` configuration object. This property relates to the phase tracking reference signal configuration and contains all properties of the specified `nrPUSCHPTRSConfig` object.

`tcr` — Target code rate
scalar in the range (0, 1)

Target code rate for the codeword in the UL-SCH transmission, specified as a scalar in the range (0, 1).

Data Types: `double`

`tbs` — Transport block size
nonnegative integer

Transport block size associated with the codeword in the UL-SCH transmission, specified as a nonnegative integer. A value of `0` indicates no transport block or no UL-SCH transmission on the PUSCH.

Data Types: `double`

`culsch` — Coded UL-SCH data bits

binary-valued column vector

Coded UL-SCH data bits, specified as a binary-valued column vector of length `gulsch`. The `gulsch` is the number of coded and rate matched UL-SCH data bits returned in the `info` output argument of the `nrULSCHInfo` function. The encoded UL-SCH bits, `culsch`, is the encoded and rate-matched bits obtained by processing the transport block. A value of `[]` indicates the absence of UL-SCH data transmission. If you specify the `tbs` input argument as `0`, the `culsch` input argument must be empty.

The `gulsch` must be equal to the bit capacity of the data.

Data Types: `int8` | `double`

`cack` — Coded HARQ-ACK bits

real-valued column vector

Coded HARQ-ACK bits, specified as a real-valued column vector of length `gack`. The `gack` is the number of coded and rate matched HARQ-ACK bits returned in the `info` output argument of the `nrULSCHInfo` function. A value of `[]` indicates the absence of HARQ-ACK transmission. The nominal values of the HAQR-ACK bits are `0`, `1`, `-1`, and `-2`.

The `gack` must be a multiple of the product of the number of transmission layers and the modulation order.

Data Types: `int8` | `double`

`ccsi1` — Coded CSI part 1 bits

real-valued column vector

Coded CSI part 1 bits, specified as a real-valued column vector of length `gcsi1`. The `gcsi1` is the number of coded and rate matched CSI part 1 bits returned in the `info` output argument of the `nrULSCHInfo` function. A value of `[]` indicates the absence of the CSI part 1 transmission. The nominal values of the CSI part 1 bits are `0`, `1`, `-1`, and `-2`.

The `gcsi1` must be a multiple of the product of the number of transmission layers and the modulation order.

Data Types: `int8` | `double`

`ccsi2` — Coded CSI part 2 bits

real-valued column vector

Coded CSI part 2 bits, specified as a real-valued column vector of length `gcsi2`. The `gcsi2` is the number of coded and rate matched CSI part 2 bits returned in the `info` output argument of the `nrULSCHInfo` function. A value of `[]` indicates the absence of the CSI part 2 transmission. The nominal values of the CSI part 2 bits are `0`, `1`, `-1`, and `-2`. Nominally, the CSI part 2 is present only when CSI part 1 is present.

The `gcsi2` must be a multiple of the product of the number of transmission layers and the modulation order.

Data Types: `int8` | `double`

Output Arguments

`cw` — Codeword

real-valued column vector

Codeword for the transmission on the PUSCH, returned as a real-valued column vector. If you provide any of the input arguments as data type `int8`, the output data type of the codeword is `int8`. The nominal values of the bits in the codewords are `0`, `1`, `-1`, and `-2`.

The length of `cw` equals the bit capacity of the PUSCH.

The `cw` output is an empty value of size `0`-by-`1` for all of these cases.

- When the `PRBSet` property of the `pusch` argument is `[]`.
- When the `SymbolAllocation` property of the `pusch` argument is `[]` or when the number of contiguous OFDM symbols allocated for PUSCH is zero
- When all of the input arguments `culsch`, `cack`, `ccsi1`, and `ccsi2` are empty

Data Types: `int8` | `double`

`info` — Location information

structure

Location information about the 1-based locations of each type in output codeword `cw`, returned as a structure containing these fields. The output data type of each field is uint32.

Field	Description
`ULSCHIndices`	Locations of coded UL-SCH bits in codeword
`ACKIndices`	Locations of coded HARQ-ACK bits in codeword
`CSI1Indices`	Locations of coded CSI part 1 bits in codeword
`CSI2Indices`	Locations of coded CSI part 2 bits in codeword
`UCIXIndices`	Locations of X UCI placeholders in codeword
`UCIYIndices`	Locations of Y UCI placeholders in codeword

If the returned codeword, `cw`, is an empty array, each field in this structure is also an empty array.

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrPUSCHConfig` | `nrULSCH`

Functions
`nrPUSCH` | `nrPUSCHScramble` | `nrRateMatchLDPC` | `nrRateMatchPolar` | `nrUCIEncode` | `nrULSCHDemultiplex` | `nrULSCHInfo`

Topics
“NR UCI Multiplexing on PUSCH”

Introduced in R2020b

resetSoftBuffer

Reset soft buffer for HARQ process in UL-SCH or DL-SCH decoder

Syntax

```
resetSoftBuffer(decUL)
resetSoftBuffer(decDL,cwid)
resetSoftBuffer = ( ___ ,harqID)
```

Description

`resetSoftBuffer(decUL)` resets the soft buffer for hybrid automatic repeat-request (HARQ) process number 0 in the specified UL-SCH decoder `decUL`.

`resetSoftBuffer(decDL,cwid)` resets the soft buffer for codeword index `cwid` and HARQ process number 0 in the specified DL-SCH decoder `decDL`. The codeword index `cwid` specifies one of the two possible codewords for DL-SCH decoding.

`resetSoftBuffer = ( ___ ,harqID)` resets the soft buffer for the specified HARQ process number `harqID`. Specify `harqID` in addition to the input arguments in any of the previous syntaxes.

To enable soft combining of retransmissions before low-density parity-check (LDPC) decoding, each decoder object maintains a soft buffer for each HARQ process. Upon successful decoding of the input, the object automatically resets the soft buffer for the HARQ process. Calling the `resetSoftBuffer` function resets the soft buffer manually. Call this function when decoding different transport blocks for the same HARQ process subsequently or when all redundancy versions for a HARQ process are complete.

Examples

Reset Soft Buffer in UL-SCH Decoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen1 = 5120;
trBlk1 = randi([0 1],trBlkLen1,1,'int8');
```

Create and configure a UL-SCH encoder System object with multiple HARQ processes.

```
encUL = nrULSCH('MultipleHARQProcesses',true);
```

Load the transport block into the UL-SCH encoder for HARQ process number 1.

```
harqID = 1;
setTransportBlock(encUL,trBlk1,harqID);
```

Call the encoder with QPSK modulation scheme, 1 transmission layer, an output length of 10,240 bits, redundancy version 0, and HARQ process number 1. The encoder applies the UL-SCH processing chain to the transport block loaded into the object using HARQ process number 1.

```
rv = 0;
codedTrBlock1 = encUL('QPSK',1,10240,rv,harqID);
```

Create and configure an UL-SCH decoder System object with multiple HARQ processes.

```
decUL = nrULSCHDecoder('MultipleHARQProcesses',true);
```

Configure the decoder for the encoded transport block.

```
decUL.TransportBlockLength = trBlkLen1;
```

Add noise to the soft bits representing the encoded transport block. Call the UL-SCH decoder on the modified soft bits for HARQ process number 1.

```
rxSoftBits1 = awgn(1-2*double(codedTrBlock1),5);
[decBits1,blkErr1] = decUL(rxSoftBits1,'QPSK',1,rv,harqID);
```

The added noise results in an error during the decoding.

```
blkErr1
```

```
blkErr1 = logical
   1
```

Repeat the encoding operation for a new transport block of length 4400 and HARQ process number 1.

```
trBlkLen2 = 4400;
trBlk2 = randi([0 1],trBlkLen2,1,'int8');
setTransportBlock(encUL,trBlk2,harqID);
codedTrBlock2 = encUL('QPSK',1,8800,rv,harqID);
```

Configure the decoder for the second transport block.

```
decUL.TransportBlockLength = trBlkLen2;
```

If an error occurred during the previous decoding with HARQ process number 1, you must reset the soft buffer of the HARQ process before decoding the second transport block.

```
if blkErr1
    resetSoftBuffer(decUL,harqID);
end
```

Call the decoder on the soft bits representing the second encoded transport block using HARQ process number 1.

```
rxBits2 = 1-2*double(codedTrBlock2);
[decBits2,blkErr2] = decUL(rxBits2,'QPSK',1,rv,harqID);
blkErr2
```

```
blkErr2 = logical
   0
```

Verify that the second transmitted and decoded message bits are identical.

```
isequal(decBits2,trBlk2)
```

```
ans = logical
   1
```

Reset Soft Buffer in DL-SCH Decoder with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen1 = 5120;
trBlk1 = randi([0 1],trBlkLen1,1,'int8');
```

Create and configure a DL-SCH encoder System object with multiple HARQ processes.

```
encDL = nrDLSCH('MultipleHARQProcesses',true);
```

Load the transport block into the DL-SCH encoder for HARQ process number 1 and codeword index 1.

```
harqID = 1;
cwID = 0;
setTransportBlock(encDL,trBlk1,cwID,harqID);
```

Call the encoder with QPSK modulation scheme, 1 transmission layer, an output length of 10,240 bits, redundancy version 0, and HARQ process number 1. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 1.

```
rv = 0;
codedTrBlock1 = encDL('QPSK',1,10240,rv,harqID);
```

Create and configure a DL-SCH decoder System object with multiple HARQ processes.

```
decDL = nrDLSCHDecoder('MultipleHARQProcesses',true);
```

Configure the decoder for the encoded transport block.

```
decDL.TransportBlockLength = trBlkLen1;
```

Add noise to the soft bits representing the encoded transport block. Call the DL-SCH decoder on the modified soft bits for HARQ process number 1.

```
rxSoftBits1 = awgn(1-2*double(codedTrBlock1),5);
[decBits1,blkErr1] = decDL(rxSoftBits1,'QPSK',1,rv,harqID);
```

The added noise results in an error during decoding.

```
blkErr1
```

```
blkErr1 = logical
   1
```

Repeat the encoding operation for a new transport block of length 4400 and HARQ process number 1.

```
trBlkLen2 = 4400;
trBlk2 = randi([0 1],trBlkLen2,1,'int8');
```

```
setTransportBlock(encDL,trBlk2,cwID,harqID);
codedTrBlock2 = encDL('QPSK',1,8800,rv,harqID);
```

Configure the decoder for the second transport block.

```
decDL.TransportBlockLength = trBlkLen2;
```

If an error occurred during the previous decoding with HARQ process number 1, you must reset the soft buffer of the HARQ process before decoding the second transport block.

```
if blkErr1
    resetSoftBuffer(decDL,harqID);
end
```

Call the decoder on the soft bits representing the second encoded transport block using HARQ process number 1.

```
rxBits2 = 1-2*double(codedTrBlock2);
[decBits2,blkErr2] = decDL(rxBits2,'QPSK',1,rv,harqID);
blkErr2
```

```
blkErr2 = logical
   0
```

Verify that the second transmitted and decoded message bits are identical.

```
isequal(decBits2,trBlk2)
```

```
ans = logical
   1
```

Input Arguments

decUL — UL-SCH decoder

`nrULSCHDecoder` System object

UL-SCH decoder, specified as a `nrULSCHDecoder` System object. The object implements the UL-SCH decoder processing chain corresponding to the inverse operation of UL-SCH encoding specified in TR 38.212 Section 6.2.

decDL — DL-SCH decoder

`nrDLSCHDecoder` System object

DL-SCH decoder, specified as a `nrDLSCHDecoder` System object. The object implements the DL-SCH decoder processing chain corresponding to the inverse operation of DL-SCH encoding specified in TR 38.212 Section 7.2.

cwid — DL-SCH codeword index

`0` | `1`

DL-SCH codeword index, specified as `0` or `1`.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`nrDLSCHDecoder` | `nrULSCHDecoder`

Introduced in R2019a

setTransportBlock

Load transport block into UL-SCH or DL-SCH encoder

Syntax

```
setTransportBlock(enc,trblk)
setTransportBlock(encDL,trblk,trblkID)
setTransportBlock( ___ ,harqID)
```

Description

`setTransportBlock(enc,trblk)` loads the transport block `trblk` into the specified uplink (UL) or downlink (DL) shared channel (SCH) encoder System object `enc`. Call this function before calling `enc`.

`setTransportBlock(encDL,trblk,trblkID)` loads the transport block `trblk` into the specified DL-SCH encoder System object `encDL` for the specified transport block number `trblkID`. Call this function before calling `encDL`.

`setTransportBlock( ___ ,harqID)` loads the transport block for the specified hybrid automatic repeat-request (HARQ) process number `harqID`. Specify `harqID` in addition to the input arguments in any of the previous syntaxes.

Examples

UL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object for use with multiple HARQ processes.

```
encUL = nrULSCH;
encUL.MultipleHARQProcesses = true;
```

Load the transport block into the UL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
setTransportBlock(encUL,trBlk,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the UL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encUL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

DL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object for use with multiple HARQ processes.

```
encDL = nrDLSCH;
encDL.MultipleHARQProcesses = true;
```

Load transport block `trBlk` for transport block number 0 into the DL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
trBlkID = 0;
setTransportBlock(encDL,trBlk,trBlkID,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encDL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

Input Arguments

enc — UL-SCH or DL-SCH encoder

`nrULSCH` System object | `nrDLSCH` System object

UL-SCH or DL-SCH encoder, specified as an `nrULSCH` or `nrDLSCH` System object. The objects implement the UL-SCH and DL-SCH processing chains specified in TS 38.212 Section 6.2 and Section 7.2, respectively.

`trblk` — Transport block
binary column vector | cell array of one or two binary column vectors

Transport block, specified as a binary column vector or a cell array of one or two binary column vectors. The cell array option applies only when `enc` is a DL-SCH encoder System object. The two-element cell array option applies only when the transport block number `trblkID` is not specified for DL-SCH processing.

Data Types: `int8` | `double` | `logical`

`encDL` — DL-SCH encoder
`nrDLSCH` System object

DL-SCH encoder, specified as an `nrDLSCH` System object. The object implements the DL-SCH processing chain specified in TS 38.212 Section 7.2.

`trblkID` — Transport block number in DL-SCH processing
`0` (default) | `1`

Transport block number in DL-SCH processing, specified as `0` or `1`.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`getTransportBlock`

Objects
`nrDLSCH` | `nrULSCH`

Introduced in R2019a

nrWaveformGenerator

Generate 5G NR waveform

Syntax

```
[wave,info] = nrWaveformGenerator(cfg)
```

```
nrWaveformGenerator
```

Description

`[wave,info] = nrWaveformGenerator(cfg)` generates 5G NR waveform `wave` for the specified configuration `cfg`. The input `cfg` contains configuration parameters for single or multiple subcarrier spacing (SCS) carriers, bandwidth parts (BWPs), synchronization signal (SS) burst, control-resource sets (CORESETs), search spaces, physical shared channels and control channels and associated reference signals, and channel-state information reference signals (CSI-RS). The function also returns a structure, `info`, containing information about the resource grid, physical downlink shared channels (PDSCH), and physical downlink control channels (PDCCH).

`nrWaveformGenerator` opens the **5G Waveform Generator** app.

Examples

Configure and Generate Single-User 5G Downlink Waveform

Create an SCS carrier configuration object with the default SCS of 15 kHz and 100 resource blocks.

```
carrier = nrSCSCarrierConfig('NSizeGrid',100);
```

Create a customized BWP configuration object for the SCS carrier.

```
bwp = nrWavegenBWPConfig('NStartBWP',carrier.NStartGrid+10);
```

Create an SS burst configuration object with block pattern Case A.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create a PDCCH configuration object, specifying an aggregation of size two and the fourth candidate for the PDCCH instance.

```
pdcch = nrWavegenPDCCHConfig('AggregationLevel',2,'AllocatedCandidate',4);
```

Create a CORESET configuration object, specifying four frequency resources and a duration of three OFDM symbols.

```
coreset = nrCORESETConfig;
coreset.FrequencyResources = [1 1 1 1];
coreset.Duration = 3;
```

Create a search space set configuration object, specifying two aggregation levels.

```
ss = nrSearchSpaceConfig;
ss.NumCandidates = [8 4 0 0 0];
```

Create a PDSCH configuration object, specifying the modulation scheme and the target code rate. Enable the PDSCH PT-RS.

```
pdsch = nrWavegenPDSCHConfig( ...
    'Modulation','16QAM','TargetCodeRate',658/1024,'EnablePTRS',true);
```

Create a PDSCH DM-RS and a PDSCH PT-RS configuration object with the specified property values.

```
dmrs = nrPDSCHDMRSConfig('DMRSTypeAPosition',3);
pdsch.DMRS = dmrs;
ptrs = nrPDSCHPTRSConfig('TimeDensity',2);
pdsch.PTRS = ptrs;
```

Create a CSI-RS configuration object with the specified property values.

```
csirs = nrWavegenCSIRSConfig('RowNumber',4,'RBOffset',10);
```

Create a single-user 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',{carrier}, ...
    'BandwidthParts',{bwp}, ...
    'SSBurst',ssb, ...
    'CORESET',{coreset}, ...
    'SearchSpaces',{ss}, ...
    'PDCCH',{pdcch}, ...
    'PDSCH',{pdsch}, ...
    'CSIRS',{csirs});
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Configure and Generate Multiuser 5G Downlink Waveform

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two custom BWP configuration objects, one for each of the carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create an SS burst configuration object with block pattern Case A, corresponding to an SCS of 15 kHz.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create two PDCCH configuration objects.

```
pdcch = {
    nrWavegenPDCCHConfig('SearchSpaceID',1,'BandwidthPartID',1,'RNTI',1,'DMRSScramblingID',1), ...
    nrWavegenPDCCHConfig('SearchSpaceID',2,'BandwidthPartID',2,'RNTI',2,'DMRSScramblingID',2, ...
        'AggregationLevel',4)};
```

Create two CORESET configuration objects and two search space set configuration objects for the two PDCCH.

```
coreset = {
    nrCORESETConfig('CORESETID',1,'FrequencyResources',[1 1 1 1 1 0 0 0 0 0 1],'Duration',3), ...
    nrCORESETConfig('CORESETID',2,'FrequencyResources',[0 0 0 0 0 0 0 0 1 1])};

ss = {
    nrSearchSpaceConfig('SearchSpaceID',1,'CORESETID',1,'StartSymbolWithinSlot',4), ...
    nrSearchSpaceConfig('SearchSpaceID',2,'CORESETID',2,'NumCandidates',[8 8 4 0 0])};
```

Create two PDSCH configuration objects with mixed modulation schemes.

```
pdsch = {
    nrWavegenPDSCHConfig('BandwidthPartID',1,'Modulation','16QAM','RNTI',1,'NID',1), ...
    nrWavegenPDSCHConfig('BandwidthPartID',2,'Modulation','QPSK','RNTI',2,'NID',2, ...
            'PRBSet', 50:59)};
```

Create two CSI-RS configuration objects.

```
 csirs = {
     nrWavegenCSIRSConfig('BandwidthPartID',1,'RowNumber',2,'RBOffset',10), ...
     nrWavegenCSIRSConfig('BandwidthPartID',2,'Density','three','RowNumber',4)};
```

Create a multiuser 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'SSBurst',ssb, ...
    'CORESET',coreset, ...
    'SearchSpaces',ss, ...
    'PDCCH',pdcch, ...
    'PDSCH',pdsch, ...
    'CSIRS',csirs);
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Input Arguments

`cfg` — Configuration parameters for 5G NR waveform generation

`nrDLCarrierConfig` object

Configuration parameters for 5G NR waveform generation, specified as an `nrDLCarrierConfig` object.

Output Arguments

`wave` — Time-domain 5G NR waveform

complex matrix

Time-domain 5G NR waveform, returned as a complex matrix. The number of matrix columns correspond to the number of transmit antennas.

Data Types: `double`

`info` — Metadata of 5G waveform

structure

Metadata of 5G waveform, returned as a structure with these fields.

`ResourceGrids` — BWP information

structure

BWP information, returned as a structure with these fields.

Fields	Values	Description
`ResourceGridBWP`	Complex 2-D or 3-D array	BWP resource grid
`ResourceGridInCarrier`	Complex 2-D or 3-D array	BWP resource grid in carrier

Fields	Values	Description
Info	Structure array	Each structure in the array contains these fields.

Fields	Values	Description
Nfft	Positive integer	Number of fast Fourier transform (FFT) points
SampleRate	Real number	Waveform sample rate
CyclicPrefixLengths	Row vector of positive integers	Cyclic prefix lengths of each OFDM symbol in a subframe, in samples
SymbolLengths	Row vector of positive integers	OFDM symbol lengths, in samples
Windowing	Positive integer	Number of time-domain samples over which the function applies raised cosine windowing and overlapping of OFDM symbols
SymbolPhases	Integer vector	Phase compensation of each OFDM symbol, in radians
SymbolsPerSlot	12 or 14	Number of OFDM symbols in a slot
SlotsPerSubframe	1, 2, 4, or 8	Number of slots in a 1 ms subframe
SlotsPerFrame	Positive integer	Number of slots in a 10 ms frame
k0	Nonnegative integer	Frequency starting position per antenna port and OFDM symbol

`WaveformResources` — Channel information
structure

Channel information, returned as a structure with these fields.

Fields	Values	Description
PDCCH	1-by-N_{PDCCH} structure array, N_{PDCCH} numb config PDCC input	Each structure in the array contains these fields.

Fields	Values	Description
Name	Character array	Name of PDCCH configuration
CDMLengths	Two-element integer vector	CDM arrangement for reference signals
Resources	1-by-M_{PDCCH} str ar wh $M_{}$ th of slo th sp PD	Each structure in the array contains these fields.

Fields	Values	Description
NSlot	Nonnegative integer	Slot number
DCIBits	Binary column vector	Downlink control information (DCI) bits
Codeword	Binary column vector	Encoded DCI codeword
G	Nonnegative integer	Bit capacity of the PDCCH
Gd	Nonnegative integer	Number of resource elements per layer or port
ChannelIndices	Column vector of positive integers	PDCCH indices
ChannelSymbols	Complex column vector	PDCCH symbols
DMRSIndices	Column vector of positive integers	PDCCH demodulation reference signal (DM-RS) indices
DMRSSymbols	Complex column vector	PDCCH DM-RS symbols

Fields	Values	Description
PDSCH	1-by-N_{PDSCH} structure array, N_{PDSCH} numb config PDSC	Each structure in the array contains these fields.

Fields	Values	Description
Name	Character array	Name of PDSCH configuration
CDMLengths	Two-element integer vector	CDM arrangement for reference signals
Resources	1-by-M_{PDSCH} str ar wh $M_{}$ th of slo th sp PD	Each structure in the array contains these fields.

Fields	Values	
NSlot	Nonnegative integer	Slot number
TransportBlockSize	Nonnegative integer	Size of PDSCH transport block
TransportBlock	Binary column vector	PDSCH transport block
RV	Nonnegative integer	Redundancy version
Codeword	Binary column vector	Codeword from DL-SCH transport channel
G	Nonnegative integer	Bit capacity of the PDSCH. This value is equal to the length of the codeword from the DL-SCH transport channel.
Gd	Nonnegative integer	Number of resource elements per layer or port
ChannelIndices	Column vector of positive integers	PDSCH indices
ChannelSymbols	Complex column vector	PDSCH symbols
DMRSIndices	Column vector of positive integers	PDSCH DM-RS indices

Fields	Values	Description
	Fields:	Values: Description:
		Fields: Values: Description:
		Fields: Values:
		DMRSSymbols Complex column vector: PDSCH DM-RS symbols
		DMRSSymbolSet Vector of nonnegative integers: OFDM symbol locations in a slot containing the DM-RS (0-based)
		PTRSIndices Column vector of positive integers: PDSCH phase tracking reference signal (PT-RS) indices
		PTRSSymbols Complex column vector: PDSCH PT-RS symbols
		PTRSSymbolSet Vector of nonnegative integers: OFDM symbol locations in a slot containing PT-RS (0-based)

Data Types: `struct`

See Also

Objects
`nrDLCarrierConfig`

Introduced in R2020b

write

Write protocol packet data to PCAP or PCAPNG file

Syntax

```
write(pcapObj,packet,timestamp)
write(pcapngObj,packet,timestamp,interfaceID)
write( ___ ,Name,Value)
```

Description

`write(pcapObj,packet,timestamp)` writes the protocol packet data to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `packet` specifies the protocol packet and input `timestamp` specifies the packet arrival time.

`write(pcapngObj,packet,timestamp,interfaceID)` writes protocol packet data to a PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `packet`, `timestamp`, and `interfaceID` specifies the protocol packet, packet arrival time, and interface identifier, respectively.

`write( ___ ,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument combinations from any of the previous syntaxes. For example, `'PacketFormat','bits'` sets the format of the protocol packets to bits.

Examples

Write 5G NR Packet Data to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file.

```
pcapObj = pcapWriter('FileName','write5GNRpacket');
timestamp = 300;                                                % Timestamp
```

5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
linkType = 113;
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,linkType);
```

5G NR packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                            % MAC subPDU (contains truncated buffe
radioType = 1;                                                  % Frequency division duplexing
linkDir = 0;                                                    % Uplink packet
rntiType = 3;                                                   % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];                      % Tag to indicate start of NR MAC sign
payloadTag = 1;                                                 % Payload tag for NR packets
```

```
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a user datagram protocol (UDP) header.

```
udpPacketLength = 8 + length(macNRInfoPacket);              % Length of header (8 bytes) and payl
udpHeader = [163; 76;                                       % Source port number
    39; 15;                                                 % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256);     % Total length of UDP packet
    0; 0];                                                  % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                      % Length of header (20 bytes) and pay
ipHeader = [69;                                             % Version of IP protocol and priority
    0;                                                      % Type of service
    fix(ipPacketLength/256); mod(ipPacketLength,256);       % Total length of the IPv4 packet
    0; 1;                                                   % Identification
    0; 0;                                                   % Flags and fragmentation offset
    64;                                                     % Time to live in seconds
    17;                                                     % UDP protocol number
    0; 0;                                                   % Header checksum
    127; 0; 0; 1;                                           % Source IP address
    127; 0; 0; 1];                                          % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0; 0;                                          % Packet type
    3; 4;                                                   % Address resolution protocol hardware
    0; 0;                                                   % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                                 % Link layer address
    8; 0];                                                  % Protocol type
```

Construct an 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet data to the PCAP file.

```
write(pcapObj,packet,timestamp);
```

Write 5G NR Packet Data to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','write5GNRpacket');
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';                                         % Interface name
linkType = 113;                                             % Link type of SLL packet
timestamp = 300;                                            % Timestamp
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

5G NR packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                        % MAC subPDU (contains short truncated
radioType = 1;                                              % Frequency division duplexing
linkDir = 0;                                                % Uplink packet
rntiType = 3;                                               % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];                  % Tag to indicate the start of NR MAC
payloadTag = 1;                                             % Payload tag for NR packets
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a UDP header.

```
udpPacketLength = 8 + length(macNRInfoPacket);              % Length of header (8 bytes) and payl
udpHeader = [163; 76;                                       % Source port number
    39; 15;                                                 % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256);     % Total length of UDP packet
    0; 0];                                                  % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                      % Length of header (20 bytes) and pay
ipHeader = [69;                                             % Version of IP protocol and Priority
    0;                                                      % Type of service
    fix(ipPacketLength/256);mod(ipPacketLength,256);        % Total length of the IPv4 packet
    0; 1;                                                   % Identification
    0; 0;                                                   % Flags and fragmentation offset
    64;                                                     % Time to live in seconds
    17;                                                     % UDP protocol number
    0; 0;                                                   % Header checksum
    127; 0; 0; 1;                                           % Source IP address
    127; 0; 0; 1];                                          % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0;0;                                           % Packet type
    3; 4;                                                   % Address resolution protocol hardware
    0; 0;                                                   % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                                 % Link layer address
    8; 0];                                                  % Protocol type
```

Construct an 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet data to the PCAPNG file.

```
packetComment = 'This is 5G NR MAC packet';                 % Packet comment
write(pcapngObj,packet,timestamp,interfaceID,'PacketComment',packetComment);
```

Input Arguments

Note The `pcapWriter` and `pcapngWriter` objects do not overwrite the existing PCAP or PCAPNG files, respectively. During each call of these objects, specify a unique PCAP or PCAPNG file name.

`pcapObj` — PCAP file writer object
`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`packet` — Protocol packet
binary-valued vector | character vector | string scalar | numeric vector | *n*-by-2 character array

Protocol packet, specified as one of these values.

- Binary-valued vector – This value represents bits.
- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Data Types: `char` | `string` | `double`

`timestamp` — Packet arrival time
nonnegative integer

Packet arrival time since 1/1/1970, specified as a nonnegative integer. This value must be expressed in microseconds.

Data Types: `double`

`pcapngObj` — PCAPNG file writer object
`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `'PacketFormat','bits'` specifies the format of the protocol packet as bits.

`PacketFormat` — Format of the protocol packet
`'octets'` (default) | `'bits'`

Format of the protocol packet, specified as the comma-separated pair consisting of `PacketFormat` and `'octets'` or `'bits'`. If this value is specified as `'octets'`, `packet` is specified as one of these values.

- Binary-valued vector – This value represents bits.

- Character vector – This value represents octets in hexadecimal format.
- String scalar – This value represents octets in hexadecimal format.
- Numeric vector with each element in the range [0, 255] – This value represents octets in decimal format.
- *n*-by-2 character array – In this value, each row represents an octet in hexadecimal format .

Data Types: `char` | `string` | `double`

`PacketComment` — Comment for protocol packet
`''` (default) | character vector | string scalar

Comment for the protocol packet, specified as the comma-separated pair consisting of `PacketComment` and a character vector or a string scalar.

Dependencies

To enable this name-value pair argument, specify the `pcapngObj` input argument.

Data Types: `char` | `string`

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org/.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`writeCustomBlock` | `writeGlobalHeader` | `writeInterfaceDescriptionBlock`

Objects
`pcapWriter` | `pcapngWriter`

Introduced in R2020b

writeCustomBlock

Write custom block to PCAPNG file

Syntax

`writeCustomBlock(pcapngObj,customData)`

Description

`writeCustomBlock(pcapngObj,customData)` writes the custom block data, `customData`, to a PCAPNG file specified in the PCAPNG file writer object.

Examples

Write 5G NR User-Defined Custom Block to PCAPNG File

Create a default PCAPNG file writer object.

```
pcapngObj = pcapngWriter;
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';         % Interface name
linkType = 113;             % Link type of SLL packet
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

Write the custom block to specify user-defined data.

```
writeCustomBlock(pcapngObj,"This block writes user-defined data");
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. During each call of this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`customData` — User-defined data

character vector | string scalar

User-defined data, specified as a character vector or a string scalar.

Data Types: `char` | `string`

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org/.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`write` | `writeInterfaceDescriptionBlock`

Objects

`pcapWriter` | `pcapngWriter`

Introduced in R2020b

writeGlobalHeader

Write global header to PCAP file

Syntax

```
writeGlobalHeader(pcapObj,linkType)
```

Description

`writeGlobalHeader(pcapObj,linkType)` writes a global header to the PCAP file specified in the PCAP file writer object, `pcapObj`. Input `linkType` specifies the unique identifier for a protocol.

Examples

Write Global Header for 5G NR Packet to PCAP File

Create a default PCAP file writer object.

```
pcapObj = pcapWriter;
```

5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
linkType = 113;
```

Use the PCAP file writer object and the link type of the SLL packet to write a global header for the 5G NR packet to the PCAP file.

```
writeGlobalHeader(pcapObj,linkType);
```

Input Arguments

Note The `pcapWriter` object does not overwrite the existing PCAP file. During each call of this object, specify a unique PCAP file name.

`pcapObj` — PCAP file writer object

`pcapWriter` object

PCAP file writer object, specified as a `pcapWriter` object.

`linkType` — Unique identifier for a protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65535].

Data Types: `double`

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org/.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`write`

Objects

`pcapWriter` | `pcapngWriter`

Introduced in R2020b

writeInterfaceDescriptionBlock

Write interface description block to PCAPNG file

Syntax

```
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)
```

Description

`interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface)` writes an interface description block to the PCAPNG file specified in the PCAPNG file writer object, `pcapngObj`. Input `linkType` specifies the unique identifier for the protocol and input `interface` specifies the interface on which the protocol packets are captured. This object function returns the unique identifier for the interface.

Examples

Write 5G NR Interface Description Block to PCAPNG File

Create a default PCAPNG file writer object.

```
pcapngObj = pcapngWriter;
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';       % Interface name
linkType = 113;           % Link type of SLL packet
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

Input Arguments

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. During each call of this object, specify a unique PCAPNG file name.

`pcapngObj` — PCAPNG file writer object

`pcapngWriter` object

PCAPNG file writer object, specified as a `pcapngWriter` object.

`linkType` — Unique identifier for protocol

integer in the range [0, 65,535].

Unique identifier for a protocol, specified as an integer in the range [0, 65,535].

Data Types: `double`

`interface` — Name of the interface on which protocol packets are captured
character vector | string scalar

Name of the interface on which protocol packets are captured, specified as a character vector or a string scalar in 8-bit unicode transformation format (UTF-8) format.

Data Types: `char` | `string`

Output Arguments

`interfaceID` — Unique identifier for an interface
nonnegative scalar

Unique identifier for an interface, specified as a nonnegative scalar.

Data Types: `double`

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org/.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`write` | `writeCustomBlock`

Objects
`pcapWriter` | `pcapngWriter`

Introduced in R2020b

System Objects

nrCDLChannel

Send signal through CDL channel model

Description

The `nrCDLChannel` System object sends an input signal through a clustered delay line (CDL) multi-input multi-output (MIMO) link-level fading channel to obtain the channel-impaired signal. The object implements the following aspects of TR 38.901 [1]:

- Section 7.7.1: CDL models
- Section 7.7.3: Scaling of delays
- Section 7.7.5.1: Scaling of angles
- Section 7.7.6: K-factor for LOS channel models

To send a signal through the CDL MIMO channel model:

1. Create the `nrCDLChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
cdl = nrCDLChannel
cdl = nrCDLChannel(Name,Value)
```

Description

`cdl = nrCDLChannel` creates a CDL MIMO channel System object.

`cdl = nrCDLChannel(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: `cdl = nrCDLChannel('DelayProfile','CDL-D','DelaySpread',2e-6)` creates the channel object with CDL-D delay profile and 2 microseconds delay spread.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`DelayProfile` — CDL delay profile

`'CDL-A'` (default) | `'CDL-B'` | `'CDL-C'` | `'CDL-D'` | `'CDL-E'` | `'Custom'`

CDL delay profile, specified as `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, `'CDL-E'`, or `'Custom'`. See TR 38.901 Section 7.7.1, Tables 7.7.1-1 to 7.7.1-5.

When you set this property to `'Custom'`, configure the delay profile using properties `PathDelays`, `AveragePathGains`, `AnglesAoA`, `AnglesAoD`, `AnglesZoA`, `AnglesZoD`, `HasLOSCluster`, `KFactorFirstCluster`, `AngleSpreads`, `XPR`, and `NumStrongestClusters`.

Data Types: `char` | `string`

`PathDelays` — Discrete path delays in seconds

`0.0` (default) | numeric scalar | row vector

Discrete path delays in seconds, specified as a numeric scalar or row vector. `AveragePathGains` and `PathDelays` must have the same size.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AveragePathGains` — Average path gains in dB

`0.0` (default) | numeric scalar | row vector

Average path gains in dB, specified as a numeric scalar or row vector. `AveragePathGains` and `PathDelays` must have the same size.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AnglesAoA` — Azimuth of arrival angle in degrees

`0.0` (default) | numeric scalar | row vector

Azimuth of arrival angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AnglesAoD` — Azimuth of departure angle in degrees

`0.0` (default) | numeric scalar | row vector

Azimuth of departure angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AnglesZoA` — Zenith of arrival angle in degrees

`0.0` (default) | numeric scalar | row vector

Zenith of arrival angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AnglesZoD` — Zenith of departure angle in degrees

`0.0` (default) | numeric scalar | row vector

Zenith of departure angle in degrees, specified as a numeric scalar or row vector. The vector elements specify the angles for each cluster.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`HasLOSCluster` — Line of sight cluster of the delay profile

`false` (default) | `true`

Line of sight (LOS) cluster of the delay profile, specified as `false` or `true`. The `PathDelays`, `AveragePathGains`, `AnglesAoA`, `AnglesAoD`, `AnglesZoA`, and `AnglesZoD` properties define the delay profile. To enable the LOS cluster of the delay profile, set `HasLOSCluster` to `true`.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `logical`

`KFactorFirstCluster` — K-factor in first cluster of delay profile in dB

`13.3` (default) | numeric scalar

K-factor in the first cluster of the delay profile in dB, specified as a numeric scalar. The default value corresponds to the K-factor in the first cluster of CDL-D as defined in TR 38.901 Section 7.7.1, Table 7.7.1-4.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` and `HasLOSCluster` to `true`.

Data Types: `double`

`AngleScaling` — Apply scaling of angles

`false` (default) | `true`

Apply scaling of angles, specified as `false` or `true` according to TR 38.901 Section 7.7.5.1. When set to `true`, the `AngleSpreads` and `MeanAngles` properties define the scaling of angles.

Dependencies

To enable this property, set `DelayProfile` to `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`. This property does not apply for custom delay profile.

Data Types: `logical`

`AngleSpreads` — Scaled or cluster-wise RMS angle spreads in degrees

`[5.0 11.0 3.0 3.0]` (default) | four-element row vector

Scaled or cluster-wise root mean square (RMS) angle spreads in degrees, specified as a four-element row vector in one of these forms:

- [*ASD ASA ZSD ZSA*] — Use this vector to specify the desired RMS angle spreads of the channel, as described in TR 38.901 Section 7.7.5.1 ($AS_{desired}$), where:
  - *ASD* is the RMS azimuth spread of departure angles
  - *ASA* is the RMS azimuth spread of arrival angles
  - *ZSD* is the RMS zenith spread of departure angles
  - *ZSA* is the RMS zenith spread of arrival angles

  To use this form, set `AngleScaling` to `true` and `DelayProfile` to `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`.
- [C_{ASD} C_{ASA} C_{ZSD} C_{ZSA}] — Use this vector to specify cluster-wise RMS angle spreads for scaling ray offset angles within a cluster, as described in TR 38.901 Section 7.7.1, Step1, where:
  - C_{ASD} is the cluster-wise RMS azimuth spread of departure angles
  - C_{ASA} is the cluster-wise RMS azimuth spread of arrival angles
  - C_{ZSD} is the cluster-wise RMS zenith spread of departure angles
  - C_{ZSA} is the cluster-wise RMS zenith spread of arrival angles

  To use this form, set `DelayProfile` to `'Custom'`. Based on TR 38.901 Section 7.7.5.1, the object does not perform angle scaling in this case.

The default value corresponds to the default cluster-wise angle spreads of CDL-A as defined in TR 38.901 Section 7.7.1 Table 7.7.1-1.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` or `AngleScaling` to `true`.

Data Types: `double`

`MeanAngles` — Scaled mean angles in degrees

`[0.0 0.0 0.0 0.0]` (default) | four-element row vector

Scaled mean angles in degrees, specified as a four-element row vector of the form [*AoD AoA ZoD ZoA*].

- *AoD* is the mean azimuth of departure angles after scaling
- *AoA* is the mean azimuth of arrival angles after scaling

- *ZoD* is the mean zenith of departure angles after scaling
- *ZoA* is the mean zenith of arrival angles after scaling

Use this vector to specify the desired mean angles of the channel used for angle scaling, as described in TR 38.901 Section 7.7.5.1 ($\mu_{\Phi,\text{desired}}$).

Dependencies

To enable this property, set `AngleScaling` to `true`.

Data Types: `double`

XPR — Cross-polarization power ratio in dB

`10.0` (default) | numeric scalar

Cross-polarization power ratio in dB, specified as a numeric scalar. The default value corresponds to the cluster-wise cross-polarization power ratio of CDL-A as defined in TR 38.901 Section 7.7.1, Table 7.7.1-1.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`DelaySpread` — Desired RMS delay spread in seconds

`30e-9` (default) | numeric scalar

Desired RMS delay spread in seconds, specified as a numeric scalar. For examples of desired RMS delay spreads, $DS_{desired}$, see TR 38.901 Section 7.7.3 and Tables 7.7.3-1 and 7.7.3-2.

Dependencies

To enable this property, set `DelayProfile` to `'CDL-A'`, `'CDL-B'`, `'CDL-C'`, `'CDL-D'`, or `'CDL-E'`. This property does not apply for custom delay profile.

Data Types: `double`

`CarrierFrequency` — Carrier frequency in Hz

`4e9` (default) | numeric scalar

Carrier frequency in Hz, specified as a numeric scalar.

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift in Hz

`5` (default) | nonnegative numeric scalar

Maximum Doppler shift in Hz, specified as a nonnegative numeric scalar. This property applies to all channel paths. When the maximum Doppler shift is set to 0, the channel remains static for the entire input. To generate a new channel realization, reset the object by calling the `reset` function.

Data Types: `double`

`UTDirectionOfTravel` — User terminal direction of travel in degrees

`[0; 90]` (default) | two-element column vector

User terminal (or user equipment) direction of travel in degrees, specified as a two-element column vector. The vector elements specify the azimuth and the elevation components [azimuth; elevation].

Data Types: `double`

`KFactorScaling` — K-factor scaling

`false` (default) | `true`

K-factor scaling, specified as `false` or `true`. When set to `true`, the `KFactor` property specifies the desired K-factor and the object applies K-factor scaling as described in TR 38.901 Section 7.7.6.

Note K-factor scaling modifies both the path delays and path powers.

Dependencies

To enable this property, set `DelayProfile` to `'CDL-D'` or `'CDL-E'`.

Data Types: `double`

`KFactor` — Desired K-factor for scaling in dB

`9.0` (default) | numeric scalar

Desired K-factor for scaling in dB, specified as a numeric scalar. For typical K-factor values, see TR 38.901 Section 7.7.6 and Table 7.5-6.

Note

- K-factor scaling modifies both the path delays and path powers.
- `K-factor` applies to the overall delay profile. Specifically, the K-factor after the scaling is K_{model} as described in TR 38.901 Section 7.7.6. K_{model} is the ratio of the power of the first path LOS to the total power of all the Laplacian clusters, including the Laplacian part of the first cluster.

Dependencies

To enable this property, set `KFactorScaling` to `true`.

Data Types: `double`

`SampleRate` — Sample rate of input signal in Hz

`30.72e6` (default) | positive numeric scalar

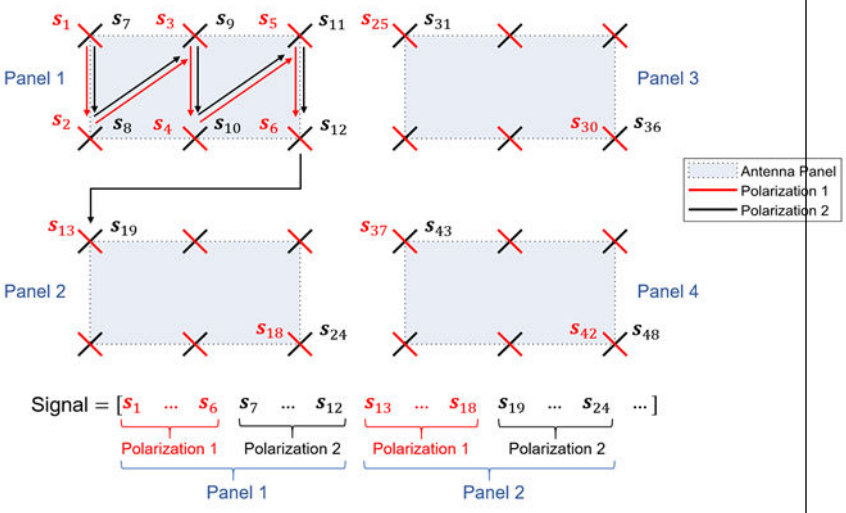
Sample rate of input signal in Hz, specified as a positive numeric scalar.

Data Types: `double`

`TransmitAntennaArray` — Transmit antenna array characteristics

structure

Transmit antenna array characteristics, specified as a structure that contains these fields:

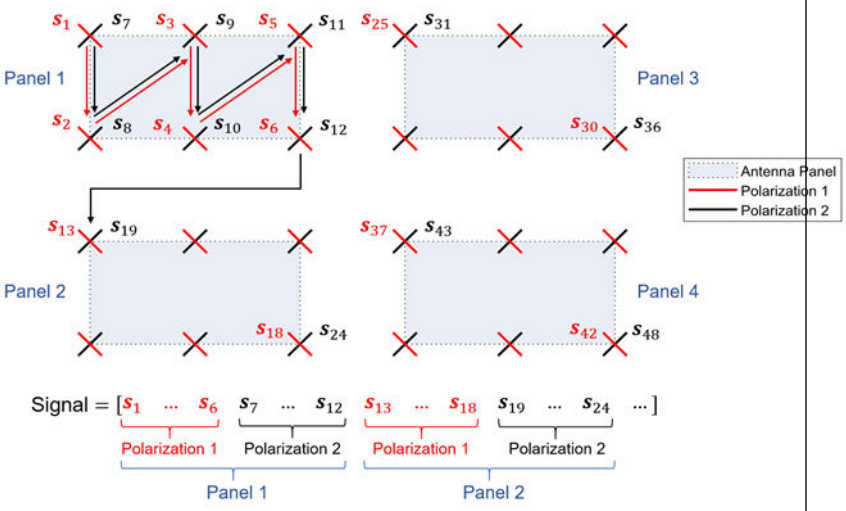
Parameter Field	Values	Description
`Size`	`[2 2 2 1 1]` (default), row vector	Size of antenna array $[M\ N\ P\ M_g\ N_g]$, where: • M and N are the number of rows and columns in the antenna array, respectively. • P is the number of polarizations (1 or 2). • M_g and N_g are the number of row and column array panels, respectively. The `nrCDLChannel` System object maps the input signal `signalIn` to the antenna array elements panel-wise, in the order that a 5-D array of size M-by-N-by-P-by-M_g-by-N_g is linearly indexed across the first dimension to the last. For example, this figure shows how the object maps the input signal `signalIn` to an antenna array of size `[2 3 2 2 2]`. The antenna array consists of 2-by-2 antenna panels of 2-by-3 elements with 2 polarizations. The object maps the first $M = 2$ columns of the input signal (s_1 and s_2) to the first column of antenna elements with the first polarization angle of the first panel. The next $M = 2$ columns of the input signal (s_3 and s_4) are mapped to the next column of antenna elements, and so on. Following this pattern, the object maps the first $M \times N = 6$ columns of the input signal (s_1 to s_6) to the antenna elements with the first polarization angle of the complete first panel. Similarly, the next 6 columns of the input signal (s_7 to s_{12}) are mapped to the antenna elements with the second polarization angle of the first panel. Subsequent sets of $M \times N \times P = 12$ columns of the input signal (s_{13} to s_{24}, s_{25} to s_{36}, s_{37} to s_{48}) are mapped to consecutive panels, taking panel rows first, then panel columns. 

Parameter Field	Values	Description
ElementSpacing	`[0.5 0.5 1.0 1.0]` (default), row vector	Element spacing, in wavelengths, specified as a row vector of the form $[\lambda_v\ \lambda_h\ dg_v\ dg_h]$. The vector elements represent the vertical and horizontal element spacing and the vertical and horizontal panel spacing, respectively.
PolarizationAngles	`[45 -45]` (default), row vector	Polarization angles in degrees, specified as a row vector of the form $[\theta\ \rho]$.
Orientation	`[0; 0; 0]` (default), column vector	Mechanical orientation of the array, in degrees, specified as a column vector of the form $[\alpha; \beta; \gamma]$. The vector elements specify the bearing, downtilt, and slant, respectively. The default value indicates that the broadside direction of the array points to the positive x-axis.
Element	`'38.901'` (default), `'isotropic'`	Antenna element radiation pattern as described in TR 38.901 Section 7.3. (Note that TR 38.901 superseded TR 38.900.)
PolarizationModel	`'Model-2'` (default), `'Model-1'`	Model that determines the radiation field patterns based on a defined radiation power pattern. For more information, see TR 38.901 Section 7.3.2.

Data Types: `struct`

ReceiveAntennaArray — Receive antenna array characteristics
structure

Receive antenna array characteristics, specified as a structure that contains these fields:

Parameter Field	Values	Description
Size	`[1 1 2 1 1]` (default), row vector	Size of antenna array $[M\ N\ P\ M_g\ N_g]$, where: • M and N are the number of rows and columns in the antenna array, respectively. • P is the number of polarizations (1 or 2). • M_g and N_g are the number of row and column array panels, respectively. The `nrCDLChannel` System object maps the antenna array elements to the output signal `signalOut` panel-wise, in the order that a 5-D array of size M-by-N-by-P-by-M_g-by-N_g is linearly indexed across the first dimension to the last. For example, this figure shows how the object maps an antenna array of size `[2 3 2 2 2]` to the output signal `signalOut`. The antenna array consists of 2-by-2 antenna panels of 2-by-3 elements with 2 polarizations. The first column of antenna elements with the first polarization angle of the first panel are mapped to the first M = 2 columns of the output signal (s_1 and s_2). The next column of antenna elements are mapped to the next M = 2 columns of the output signal (s_3 and s_4), and so on. Following this pattern, the object maps the antenna elements with the first polarization angle of the complete first panel to the first $M \times N$ = 6 columns of the output signal (s_1 to s_6). Similarly, the antenna elements with the second polarization angle of the first panel are mapped to the next 6 columns of the output signal (s_7 to s_{12}). Consecutive panels are mapped to subsequent sets of $M \times N \times P$ = 12 columns of the output signal (s_{13} to s_{24}, s_{25} to s_{36}, s_{37} to s_{48}), taking panel rows first, then panel columns. 

Parameter Field	Values	Description
`ElementSpacing`	`[0.5 0.5 0.5 0.5]` (default), row vector	Element spacing, in wavelengths, specified as a row vector of the form $[\lambda_v\ \lambda_h\ dg_v\ dg_h]$. The vector elements represent the vertical and horizontal element spacing and the vertical and horizontal panel spacing, respectively.
`PolarizationAngles`	`[0 90]` (default), row vector	Polarization angles in degrees, specified as a row vector of the form $[\theta\ \rho]$.
`Orientation`	`[0; 0; 0]` (default), column vector	Mechanical orientation of the array, in degrees, specified as a column vector of the form $[\alpha; \beta; \gamma]$. The vector elements specify the bearing, downtilt, and slant, respectively. The default value indicates that the broadside direction of the array points to the positive x-axis.
`Element`	`'isotropic'` (default), `'38.901'`	Antenna element radiation pattern as described in TR 38.901 Section 7.3. (Note that TR 38.901 superseded TR 38.900.)
`PolarizationModel`	`'Model-2'` (default), `'Model-1'`	Model that determines the radiation field patterns based on a defined radiation power pattern. For more information, see TR 38.901 Section 7.3.2.

Data Types: `struct`

`SampleDensity` — Number of time samples per half wavelength
`64` (default) | `Inf` | numeric scalar

Number of time samples per half wavelength, specified as `Inf` or a numeric scalar. The `SampleDensity` and `MaximumDopplerShift` properties control the coefficient generation sampling rate, *Fcg*, given by

Fcg = `MaximumDopplerShift` × 2 × `SampleDensity`.

Setting `SampleDensity` to `Inf` assigns *Fcg* the value of the `SampleRate` property.

Data Types: `double`

`NormalizePathGains` — Normalize path gains
`true` (default) | `false`

Normalize path gains, specified as `true` or `false`. Use this property to normalize the fading processes. When this property is set to `true`, the total power of the path gains, averaged over time, is 0 dB. When this property is set to `false`, the path gains are not normalized. The average powers of the path gains are specified by the selected delay profile, or if `DelayProfile` is set to `'Custom'`, by the `AveragePathGains` property.

Data Types: `logical`

`InitialTime` — Time offset of fading process in seconds
`0.0` (default) | numeric scalar

Time offset of fading process in seconds, specified as a numeric scalar.

Tunable: Yes

Data Types: `double`

`NumStrongestClusters` — Number of strongest clusters to split into subclusters

`0` (default) | numeric scalar

Number of strongest clusters to split into subclusters, specified as a numeric scalar. See TR 38.901 Section 7.5, Step 11.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`ClusterDelaySpread` — Cluster delay spread in seconds

`3.90625e-9` (default) | nonnegative scalar

Cluster delay spread in seconds, specified as a nonnegative scalar. Use this property to specify the delay offset between subclusters for clusters split into subclusters. See TR 38.901 Section 7.5, Step 11.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` and `NumStrongestClusters` to a value greater than zero.

Data Types: `double`

`RandomStream` — Source of random number stream

`'mt19937ar with seed'` (default) | `'Global stream'`

Source of random number stream, specified as one of the following:

- `'mt19937ar with seed'` — The object uses the mt19937ar algorithm for normally distributed random number generation. Calling the `reset` function resets the filters and reinitializes the random number stream to the value of the `Seed` property.
- `'Global stream'` — The object uses the current global random number stream for normally distributed random number generation. Calling the `reset` function resets only the filters.

`Seed` — Initial seed of mt19937ar random number stream

`73` (default) | nonnegative numeric scalar

Initial seed of mt19937ar random number stream, specified as a nonnegative numeric scalar.

Dependencies

To enable this property, set RandomStream to `'mt19937ar with seed'`. When calling the `reset` function, the seed reinitializes the mt19937ar random number stream.

Data Types: `double`

`ChannelFiltering` — Fading channel filtering

`true` (default) | `false`

Fading channel filtering, specified as `true` or `false`. When this property is set to `false`, the following conditions apply:

- The object takes no input signal and returns only the path gains and sample times.
- The `SampleDensity` property determines when to sample the channel coefficients.
- The `NumTimeSamples` property controls the duration of the fading process realization at a sampling rate given by the `SampleRate` property.

Data Types: `logical`

`NumTimeSamples` — Number of time samples
`30720` (default) | positive integer

Number of time samples, specified as a positive integer. Use this property to set the duration of the fading process realization.

Tunable: Yes

Dependencies

To enable this property, set `ChannelFiltering` to `false`.

Data Types: `double`

`NormalizeChannelOutputs` — Normalize channel outputs by the number of receive antennas
`true` (default) | `false`

Normalize channel outputs by the number of receive antennas, specified as `true` or `false`.

Dependencies

To enable this property, set `ChannelFiltering` to `true`.

Data Types: `logical`

Usage

Syntax

```
signalOut = cdl(signalIn)
[signalOut,pathGains] = cdl(signalIn)
[signalOut,pathGains,sampleTimes] = cdl(signalIn)

[pathGains,sampleTimes] = cdl()
```

Description

`signalOut = cdl(signalIn)` sends the input signal through a CDL MIMO fading channel and returns the channel-impaired signal.

`[signalOut,pathGains] = cdl(signalIn)` also returns the MIMO channel path gains of the underlying fading process.

`[signalOut,pathGains,sampleTimes] = cdl(signalIn)` also returns the sample times of the channel snapshots of `pathGains` (first-dimension elements).

`[pathGains,sampleTimes] = cdl()` returns only the path gains and the sample times. In this case, the `NumTimeSamples` property determines the duration of the fading process. The object acts as a source of the path gains and sample times without filtering an input signal.

To use this syntax, you must set the `ChannelFiltering` property of `cdl` to `false`.

Input Arguments

`signalIn` — Input signal
complex scalar | vector | N_S-by-N_T matrix

Input signal, specified as a complex scalar, vector, or N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`signalOut` — Output signal
complex scalar | vector | N_S-by-N_R matrix

Output signal, returned as a complex scalar, vector, or N_S-by-N_R matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas.

The output signal data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathGains` — MIMO channel path gains of fading process
N_{CS}-by-N_P-by-N_T-by-N_R complex matrix

MIMO channel path gains of the fading process, returned as an N_{CS}-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_{CS} is the number of channel snapshots, controlled by the `SampleDensity` property of `cdl`.
- N_P is the number of paths, specified by the size of the `PathDelays` property of `cdl`.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

The path gains data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampleTimes` — Sample times of channel snapshots
N_{CS}-by-1 column vector

Sample times of channel snapshots, returned as an N_{CS}-by-1 column vector, where N_{CS} is the number of channel snapshots controlled by the `SampleDensity` property.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrCDLChannel`

info	Get characteristic information about link-level MIMO fading channel
getPathFilters	Get path filter impulse response for link-level MIMO fading channel
displayChannel	Visualize and explore CDL channel model characteristics

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Transmission Over Channel Model with Delay Profile CDL-D

Transmit waveform through 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 UE velocity of 15 km/h:

```
v = 15.0;                    % UE 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;     % UE 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 a vector of the form [*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 two polarization angles (*P*=2). Configure the receive antenna array as a vector of the form [*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 = cdl(txWaveform);
```

Plot Channel Transmission Properties with SISO and Delay Profile CDL-B

Plot channel output and path gain snapshots for various sample density values while using an `nrCDLChannel` System object.

Configure a channel with delay profile CDL-B from TR 38.901 Section 7.7.1. Set the maximum Doppler shift to 300 Hz and the channel sampling rate to 10 kHz.

```
cdl = nrCDLChannel;
cdl.DelayProfile = 'CDL-B';
cdl.MaximumDopplerShift = 300.0;
cdl.SampleRate = 10e3;
cdl.Seed = 19;
```

Configure the transmit and receive antenna arrays for single-input/single-output (SISO) operation.

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

Create an input waveform with a length of 40 samples.

```
T = 40;
in = ones(T,1);
```

Plot the step response of the channel (displayed as lines) and the corresponding path gain snapshots (displayed circles) for various values of the `SampleDensity` property. The sample density property controls how often the channel snapshots are taken relative to the Doppler frequency.

- When `SampleDensity` is set to `Inf`, a channel snapshot is taken for every input sample.
- When `SampleDensity` is set to a scalar S, a channel snapshot is taken at a rate of $F_{CS} = 2S \times MaximumDopplerShift$.

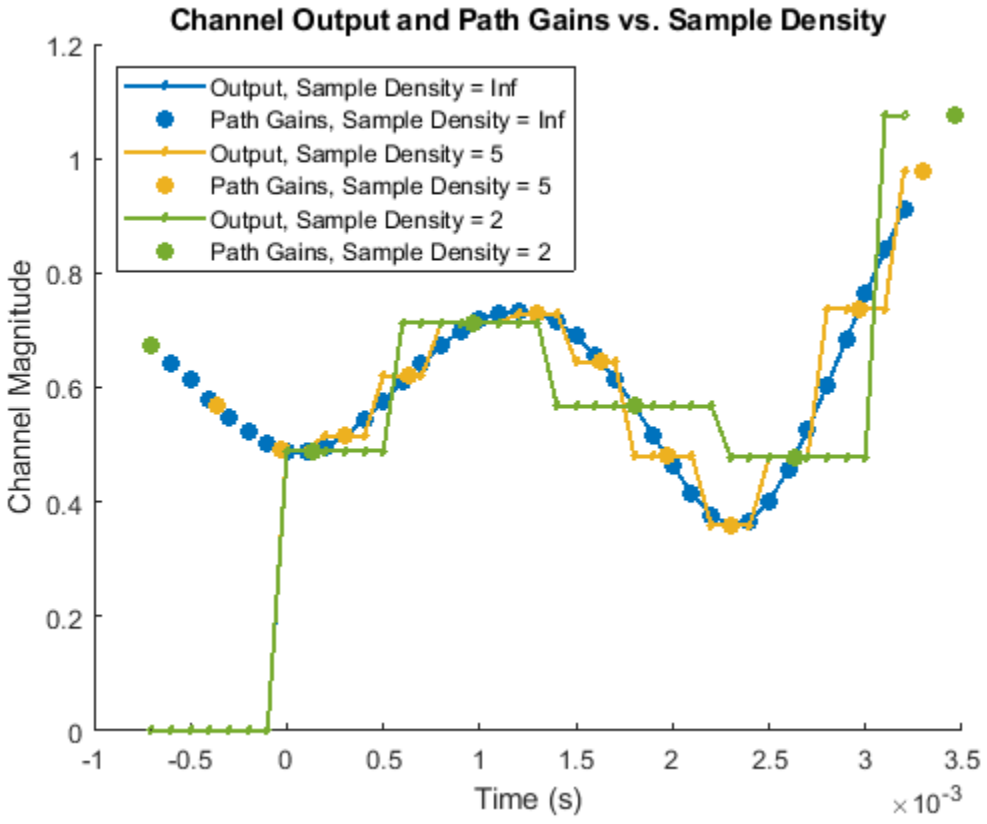
The `nrCDLChannel` object applies the channel snapshots to the input waveform by means of zero-order hold interpolation. The object takes an extra snapshot beyond the end of the input. Some of the final output samples use this extra value to minimize the interpolation error. The channel output contains a transient (and a delay) due to the filters that implement the path delays.

```
s = [Inf 5 2]; % sample densities

legends = {};
figure; hold on;
SR = cdl.SampleRate;
```

```
for i = 1:length(s)

    % call channel with chosen sample density
    release(cdl); cdl.SampleDensity = s(i);
    [out,pathgains,sampletimes] = cdl(in);
    chInfo = info(cdl); tau = chInfo.ChannelFilterDelay;

    % plot channel output against time
    t = cdl.InitialTime + ((0:(T-1)) - tau).' / SR;
    h = plot(t,abs(out),'o-');
    h.MarkerSize = 2;
    h.LineWidth = 1.5;
    desc = ['Sample Density = ' num2str(s(i))];
    legends = [legends ['Output, ' desc]];
    disp([desc ', Ncs = ' num2str(length(sampletimes))]);

    % plot path gains against sample times
    h2 = plot(sampletimes-tau/SR,abs(sum(pathgains,2)),'o');
    h2.Color = h.Color;
    h2.MarkerFaceColor = h.Color;
    legends = [legends ['Path Gains, ' desc]];
end
```

```
Sample Density = Inf, Ncs = 40

Sample Density = 5, Ncs = 13

Sample Density = 2, Ncs = 6
```

```
xlabel('Time (s)');
title('Channel Output and Path Gains vs. Sample Density');
ylabel('Channel Magnitude');
legend(legends,'Location','NorthWest');
```



References

[1] 3GPP TR 38.901. "Study on channel model for frequencies from 0.5 to 100 GHz." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Functions

`nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

Objects

`comm.MIMOChannel` | `nrTDLChannel`

Topics

"Visualize CDL Channel Model Characteristics"

Introduced in R2018b

nrDLSCH

Apply DL-SCH encoder processing chain

Description

The `nrDLSCH` System object applies the downlink shared channel (DL-SCH) encoder processing chain to one or two transport blocks. The DL-SCH encoding process consists of cyclic redundancy check (CRC), code block segmentation and CRC, low-density parity-check (LDPC) encoding, rate matching, and code block concatenation. The System object implements TS 38.212 Section 7.2 [1].

To apply the DL-SCH encoder processing chain:

1. Create the `nrDLSCH` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
encDL = nrDLSCH
encDL = nrDLSCH(Name,Value)
```

Description

`encDL = nrDLSCH` creates a DL-SCH encoder System object.

`encDL = nrDLSCH(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrDLSCH('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes
`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. In both cases, to enable retransmissions when a failure occurs, the object buffers the input data.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | numeric scalar | 1-by-2 numeric vector

Target code rate, specified as a numeric scalar or a 1-by-2 numeric vector. The values must be in the interval (0, 1). The default value corresponds to 526/1024. If you specify `TargetCodeRate` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify different target code rates for each transport block, specify `TargetCodeRate` as a vector.

Tunable: Yes

Data Types: `double`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate matching, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Data Types: `double`

Usage

Syntax

```
codedBits = encDL(mod,nLayers,outlen,rv)
codedBits = encDL( ___ ,harqID)
```

Description

`codedBits = encDL(mod,nLayers,outlen,rv)` applies the DL-SCH encoder processing chain to one or two transport blocks. The object returns encoded, rate-matched, and concatenated code blocks as one or two codewords of length `outlen`. Before you call this object, you must load the transport blocks into the object by using the `setTransportBlock` object function. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`codedBits = encDL( ___ ,harqID)` specifies the HARQ process number `harqID` used with the current transmission in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

When processing two transport blocks, specify the same HARQ process number for each transport block when calling the `setTransportBlock` function.

Input Arguments

`mod` — Modulation scheme
`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, a string array, or cell array of character vectors. This modulation scheme determines the modulation type and number of bits used per modulation symbol. For two transport blocks, the modulation scheme applies to both blocks. Alternatively, you can specify different modulation schemes for each transport block by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers
integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For `nLayers` > 4, the object expects two transports blocks.

Data Types: `double`

`outlen` — Output codeword length
nonnegative integer | 1-by-2 integer vector

Output codeword length, in bits, specified as a nonnegative integer or a 1-by-2 integer vector. If you specify `outlen` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify a different codeword length for each transport block, specify `outlen` as a vector.

The actual output length is a multiple of the product of the number of bits per symbol and the number of transmission layers. For example, for 64-QAM and 1 transmission layer, if you specify 16 for `outlen`, the actual output length is 6x1x3 = 18.

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3 | 1-by-2 integer vector

Redundancy version, specified as an integer from 0 to 3 or a 1-by-2 integer vector. If you specify `rv` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify a different redundancy version for each transport block, specify `rv` as a vector.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`codedBits` — One or two DL-SCH codewords
binary column vector | cell array of two binary column vectors

One or two DL-SCH codewords, returned as a binary column vector or a cell array of two binary column vectors. A codeword is the encoded, rate-matched, and concatenated code blocks obtained by processing one transport block. Specify the length of the codewords by using the `outlen` input argument.

Data Types: `int8`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrDLSCH`

getTransportBlock	Get transport block from UL-SCH or DL-SCH encoder
setTransportBlock	Load transport block into UL-SCH or DL-SCH encoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect DL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encDL = nrDLSCH;
encDL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the DL-SCH encoder.

```
setTransportBlock(encDL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the DL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
```

```
rv = 0;
codedTrBlock = encDL(mod,nLayers,outlen,rv);
```

Create and configure a DL-SCH decoder System object.

```
decDL = nrDLSCHDecoder;
decDL.TargetCodeRate = targetCodeRate;
decDL.TransportBlockLength = trBlkLen;
```

Call the DL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decDL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮


blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

DL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object for use with multiple HARQ processes.

```
encDL = nrDLSCH;
encDL.MultipleHARQProcesses = true;
```

Load transport block `trBlk` for transport block number 0 into the DL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
trBlkID = 0;
setTransportBlock(encDL,trBlk,trBlkID,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the DL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encDL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`nrDLSCHDecoder` | `nrULSCH`

Functions

`nrDLSCHInfo` | `nrPDSCH`

Topics

“NR PDSCH Throughput”
“5G NR Downlink Carrier Waveform Generation”

Introduced in R2019a

nrDLSCHDecoder

Apply DL-SCH decoder processing chain

Description

The `nrDLSCHDecoder` System object applies the downlink shared channel (DL-SCH) decoder processing chain to the soft bits corresponding to one or two DL-SCH-encoded transport blocks. The DL-SCH decoding process consists of rate recovery, low-density parity-check (LDPC) decoding, desegmentation, and cyclic redundancy check (CRC) decoding. The object implements the inverse operation of the DL-SCH encoding process specified in TS 38.212 Section 7.2 [1].

To apply the DL-SCH decoder processing chain:

1. Create the `nrDLSCHDecoder` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
decDL = nrDLSCHDecoder
decDL = nrDLSCHDecoder(Name,Value)
```

Description

`decDL = nrDLSCHDecoder` creates a DL-SCH decoder System object.

`decDL = nrDLSCHDecoder(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrDLSCHDecoder('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes
`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. To enable soft combining of retransmissions before LDPC decoding, the object maintains a soft buffer for each HARQ process.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | numeric scalar | 1-by-2 numeric vector

Target code rate, specified as a numeric scalar or a 1-by-2 numeric vector. The values must be in the interval (0, 1). The default value corresponds to 526/1024. If you specify `TargetCodeRate` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify different target code rates for each transport block, specify `TargetCodeRate` as a vector.

Tunable: Yes

Data Types: `double`

`TransportBlockLength` — Length of decoded transport block
`5120` (default) | positive scalar integer | 1-by-2 integer vector

Length of decoded transport block, or transport blocks, in bits, specified as a positive scalar integer or a 1-by-2 integer vector. If you specify `TransportBlockLength` as a scalar, the object applies scalar expansion when processing two transport blocks. To specify a different length for the decoded transport blocks, specify `TransportBlockLength` as a vector.

Tunable: Yes

Data Types: `double`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate recovery, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum LDPC decoding iterations
`12` (default) | positive integer

Maximum LDPC decoding iterations, specified as a positive integer. Since early termination is enabled, decoding stops once parity-checks are satisfied. In this case, fewer iterations take place than the maximum specified by this argument.

Data Types: `double`

`LDPCDecodingAlgorithm` — LDPC decoding algorithm
`'Belief propagation'` (default) | `'Layered belief propagation'` | `'Normalized min-sum'` | `'Offset min-sum'`

LDPC decoding algorithm, specified as one of these values:

- `'Belief propagation'` — Use this option to specify the belief-passing or message-passing algorithm.

- `'Layered belief propagation'` — Use this option to specify the layered belief-passing algorithm, which is suitable for quasi-cyclic parity-check matrices (PCMs).
- `'Normalized min-sum'` — Use this option to specify the layered belief propagation algorithm with normalized min-sum approximation.
- `'Offset min-sum'` — Use this option to specify the layered belief propagation algorithm with offset min-sum approximation.

For more information on these algorithms, see LDPC Decoding Algorithms on page 2-31.

Data Types: `char` | `string`

`ScalingFactor` — Scaling factor for normalized min-sum decoding
`0.75` (default) | real scalar in the range (0, 1]

Scaling factor for normalized min-sum decoding, specified as a real scalar in the range (0, 1].

Dependencies

To enable this property, set the `LDPCDecodingAlgorithm` property to `'Normalized min-sum'`.

Data Types: `double`

`Offset` — Offset for offset min-sum decoding
`0.5` (default) | nonnegative finite real scalar

Offset for offset min-sum decoding, specified as a nonnegative finite real scalar.

Dependencies

To enable this property, set the `LDPCDecodingAlgorithm` property to `'Offset min-sum'`.

Data Types: `double`

Usage

Syntax

```
trblk = decDL(softbits,mod,nLayers,rv)
trblk = decDL(___,harqID)
[trblk,blkerr] = decDL(___)
```

Description

`trblk = decDL(softbits,mod,nLayers,rv)` applies the DL-SCH decoder processing chain to the input `softbits` and returns the decoded bits. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`trblk = decDL(___,harqID)` specifies the HARQ process number `harqID` used with the current transmission in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

When the object receives codewords with different redundancy version for an individual HARQ process, the object uses soft buffer state retention to enable soft combining of retransmissions. When you enable multiple HARQ processes, the object maintains independent buffers for each process.

`[trblk,blkerr] = decDL( ___ )` returns an error flag, using the input arguments in any of the previous syntaxes. A value of 1 in `blkerr` indicates an error during transport block decoding.

Input Arguments

`softbits` — Approximate LLR soft bits

real column vector | cell array of two real column vectors

Approximate log-likelihood ratio (LLR) soft bits, corresponding to one or two DL-SCH-encoded transport blocks, specified as a real column vector or a cell array of two real column vectors.

Data Types: `single` | `double`

`mod` — Modulation scheme

`'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, `'256QAM'`, a string array, or cell array of character vectors. This modulation scheme determines the modulation type and number of bits used per modulation symbol. For two transport blocks, the modulation scheme applies to both blocks. Alternatively, you can specify different modulation schemes for each transport block by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers

integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8. For `nLayers` > 4, the object expects two encoded transports blocks as input.

Data Types: `double`

`rv` — Redundancy version

integer from 0 to 3 | 1-by-2 integer vector

Redundancy version, specified as an integer from 0 to 3 or a 1-by-2 integer vector. If you specify `rv` as a scalar, the object applies scalar expansion when processing two encoded transport blocks. To specify a different redundancy version for each encoded transport block, specify `rv` as a vector.

Data Types: `double`

`harqID` — HARQ process number

integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`trblk` — Decoded DL-SCH transport blocks
binary column vector | cell array of two binary column vectors

Decoded DL-SCH transport blocks, returned as a binary column vector or cell array of two binary column vectors. The `TransportBlockLength` property specifies the length of the column vectors.

`blkerr` — Result of DL-SCH transport block decoding
logical scalar | logical vector

Result of DL-SCH transport block decoding for each transport block, returned as a logical scalar or logical vector of length 2. A value of `1` in `blkerr` indicates an error during transport block decoding.

Data Types: `logical`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrDLSCHDecoder`

resetSoftBuffer	Reset soft buffer for HARQ process in UL-SCH or DL-SCH decoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect DL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure a DL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encDL = nrDLSCH;
encDL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the DL-SCH encoder.

```
setTransportBlock(encDL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the DL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encDL(mod,nLayers,outlen,rv);
```

Create and configure a DL-SCH decoder System object.

```
decDL = nrDLSCHDecoder;
decDL.TargetCodeRate = targetCodeRate;
decDL.TransportBlockLength = trBlkLen;
```

Call the DL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decDL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮


blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

Algorithms

LDPC Decoding Algorithms

The `nrDLSCHDecoder` object supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The implementation of the belief propagation algorithm is based on the decoding algorithm presented in [2]. For transmitted LDPC-encoded codeword, c, where $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 1 \,|\, \text{channel output for } c_i)}\right)$.

In each iteration, the key components of the algorithm are updated based on these equations:

$$L(r_{ji}) = 2\,\text{atanh}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right)\right),$$

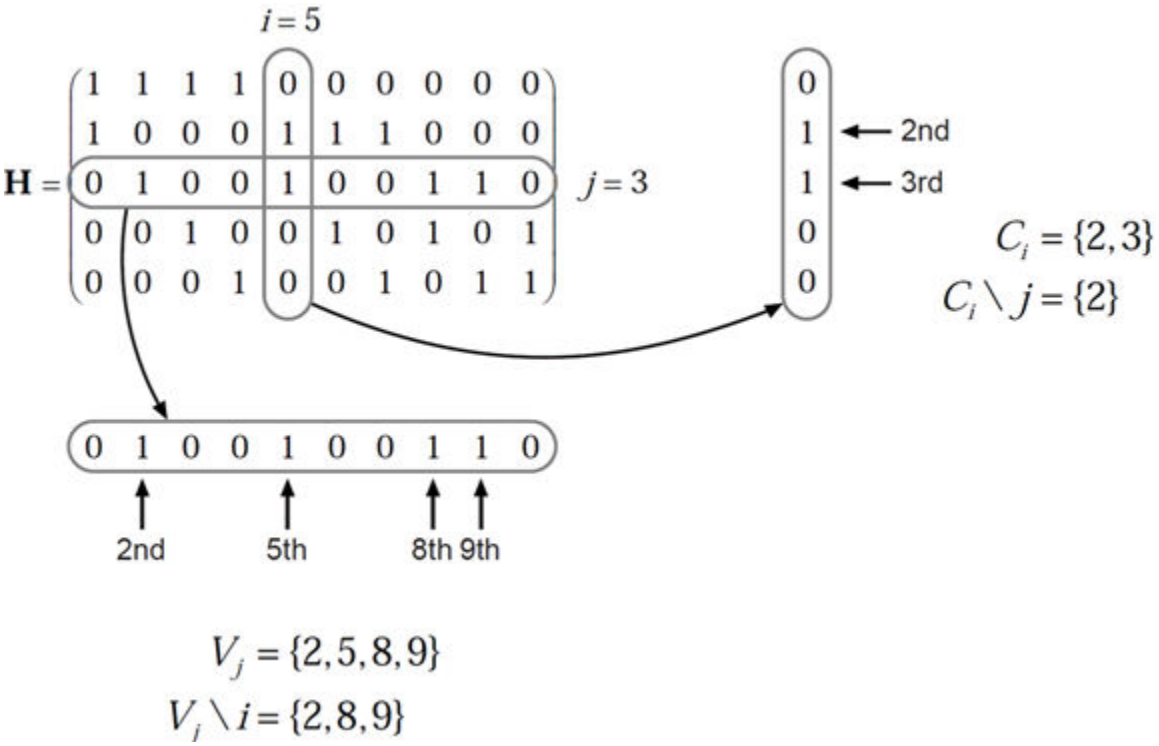
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

At the end of each iteration, $L(Q_i)$ is an updated estimate of the LLR value for the transmitted bit c_i. The value $L(Q_i)$ is the soft-decision output for c_i. If $L(Q_i) < 0$, the hard-decision output for c_i is 1. Otherwise, the output is 0.

Index sets $C_i \backslash j$ and $V_j \backslash i$ are based on the parity-check matrix (PCM). Index sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure highlights the computation of these index sets in a given PCM for $i = 5$ and $j = 3$.



To avoid infinite numbers in the algorithm equations, atanh(1) and atanh(-1) are set to 19.07 and –19.07, respectively. Due to finite precision, MATLAB returns 1 for tanh(19.07) and –1 for tanh(–19.07).

The decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^{\mathrm{T}} = 0$) or after `MaximumLDPCIterationCount` number of iterations.

Layered Belief Propagation Decoding

The implementation of the layered belief propagation algorithm is based on the decoding algorithm presented in [3], Section II.A. The decoding loop iterates over subsets of rows (layers) of the PCM. For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations:

(1) $L(q_{mj}) = L(q_j) - R_{mj}$,

(2) $A_{mj} = \underset{n \in N(m),\ n \neq j}{\sum} \psi(L(q_{mn}))$,

(3) $s_{mj} = \underset{n \in N(m),\ n \neq j}{\prod} \mathrm{sign}(L(q_{mn}))$,

(4) $R_{mj} = -s_{mj}\psi(A_{mj})$, and

(5) $L(q_j) = L(q_{mj}) + R_{mj}$.

For each layer, the decoding equation (5) works on the combined input obtained from the current LLR inputs $L(q_{mj})$ and the previous layer updates R_{mj}.

Because only a subset of the nodes is updated in a layer, the layered belief propagation algorithm is faster compared to the belief propagation algorithm. To achieve the same error rate as attained with belief propagation decoding, use half the number of decoding iterations when using the layered belief propagation algorithm.

Normalized Min-Sum Decoding

The implementation of the normalized min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \underset{n \in N(m),\ n \neq j}{\min} (|L(q_{mn})| \cdot \alpha),$$

where α is in the range (0, 1] and is the scaling factor specified by `ScalingFactor`. This equation is an adaptation of equation (4) presented in [4].

Offset Min-Sum Decoding

The implementation of the offset min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \max\left(\underset{n \in N(m),\ n \neq j}{\min} (|L(q_{mn})| - \beta),\ 0\right),$$

where $\beta \geq 0$ and is the offset specified by `Offset`. This equation is an adaptation of equation (5) presented in [4].

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Networ*k.

[2] Gallager, Robert G. *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

[3] Hocevar, D.E. "A reduced complexity decoder architecture via layered decoding of LDPC codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004*. dois: 10.1109/SIPS.2004.1363033

[4] Chen, Jinghu, R.M. Tanner, C. Jones, and Yan Li. "Improved min-sum decoding algorithms for irregular LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005*. doi: 10.1109/ISIT.2005.1523374

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`nrDLSCH` | `nrULSCHDecoder`

Functions

`nrDLSCHInfo` | `nrPDSCHDecode`

Topics

“NR PDSCH Throughput”

Introduced in R2019a

nrULSCH

Apply UL-SCH encoder processing chain

Description

The `nrULSCH` System object applies the uplink shared channel (UL-SCH) encoder processing chain to a transport block. The UL-SCH encoding process consists of cyclic redundancy check (CRC), code block segmentation and CRC, low-density parity-check (LDPC) encoding, rate matching, and code block concatenation. The object implements these aspects of TS 38.212 [1]:

- Sections 6.2.1: Transport block CRC attachment
- Sections 6.2.2: LDPC base graph selection
- Sections 6.2.3: Code block segmentation and code block CRC attachment
- Sections 6.2.4: Channel coding of UL-SCH
- Sections 6.2.5: Rate matching
- Sections 6.2.6: Code block concatenation

To apply the UL-SCH encoder processing chain:

1. Create the `nrULSCH` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
encUL = nrULSCH
encUL = nrULSCH(Name,Value)
```

Description

`encUL = nrULSCH` creates a UL-SCH encoder System object.

`encUL = nrULSCH(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrULSCH('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes
`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. In both cases, to enable retransmissions when a failure occurs, the object buffers the input data.

Data Types: `logical`

`TargetCodeRate` — Target code rate
`0.5137` (default) | real number

Target code rate, specified as a real number in the interval (0, 1). The default value corresponds to 526/1024.

Tunable: Yes

Data Types: `double`

`LimitedBufferRateMatching` — Enable limited buffer rate matching
`false` (default) | `true`

Enable limited buffer rate matching, specified as `false` or `true`. When set to `false`, the size of the internal buffer used for rate matching is the full coded length of each code block. When set to `true`, you can specify the size of the internal buffer used for rate matching by setting the `LimitedBufferSize` property.

Data Types: `logical`

`LimitedBufferSize` — Limited buffer size
`25344` (default) | positive integer

Limited buffer size used for rate matching, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Dependencies

To enable this property, set `LimitedBufferRateMatching` to `true`.

Data Types: `double`

Usage

Syntax

```
codedBits = encUL(mod,nLayers,outlen,rv)
codedBits = encUL( ___ ,harqID)
```

Description

`codedBits = encUL(mod,nLayers,outlen,rv)` applies the UL-SCH encoder processing chain to the transport block previously loaded into the object. The object returns the encoded, rate-matched, and concatenated code blocks as a codeword of length `outlen`. Before you call this object, you must load a transport block into the object by using the `setTransportBlock` object function. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`codedBits = encUL( ___ ,harqID)` specifies the HARQ process number `harqID` used with the current transport block in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

Input Arguments

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers

integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`outlen` — Output codeword length

nonnegative integer

Output codeword length, in bits, specified as a nonnegative integer. The actual output length is a multiple of the product of the number of bits per symbol and the number of transmission layers. For example, for 64-QAM and 1 transmission layer, if you specify 16 for `outlen`, the actual output length is 6x1x3 = 18.

Data Types: `double`

`rv` — Redundancy version

integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`codedBits` — UL-SCH codeword
binary column vector

UL-SCH codeword, returned as a binary column vector of length `outlen`. A codeword is the encoded, rate-matched, and concatenated code blocks obtained by processing the transport block.

Data Types: `int8`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrULSCH`

getTransportBlock	Get transport block from UL-SCH or DL-SCH encoder
setTransportBlock	Load transport block into UL-SCH or DL-SCH encoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect UL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encUL = nrULSCH;
encUL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the UL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encUL(mod,nLayers,outlen,rv);
```

Create and configure an UL-SCH decoder System object.

```
decUL = nrULSCHDecoder;
decUL.TargetCodeRate = targetCodeRate;
decUL.TransportBlockLength = trBlkLen;
```

Call the UL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decUL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮


blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

UL-SCH Encoding with Multiple HARQ Processes

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object for use with multiple HARQ processes.

```
encUL = nrULSCH;
encUL.MultipleHARQProcesses = true;
```

Load the transport block into the UL-SCH encoder, specifying HARQ process number 2.

```
harqID = 2;
setTransportBlock(encUL,trBlk,harqID);
```

Call the encoder with QPSK modulation scheme, 3 transmission layers, an output length of 10,002 bits, and redundancy version 3. The encoder applies the UL-SCH processing chain to the transport block loaded into the object for HARQ process number 2.

```
mod = 'QPSK';
nLayers = 3;
outlen = 10002;
rv = 3;
codedTrBlock = encUL(mod,nLayers,outlen,rv,harqID);
```

Verify that the encoded transport block has the required number of bits.

```
isequal(length(codedTrBlock),outlen)
```

```
ans = logical
   1
```

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
nrDLSCH | nrULSCHDecoder

Functions
nrPUSCH | nrULSCHInfo

Introduced in R2019a

nrULSCHDecoder

Apply UL-SCH decoder processing chain

Description

The `nrULSCHDecoder` System object applies the uplink shared channel (UL-SCH) decoder processing chain to the soft bits corresponding to a UL-SCH-encoded transport block. The UL-SCH decoding process consists of rate recovery, low-density parity-check (LDPC) decoding, desegmentation, and cyclic redundancy check (CRC) decoding. The object implements the inverse operation of the UL-SCH encoding process specified in these sections of TS 38.212 [1]:

- Sections 6.2.1: Transport block CRC attachment
- Sections 6.2.2: LDPC base graph selection
- Sections 6.2.3: Code block segmentation and code block CRC attachment
- Sections 6.2.4: Channel coding of UL-SCH
- Sections 6.2.5: Rate matching
- Sections 6.2.6: Code block concatenation

To apply the UL-SCH decoder processing chain:

1. Create the `nrULSCHDecoder` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
decUL = nrULSCHDecoder
decUL = nrULSCHDecoder(Name,Value)
```

Description

`decUL = nrULSCHDecoder` creates a UL-SCH decoder System object.

`decUL = nrULSCHDecoder(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: For example, `nrULSCHDecoder('MultipleHARQProcesses',true)` creates the object and enables multiple hybrid automatic repeat-request (HARQ) processes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`MultipleHARQProcesses` — Enable multiple HARQ processes

`false` (default) | `true`

Enable multiple HARQ processes, specified as `false` or `true`. When set to `false`, the object uses a single process. When set to `true`, the object uses multiple HARQ processes, at most 16. To enable soft combining of retransmissions before LDPC decoding, the object maintains a soft buffer for each HARQ process.

Data Types: `logical`

`TargetCodeRate` — Target code rate

`0.5137` (default) | real number

Target code rate, specified as a real number in the interval (0, 1). The default value corresponds to 526/1024.

Tunable: Yes

Data Types: `double`

`TransportBlockLength` — Length of decoded transport block

`5120` (default) | positive integer

Length of decoded transport block, in bits, specified as a positive integer.

Tunable: Yes

Data Types: `double`

`LimitedBufferRateRecovery` — Enable limited buffer rate recovery

`false` (default) | `true`

Enable limited buffer rate recovery, specified as `false` or `true`. When set to `false`, the size of the internal buffer used for rate recovery is the full coded length of each code block. When set to `true`, you can specify the size of the internal buffer used for rate recovery by setting the `LimitedBufferSize` property.

Data Types: `logical`

`LimitedBufferSize` — Limited buffer size

`25344` (default) | positive integer

Limited buffer size used for rate recovery, specified as a positive integer. The default value corresponds to 384×66, which is the maximum coded length of a code block. The default value implies no limit on the buffer size.

Dependencies

To enable this property, set `LimitedBufferRateRecovery` to `true`.

Data Types: `double`

`MaximumLDPCIterationCount` — Maximum LDPC decoding iterations

`12` (default) | positive integer

Maximum LDPC decoding iterations, specified as a positive integer. Since early termination is enabled, decoding stops once parity-checks are satisfied. In this case, fewer iterations take place than the maximum specified by this argument.

Data Types: `double`

`LDPCDecodingAlgorithm` — LDPC decoding algorithm

`'Belief propagation'` (default) | `'Layered belief propagation'` | `'Normalized min-sum'` | `'Offset min-sum'`

LDPC decoding algorithm, specified as one of these values:

- `'Belief propagation'` — Use this option to specify the belief-passing or message-passing algorithm.
- `'Layered belief propagation'` — Use this option to specify the layered belief-passing algorithm, which is suitable for quasi-cyclic parity-check matrices (PCMs).
- `'Normalized min-sum'` — Use this option to specify the layered belief propagation algorithm with normalized min-sum approximation.
- `'Offset min-sum'` — Use this option to specify the layered belief propagation algorithm with offset min-sum approximation.

For more information on these algorithms, see LDPC Decoding Algorithms on page 2-48.

Data Types: `char` | `string`

`ScalingFactor` — Scaling factor for normalized min-sum decoding

`0.75` (default) | real scalar in the range (0, 1]

Scaling factor for normalized min-sum decoding, specified as a real scalar in the range (0, 1].

Dependencies

To enable this property, set the `LDPCDecodingAlgorithm` property to `'Normalized min-sum'`.

Data Types: `double`

`Offset` — Offset for offset min-sum decoding

`0.5` (default) | nonnegative finite real scalar

Offset for offset min-sum decoding, specified as a nonnegative finite real scalar.

Dependencies

To enable this property, set the `LDPCDecodingAlgorithm` property to `'Offset min-sum'`.

Data Types: `double`

Usage

Syntax

```
trblk = decUL(softbits,mod,nLayers,rv)
trblk = decUL( ___ ,harqID)
[trblk,blkerr] = decUL( ___ )
```

Description

`trblk = decUL(softbits,mod,nLayers,rv)` applies the UL-SCH decoder processing chain to the input `softbits` and returns the decoded bits. `mod` specifies the modulation scheme. `nLayers` specifies the number of transmission layers. `rv` specifies the redundancy version of the transmission.

`trblk = decUL( ___ ,harqID)` specifies the HARQ process number `harqID` used with the current transport block in addition to the input arguments in the previous syntax. To use this syntax, set the `MultipleHARQProcesses` property to `true`. When the property is set to `false`, the object uses HARQ process number 0.

When the object receives codewords with different redundancy version for an individual HARQ process, the object uses soft buffer state retention to enable soft combining of retransmissions. When you enable multiple HARQ processes, the object maintains independent buffers for each process.

`[trblk,blkerr] = decUL( ___ )` returns an error flag, using the input arguments in any of the previous syntaxes. A value of 1 in `blkerr` indicates an error during transport block decoding.

Input Arguments

`softbits` — Approximate LLR soft bits

real column vector

Approximate log-likelihood ratio (LLR) soft bits, corresponding to the UL-SCH-encoded transport block, specified as a real column vector.

Data Types: `single` | `double`

`mod` — Modulation scheme

`'pi/2-BPSK'` | `'QPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'`

Modulation scheme, specified as `'pi/2-BPSK'`, `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`. This modulation scheme determines the modulation type and number of bits used per modulation symbol.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`nLayers` — Number of transmission layers

integer from 1 to 4

Number of transmission layers, specified as an integer from 1 to 4. For more information, see TS 38.211 Section 6.3.1.3.

Data Types: `double`

`rv` — Redundancy version
integer from 0 to 3

Redundancy version, specified as an integer from 0 to 3.

Data Types: `double`

`harqID` — HARQ process number
integer from 0 to 15

HARQ process number, specified as an integer from 0 to 15.

Data Types: `double`

Output Arguments

`trblk` — Decoded UL-SCH transport blocks
binary column vector

Decoded UL-SCH transport block, returned as a binary column vector of length specified by the `TransportBlockLength` property.

`blkerr` — Result of UL-SCH transport block decoding
logical scalar

Result of UL-SCH transport block decoding, returned as a logical scalar. A value of `1` indicates an error during transport block decoding.

Data Types: `logical`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrULSCHDecoder`

resetSoftBuffer	Reset soft buffer for HARQ process in UL-SCH or DL-SCH decoder

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Connect UL-SCH Encoder and Decoder Back to Back

Generate a random sequence of binary values corresponding to one transport block of length 5120.

```
trBlkLen = 5120;
trBlk = randi([0 1],trBlkLen,1,'int8');
```

Create and configure an UL-SCH encoder System object with the specified target code rate.

```
targetCodeRate = 567/1024;
encUL = nrULSCH;
encUL.TargetCodeRate = targetCodeRate;
```

Load the transport block into the UL-SCH encoder.

```
setTransportBlock(encUL,trBlk);
```

Call the encoder with 64-QAM modulation scheme, 1 transmission layer, an output length of 10,240 bits, and redundancy version 0. The encoder applies the UL-SCH processing chain to the transport block loaded into the object.

```
mod = '64QAM';
nLayers = 1;
outlen = 10240;
rv = 0;
codedTrBlock = encUL(mod,nLayers,outlen,rv);
```

Create and configure an UL-SCH decoder System object.

```
decUL = nrULSCHDecoder;
decUL.TargetCodeRate = targetCodeRate;
decUL.TransportBlockLength = trBlkLen;
```

Call the UL-SCH decoder on the soft bits representing the encoded transport block. Use the configuration parameters specified for the encoder. The error flag in the output indicates that the block decoding does not have errors.

```
rxSoftBits = 1.0 - 2.0*double(codedTrBlock);
[decbits,blkerr] = decUL(rxSoftBits,mod,nLayers,rv)
```

```
decbits = 5120x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮
```

```
blkerr = logical
   0
```

Verify that the transmitted and received message bits are identical.

```
isequal(decbits,trBlk)
```

```
ans = logical
   1
```

Algorithms

LDPC Decoding Algorithms

The `nrULSCHDecoder` object supports these four LDPC decoding algorithms.

Belief Propagation Decoding

The implementation of the belief propagation algorithm is based on the decoding algorithm presented in [3]. For transmitted LDPC-encoded codeword, c, where $c = (c_0, c_1, ..., c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \,|\, \text{channel output for } c_i)}{\Pr(c_i = 1 \,|\, \text{channel output for } c_i)}\right)$.

In each iteration, the key components of the algorithm are updated based on these equations:

$$L(r_{ji}) = 2\,\text{atanh}\left(\prod_{i' \in V_j \backslash i} \tanh\left(\frac{1}{2}L(q_{i'j})\right)\right),$$

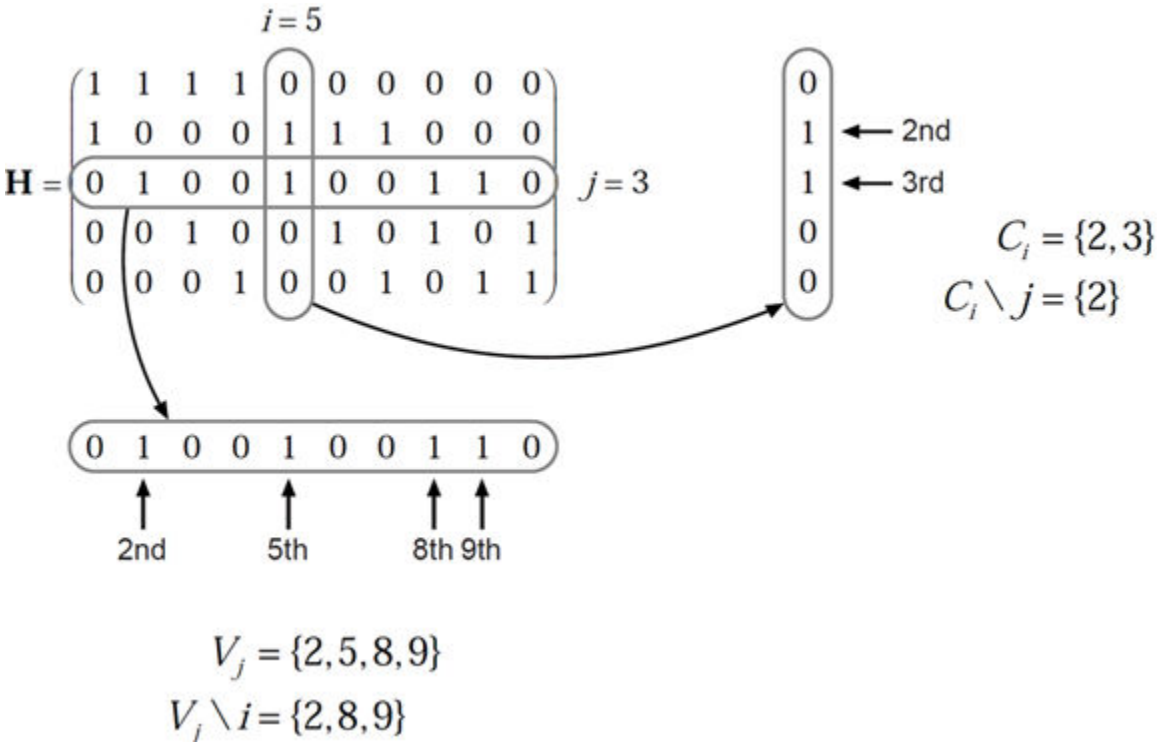
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i}).$$

At the end of each iteration, $L(Q_i)$ is an updated estimate of the LLR value for the transmitted bit c_i. The value $L(Q_i)$ is the soft-decision output for c_i. If $L(Q_i) < 0$, the hard-decision output for c_i is 1. Otherwise, the output is 0.

Index sets $C_i \backslash j$ and $V_j \backslash i$ are based on the parity-check matrix (PCM). Index sets C_i and V_j correspond to all nonzero elements in column i and row j of the PCM, respectively.

This figure highlights the computation of these index sets in a given PCM for $i = 5$ and $j = 3$.



To avoid infinite numbers in the algorithm equations, atanh(1) and atanh(-1) are set to 19.07 and –19.07, respectively. Due to finite precision, MATLAB returns 1 for tanh(19.07) and –1 for tanh(–19.07).

The decoding terminates when all parity checks are satisfied ($\mathbf{Hc}^{\mathrm{T}} = 0$) or after `MaximumLDPCIterationCount` number of iterations.

Layered Belief Propagation Decoding

The implementation of the layered belief propagation algorithm is based on the decoding algorithm presented in [4], Section II.A. The decoding loop iterates over subsets of rows (layers) of the PCM. For each row, m, in a layer and each bit index, j, the implementation updates the key components of the algorithm based on these equations:

(1) $L(q_{mj}) = L(q_j) - R_{mj}$,

(2) $A_{mj} = \underset{n \in N(m),\ n \neq j}{\sum} \psi(L(q_{mn}))$,

(3) $s_{mj} = \underset{n \in N(m),\ n \neq j}{\prod} \mathrm{sign}(L(q_{mn}))$,

(4) $R_{mj} = -s_{mj}\psi(A_{mj})$, and

(5) $L(q_j) = L(q_{mj}) + R_{mj}$.

For each layer, the decoding equation (5) works on the combined input obtained from the current LLR inputs $L(q_{mj})$ and the previous layer updates R_{mj}.

Because only a subset of the nodes is updated in a layer, the layered belief propagation algorithm is faster compared to the belief propagation algorithm. To achieve the same error rate as attained with belief propagation decoding, use half the number of decoding iterations when using the layered belief propagation algorithm.

Normalized Min-Sum Decoding

The implementation of the normalized min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})|) \cdot \alpha),$$

where α is in the range (0, 1] and is the scaling factor specified by `ScalingFactor`. This equation is an adaptation of equation (4) presented in [5].

Offset Min-Sum Decoding

The implementation of the offset min-sum decoding algorithm follows the layered belief propagation algorithm with equation (2) replaced by

$$A_{mj} = \max(\min_{\substack{n \in N(m) \\ n \neq j}} (|L(q_{mn})| - \beta), 0),$$

where $\beta \geq 0$ and is the offset specified by `Offset`. This equation is an adaptation of equation (5) presented in [5].

References

[1] 3GPP TS 38.212. “NR; Multiplexing and channel coding.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] Gallager, Robert G. *Low-Density Parity-Check Codes*, Cambridge, MA, MIT Press, 1963.

[4] Hocevar, D.E. "A reduced complexity decoder architecture via layered decoding of LDPC codes." In *IEEE Workshop on Signal Processing Systems, 2004. SIPS 2004*. doi: 10.1109/SIPS.2004.1363033

[5] Chen, Jinghu, R.M. Tanner, C. Jones, and Yan Li. "Improved min-sum decoding algorithms for irregular LDPC codes." In *Proceedings. International Symposium on Information Theory, 2005. ISIT 2005*. doi: 10.1109/ISIT.2005.1523374

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
`nrDLSCHDecoder` | `nrULSCH`

Functions
`nrPUSCHDecode` | `nrULSCHInfo`

Introduced in R2019a

nrTDLChannel

Send signal through TDL channel model

Description

The `nrTDLChannel` System object sends an input signal through a tapped delay line (TDL) multi-input multi-output (MIMO) link-level fading channel to obtain the channel-impaired signal. The object implements the following aspects of TR 38.901 [1]:

- Section 7.7.2: TDL models
- Section 7.7.3: Scaling of delays
- Section 7.7.5.2 TDL extension: Applying a correlation matrix
- Section 7.7.6: K-factor for LOS channel models

To send a signal through the TDL MIMO channel model:

1. Create the `nrTDLChannel` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?.

Creation

Syntax

```
tdl = nrTDLChannel
tdl = nrTDLChannel(Name,Value)
```

Description

`tdl = nrTDLChannel` creates a TDL MIMO channel System object.

`tdl = nrTDLChannel(Name,Value)` creates the object with properties set by using one or more name-value pairs. Enclose the property name inside quotes, followed by the specified value. Unspecified properties take default values.

Example: `tdl = nrTDLChannel('DelayProfile','TDL-D','DelaySpread',2e-6)` creates a TDL channel model with TDL-D delay profile and a 2-microseconds delay spread.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`DelayProfile` — TDL delay profile

`'TDL-A'` (default) | `'TDL-B'` | `'TDL-C'` | `'TDL-D'` | `'TDL-E'` | `'Custom'`

TDL delay profile, specified as one of `'TDL-A'`, `'TDL-B'`, `'TDL-C'`, `'TDL-D'`, `'TDL-E'`, or `'Custom'`. See TR 38.901 Section 7.7.2, Tables 7.7.2-1 to 7.7.2-5.

When you set this property to `'Custom'`, configure the delay profile using properties `PathDelays`, `AveragePathGains`, `FadingDistribution`, and `KFactorFirstTap`.

Data Types: `char` | `string`

`PathDelays` — Discrete path delays in seconds

`0.0` (default) | numeric scalar | row vector

Discrete path delays in seconds, specified as a numeric scalar or row vector. `AveragePathGains` and `PathDelays` must have the same size.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`AveragePathGains` — Average path gains in dB

`0.0` (default) | numeric scalar | row vector

Average path gains in dB, specified as a numeric scalar or row vector. `AveragePathGains` and `PathDelays` must have the same size.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `double`

`FadingDistribution` — Fading process statistical distribution

`'Rayleigh'` (default) | `'Rician'`

Fading process statistical distribution, specified as `'Rayleigh'` or `'Rician'`.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'`.

Data Types: `char` | `string`

`KFactorFirstTap` — K-factor of first tap of delay profile in dB

`13.3` (default) | numeric scalar

K-factor of first tap of delay profile in dB, specified as a numerical scalar. The default value corresponds to the K-factor of the first tap of TDL-D as defined in TR 38.901 Section 7.7.2, Table 7.7.2-4.

Dependencies

To enable this property, set `DelayProfile` to `'Custom'` and `FadingDistribution` to `'Rician'`.

Data Types: `double`

`DelaySpread` — Desired RMS delay spread in seconds

`30e-9` (default) | numeric scalar

Desired root mean square (RMS) delay spread in seconds, specified as a numeric scalar. For examples of desired RMS delay spreads, $DS_{desired}$, see TR 38.901 Section 7.7.3 and Tables 7.7.3-1 and 7.7.3-2.

Dependencies

To enable this property, set `DelayProfile` to `'TDL-A'`, `'TDL-B'`, `'TDL-C'`, `'TDL-D'`, or `'TDL-E'`. This property does not apply for custom delay profile.

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift in Hz

`5` (default) | nonnegative numeric scalar

Maximum Doppler shift in Hz, specified as a nonnegative numeric scalar. This property applies to all channel paths. When the maximum Doppler shift is set to 0, the channel remains static for the entire input. To generate a new channel realization, reset the object by calling the `reset` function.

Data Types: `double`

`KFactorScaling` — K-factor scaling

`false` (default) | `true`

K-factor scaling, specified as `false` or `true`. When set to `true`, the `KFactor` property specifies the desired K-factor, and the object applies K-factor scaling as described in TR 38.901 Section 7.7.6.

Note K-factor scaling modifies both the path delays and path powers.

Dependencies

To enable this property, set `DelayProfile` to `'TDL-D'` or `'TDL-E'`.

Data Types: `double`

`KFactor` — Desired K-factor for scaling in dB

`9.0` (default) | numeric scalar

Desired K-factor for scaling in dB, specified as a numeric scalar. For typical K-factor values, see TR 38.901 Section 7.7.6 and Table 7.5-6.

Note

- K-factor scaling modifies both the path delays and path powers.
- `K-factor` applies to the overall delay profile. Specifically, the K-factor after the scaling is K_{model} as described in TR 38.901 Section 7.7.6. K_{model} is the ratio of the power of the first path LOS to the total power of all the Rayleigh paths, including the Rayleigh part of the first path.

Dependencies

To enable this property, set `KFactorScaling` to `true`.

Data Types: `double`

`SampleRate` — Sample rate of input signal in Hz

`30.72e6` (default) | positive numeric scalar

Sample rate of input signal in Hz, specified as a positive numeric scalar.

Data Types: `double`

`MIMOCorrelation` — Correlation between UE and BS antennas

`'Low'` (default) | `'Medium'` | `'Medium-A'` | `'UplinkMedium'` | `'High'` | `'Custom'`

Correlation between user equipment (UE) and base station (BS) antennas, specified as one of these values:

- `'Low'` or `'High'` — Applies to both uplink and downlink. `'Low'` is equivalent to no correlation between antennas.
- `'Medium'` or `'Medium-A'` — For downlink, see TS 36.101 Annex B.2.3.2. For uplink, see TS 36.104 Annex B.5.2. The `TransmissionDirection` property controls the transmission direction.
- `'UplinkMedium'` — See TS 36.104, Annex B.5.2.
- `'Custom'` — The `ReceiveCorrelationMatrix` property specifies the correlation between UE antennas, and the `TransmitCorrelationMatrix` property specifies the correlation between BS antennas. See TR 38.901 Section 7.7.5.2.

For more details on correlation between UE and BS antennas, see TS 36.101 [2] and TS 36.104 [3]

Data Types: `char` | `string`

`Polarization` — Antenna polarization arrangement

`'Co-Polar'` (default) | `'Cross-Polar'` | `'Custom'`

Antenna polarization arrangement, specified as `'Co-Polar'`, `'Cross-Polar'`, `'Custom'`.

Data Types: `char` | `string`

`TransmissionDirection` — Transmission direction

`'Downlink'` (default) | `'Uplink'`

Transmission direction, specified as `'Downlink'` or `'Uplink'`.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Low'`, `'Medium'`, `'Medium-A'`, `'UplinkMedium'`, or `'High'`.

Data Types: `char` | `string`

`NumTransmitAntennas` — Number of transmit antennas

`1` (default) | positive integer

Number of transmit antennas, specified as a positive integer.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Low'`, `'Medium'`, `'Medium-A'`, `'UplinkMedium'`, or `'High'`, or set both `MIMOCorrelation` and `Polarization` to `'Custom'`.

Data Types: `double`

`NumReceiveAntennas` — Number of receive antennas

2 (default) | positive integer

Number of receive antennas, specified as a positive integer.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Low'`, `'Medium'`, `'Medium-A'`, `'UplinkMedium'`, or `'High'`.

Data Types: `double`

`TransmitCorrelationMatrix` — Spatial correlation of transmitter

`[1]` (default) | 2-D matrix | 3-D array

Spatial correlation of transmitter, specified as a 2-D matrix or 3-D array.

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `TransmitCorrelationMatrix` as a 2-D Hermitian matrix of size N_T-by-N_T. N_T is the number of transmit antennas. The main diagonal elements must be all ones, and the off-diagonal elements must have a magnitude smaller than or equal to one.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `TransmitCorrelationMatrix` as one of these arrays:
  - 2-D Hermitian matrix of size N_T-by-N_T with element properties as previously described. Each path has the same transmit correlation matrix.
  - 3-D array of size N_T-by-N_T-by-N_P, where each submatrix of size N_T-by-N_T is a Hermitian matrix with element properties as previously described. Each path has its own transmit correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to either `'Co-Polar'` or `'Cross-Polar'`.

Data Types: `double`

`ReceiveCorrelationMatrix` — Spatial correlation of receiver

`[1 0; 0 1]` (default) | 2-D matrix | 3-D array

Spatial correlation of receiver, specified as a 2-D matrix or 3-D array.

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `ReceiveCorrelationMatrix` as a 2-D Hermitian matrix of size N_R-by-N_R. N_R is the number of receive antennas. The main diagonal elements must be all ones, and the off-diagonal elements must have a magnitude smaller than or equal to one.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `ReceiveCorrelationMatrix` as one of these arrays:

- 2-D Hermitian matrix of size N_R-by-N_R with element properties as previously described. Each path has the same receive correlation matrix.
- 3-D array of size N_R-by-N_R-by-N_P, where each submatrix of size N_R-by-N_R is a Hermitian matrix with element properties as previously described. Each path has its own receive correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to either `'Co-Polar'` or `'Cross-Polar'`.

Data Types: `double`

`TransmitPolarizationAngles` — Transmit polarization slant angles in degrees

`[45 -45]` (default) | row vector

Transmit polarization slant angles in degrees, specified as a row vector.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Cross-Polar'`.

Data Types: `double`

`ReceivePolarizationAngles` — Receive polarization slant angles in degrees

`[90 0]` (default) | row vector

Receive polarization slant angles in degrees, specified as a row vector.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Cross-Polar'`.

Data Types: `double`

`XPR` — Cross-polarization power ratio in dB

`10.0` (default) | numeric scalar | row vector

Cross-polarization power ratio in dB, specified as a numeric scalar or a row vector. This property corresponds to the ratio between the vertical-to-vertical (P_{VV}) and vertical-to-horizontal (P_{VH}) polarizations defined for the clustered delay line (CDL) models in TR 38.901 Section 7.7.1.

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `XPR` as a scalar.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `XPR` as one of these values:
  - Scalar — Each path has the same cross-polarization power ratio.
  - Row vector of size 1-by-N_P — Each path has its own cross-polarization power ratio.

The default value corresponds to the cluster-wise cross-polarization power ratio of CDL-A as defined in TR 38.901 Section 7.7.1, Table 7.7.1-1.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Cross-Polar'`.

Data Types: `double`

`SpatialCorrelationMatrix` — Combined correlation for channel

`[1 0; 0 1]` (default) | 2-D matrix | 3-D array

Combined correlation for the channel, specified as 2-D matrix or 3-D array. The matrix determines the product of the number of transmit antennas (N_T) and the number of receive antennas (N_R).

- If the channel is frequency-flat (`PathDelays` is a scalar), specify `SpatialCorrelationMatrix` as a 2-D Hermitian matrix of size (N_T × N_R)-by-(N_T × N_R).The magnitude of any off-diagonal element must be no larger than the geometric mean of the two corresponding diagonal elements.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `SpatialCorrelationMatrix` as one of these arrays:
  - 2-D Hermitian matrix of size (N_T × N_R)-by-(N_T × N_R) with off-diagonal element properties as previously described. Each path has the same spatial correlation matrix.
  - 3-D array of size (N_T × N_R)-by-(N_T × N_R)-by-N_P array — where each matrix of size (N_T × N_R)-by-(N_T × N_R) is a Hermitian matrix with off-diagonal element properties as previously described. Each path has its own spatial correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Custom'`.

Data Types: `double`

`NormalizePathGains` — Normalize path gains

`true` (default) | `false`

Normalize path gains, specified as `true` or `false`. Use this property to normalize the fading processes. When this property is set to `true`, the total power of the path gains, averaged over time, is 0 dB. When this property is set to `false`, the path gains are not normalized. The average powers of the path gains are specified by the selected delay profile, or if `DelayProfile` is set to `'Custom'`, by the `AveragePathGains` property.

Data Types: `logical`

`InitialTime` — Time offset of fading process in seconds

`0.0` (default) | numeric scalar

Time offset of fading process in seconds, specified as a numeric scalar.

Tunable: Yes

Data Types: `double`

`NumSinusoids` — Number of modeling sinusoids

48 (default) | positive integer

Number of modeling sinusoids, specified as a positive integer. These sinusoids model the fading process.

Data Types: `double`

`RandomStream` — Source of random number stream
`'mt19937ar with seed'` (default) | `'Global stream'`

Source of random number stream, specified as one of the following:

- `'mt19937ar with seed'` — The object uses the mt19937ar algorithm for normally distributed random number generation. Calling the `reset` function resets the filters and reinitializes the random number stream to the value of the `Seed` property.
- `'Global stream'` — The object uses the current global random number stream for normally distributed random number generation. Calling the `reset` function resets only the filters.

`Seed` — Initial seed of mt19937ar random number stream
73 (default) | nonnegative numeric scalar

Initial seed of mt19937ar random number stream, specified as a nonnegative numeric scalar.

Dependencies

To enable this property, set RandomStream to `'mt19937ar with seed'`. When calling the `reset` function, the seed reinitializes the mt19937ar random number stream.

Data Types: `double`

`NormalizeChannelOutputs` — Normalize channel outputs by the number of receive antennas
`true` (default) | `false`

Normalize channel outputs by the number of receive antennas, specified as `true` or `false`.

Data Types: `logical`

Usage

Syntax

```
signalOut = tdl(signalIn)
[signalOut,pathGains] = tdl(signalIn)
[signalOut,pathGains,sampleTimes] = tdl(signalIn)
```

Description

`signalOut = tdl(signalIn)` sends the input signal through a TDL MIMO fading channel and returns the channel-impaired signal.

`[signalOut,pathGains] = tdl(signalIn)` also returns the MIMO channel path gains of the underlying fading process.

`[signalOut,pathGains,sampleTimes] = tdl(signalIn)` also returns the sample times of the channel snapshots of the path gains.

Input Arguments

`signalIn` — Input signal

complex scalar | vector | N_S-by-N_T matrix

Input signal, specified as a complex scalar, vector, or N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`signalOut` — Output signal

complex scalar | vector | N_S-by-N_R matrix

Output signal, returned as a complex scalar, vector, or N_S-by-N_R matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas.

The output signal data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathGains` — MIMO channel path gains of fading process

N_S-by-N_P-by-N_T-by-N_R complex matrix

MIMO channel path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_P is the number of paths, specified by the length of the `PathDelays` property of `tdl`.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

The path gains data type is of the same precision as the input signal data type.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampleTimes` — Sample times of channel snapshots

N_S-by-1 column vector of real numbers

Sample times of the channel snapshots of the path gains, returned as an N_S-by-1 column vector of real numbers. N_S is the first dimension of `pathGains` that corresponds to the number of samples.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrTDLChannel`

info	Get characteristic information about link-level MIMO fading channel
getPathFilters	Get path filter impulse response for link-level MIMO fading channel

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Transmission Over MIMO Channel Model with Delay Profile TDL

Display the 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 UE velocity of 30 km/h:

```
v = 30.0;                    % UE 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;     % UE 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.

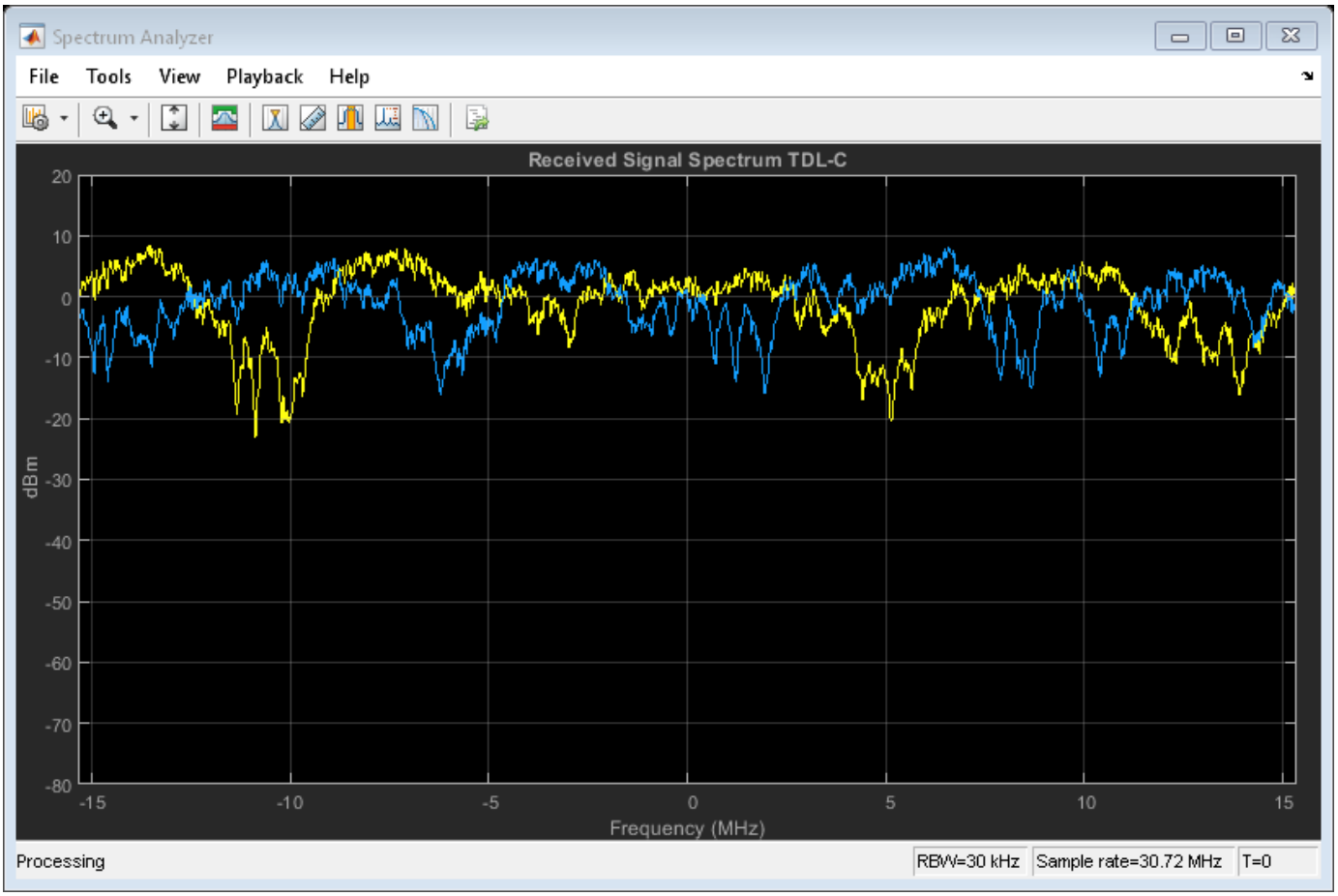
```
SR = 30.72e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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,...
    'AveragingMethod','Exponential','ForgettingFactor',0.99 );
analyzer.Title = ['Received Signal Spectrum ' tdl.DelayProfile];
analyzer(rxWaveform);
```



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 the transmit and receive antenna arrays for SISO operation.

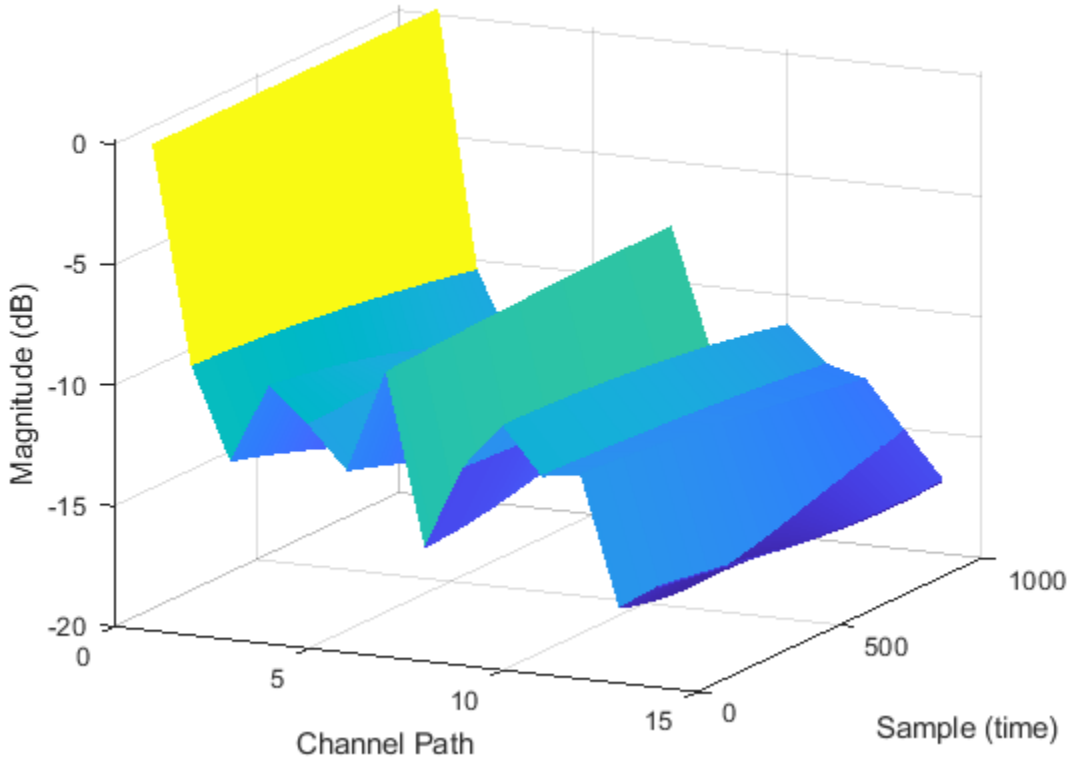
```
tdl.NumTransmitAntennas = 1;
tdl.NumReceiveAntennas = 1;
```

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)');
```



Transmission Over TDL-D Channel Model with Cross-Polar Antennas

Display the waveform spectrum received through a tapped delay line (TDL) channel model using delay profile TDL-D from TR 38.901 Section 7.7.2.

Configure 4-by-2, high-correlation, cross-polar antennas as specified in TS 36.101 Annex B.2.3A.3.

```
tdl = nrTDLChannel;
tdl.NumTransmitAntennas = 4;
tdl.DelayProfile = 'TDL-D';
tdl.DelaySpread = 10e-9;
tdl.KFactorScaling = true;
tdl.KFactor = 7.0;
```

```
tdl.MIMOCorrelation = 'High';
tdl.Polarization = 'Cross-Polar';
```

Create a random waveform of 1 subframe duration with 4 antennas.

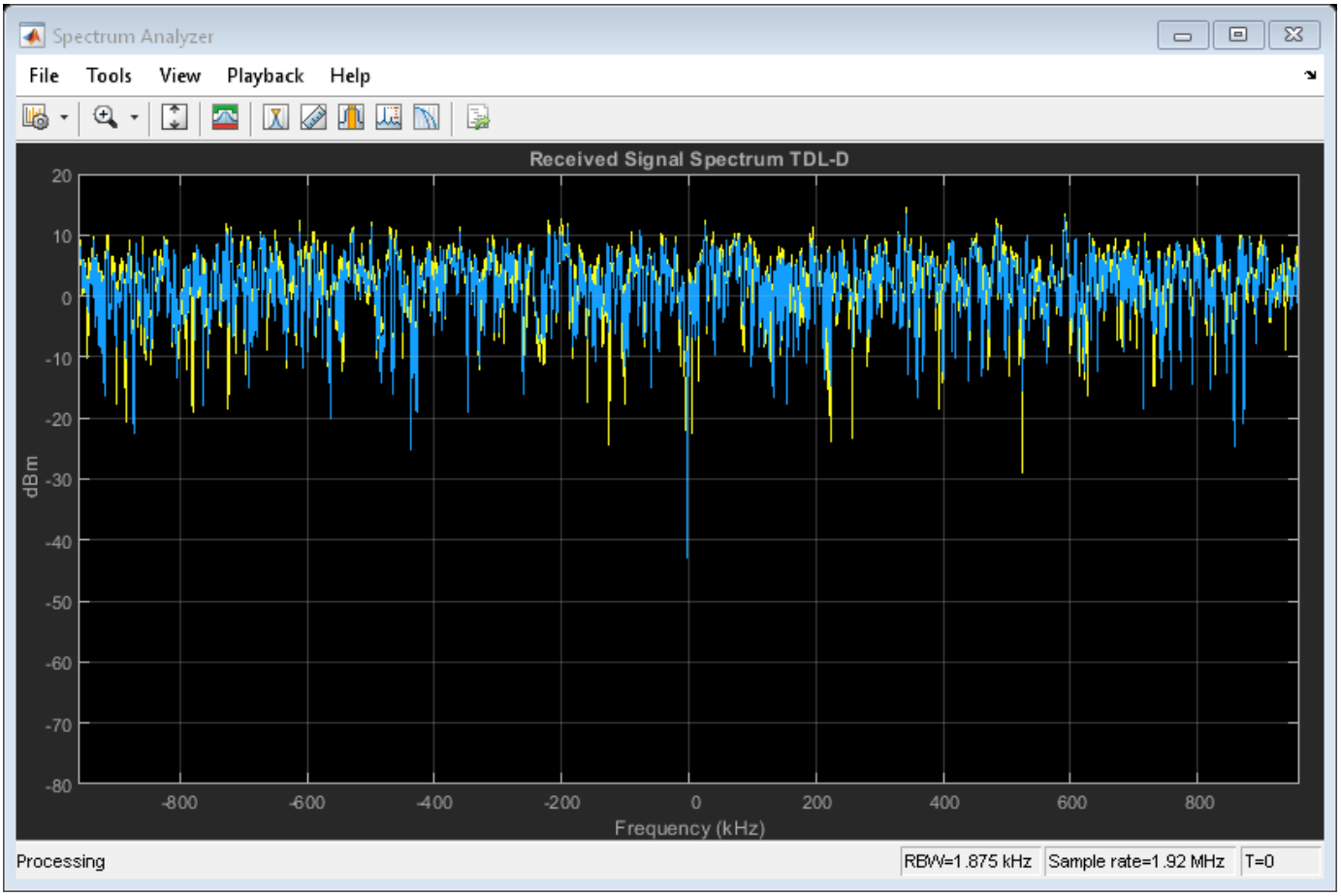
```
SR = 1.92e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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);
```



Transmission Over TDL Channel Model with Custom Delay Profile

Transmit waveform through a tapped delay line (TDL) channel model from TR 38.901 Section 7.7.2 with customized delay profile.

Define the channel configuration structure using an `nrTDLChannel` System object. Customize the delay profile with two taps.

- First tap: Rician with average power 0 dB, K-factor 10 dB, and zero delay.
- Second tap: Rayleigh with average power −5 dB, and 45 ns path delay using TDL-D.

```
tdl = nrTDLChannel;
tdl.NumTransmitAntennas = 1;
tdl.DelayProfile = 'Custom';
tdl.FadingDistribution = 'Rician';
tdl.KFactorFirstTap = 10.0;
tdl.PathDelays = [0.0 45e-9];
tdl.AveragePathGains = [0.0 -5.0];
```

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);
```

References

[1] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 36.101. “Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 36.104. “Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

Objects
`comm.MIMOChannel` | `nrCDLChannel`

Introduced in R2018b

Objects

networkTrafficFTP

FTP application traffic pattern generator

Description

The `networkTrafficFTP` object specifies the configuration parameters to generate a file transfer protocol (FTP) application traffic pattern based on the 3GPP TR 36.814 specification and the IEEE® 802.11ax™ Evaluation Methodology.

You can use the FTP application traffic pattern in 5G and WLAN (requires WLAN Toolbox™) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgFTP = networkTrafficFTP
cfgFTP = networkTrafficFTP(Name,Value)
```

Description

`cfgFTP = networkTrafficFTP` creates a default FTP application traffic pattern object.

`cfgFTP = networkTrafficFTP(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an FTP packet.

Properties

`FixedFileSize` — Custom size of file to be transmitted

`[ ]` (default) | positive scalar

Custom size of the file to be transmitted, specified as a positive scalar. This property must be expressed in megabytes. If you do not specify this property, the object uses the truncated Lognormal distribution to calculate the file size.

Data Types: `double`

`LogNormalMu` — Truncated Lognormal distribution mu value

`14.45` (default) | positive scalar

Truncated Lognormal distribution mu value, specified as a positive scalar. The object uses this property to calculate the file size.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

`LogNormalSigma` — Truncated Lognormal distribution sigma value

`0.35` (default) | positive scalar

Truncated Lognormal distribution sigma value, specified as a positive scalar. The object uses this value to calculate the file size.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

`UpperLimit` — Truncated Lognormal distribution upper limit

`5` (default) | positive scalar

Truncated Lognormal distribution upper limit, specified as a positive scalar. The object uses this value to calculate the file size. The generated file size value must be less than or equal to the `UpperLimit` value. If the generated file size value is greater than `UpperLimit`, the object discards the file size and creates a new one.

Dependencies

To enable this property, set the `FixedFileSize` property to `[ ]`.

Data Types: `double`

`ReadingTime` — Time interval between two successive file transfers

[ ] (default) | positive scalar

Time interval between two consecutive file transfers, specified as a positive scalar. This property must be expressed in milliseconds. To specify a customized value for the reading time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the reading time.

Data Types: `double`

`ExponentialMean` — Exponential distribution mean value

`180000` (default) | nonnegative scalar

Exponential distribution mean value, specified as a nonnegative scalar. This property must be expressed in milliseconds. The object uses this property to calculate the reading time.

Dependencies

To enable this property, set the `ReadingTime` property to `[ ]`.

Data Types: `double`

`PoissonMean` — Poisson distribution mean value

[ ] (default) | nonnegative scalar

Poisson distribution mean value, specified as a nonnegative scalar. This property must be expressed in milliseconds. The object uses this property to calculate the packet interarrival time.

Data Types: `double`

`PacketInterArrivalTime` — Time interval between two successively generated packets

`0` (default) | nonnegative scalar

Time interval between two successively generated packets, specified as a nonnegative scalar. This property must be expressed in milliseconds.

Dependencies

To enable this property, set the `PoissonMean` property to `[ ]`.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate FTP packet

`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an FTP packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in FTP packet

1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the FTP packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the packet size, the object truncates the application data. If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `true` or `1`.

Data Types: `double`

Object Functions

Specific to This Object

generate Generate next FTP, On-Off, or VoIP application traffic packet

Examples

Generate FTP Application Traffic Pattern

Create a default FTP application traffic pattern object.

```
cfgFTP = networkTrafficFTP;
```

Generate an FTP application traffic pattern.

```
[dt,packetSize] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern Using Reading Time

Create an FTP application traffic pattern object, specifying the reading time.

```
cfgFTP = networkTrafficFTP('ReadingTime',5);
```

Generate an FTP application traffic pattern.

```
[dt,packetSize] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern and Data Packet

Create an FTP application traffic pattern object to generate an FTP data packet.

```
cfgFTP = networkTrafficFTP('GeneratePacket',true);
```

Generate an FTP application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgFTP);
```

Generate FTP Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default FTP application traffic pattern object.

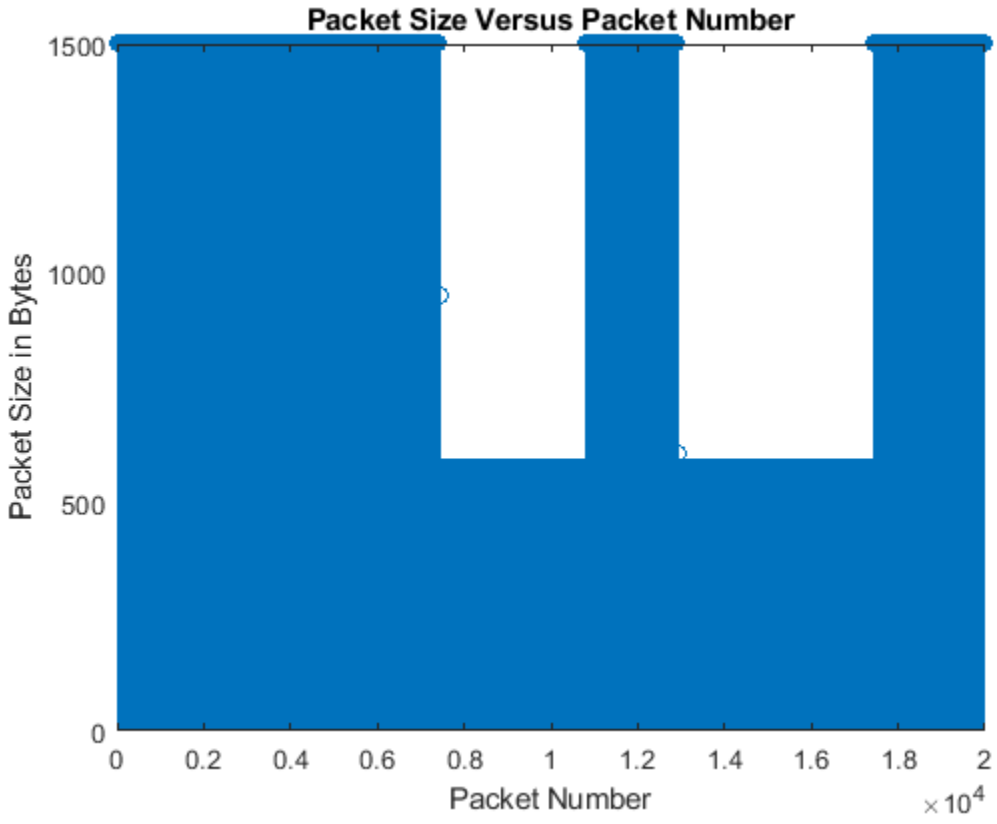
```
cfgFTP = networkTrafficFTP;
```

Generate an FTP application traffic pattern with 20,000 FTP packets.

```
for packetCount = 1:20000
    [dt(packetCount),packetSize(packetCount)] = generate(cfgFTP);
end
```

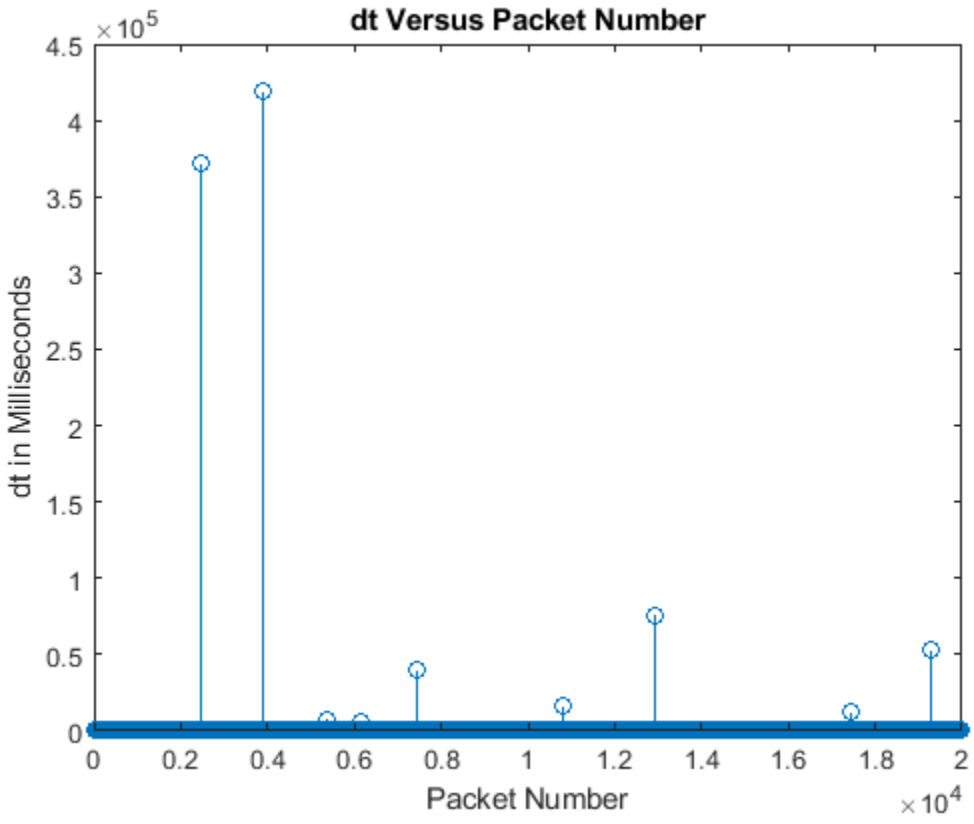
Visualize the FTP packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the FTP packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



References

[1] 3GPP TR 36.814. "Evolved Universal Terrestrial Radio Access (E-UTRA). Further advancements for E-UTRA physical layer aspects". Release 15. *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*. https://www.3gpp.org.

[2] IEEE 802.11-14/0571r12 . "11ax Evaluation Methodology". IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`networkTrafficOnOff` | `networkTrafficVoIP`

Introduced in R2020b

networkTrafficOnOff

On-Off application traffic pattern generator

Description

The `networkTrafficOnOff` object specifies the configuration parameters to generate an On-Off application traffic pattern.

You can use the On-Off application traffic pattern in 5G and WLAN (requires WLAN Toolbox) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgOnOff = networkTrafficOnOff
cfgOnOff = networkTrafficOnOff(Name,Value)
```

Description

`cfgOnOff = networkTrafficOnOff` creates a default On-Off application traffic pattern object.

`cfgOnOff = networkTrafficOnOff(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'GeneratePacket',true` generates an application packet.

Properties

`OnTime` — On state duration
[ ] (default) | positive scalar

On state duration, specified as a positive scalar. This property must be expressed in seconds. To specify a customized value for the On time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the On time.

Data Types: `double`

`OffTime` — Off state duration
[ ] (default) | nonnegative scalar

Off state duration, specified as a nonnegative scalar. This property must be expressed in seconds. To specify a customized value for the Off time, specify this property. If you do not specify this property, the object uses the exponential distribution to calculate the Off time.

Data Types: `double`

`OnExponentialMean` — Exponential distribution mean value to calculate On state duration
1 (default) | positive scalar

Exponential distribution mean value to calculate the On state duration, specified as a positive scalar. This property must be expressed in seconds.

Dependencies

To enable this property, set the `OnTime` property to `[ ]`.

Data Types: `double`

`OffExponentialMean` — Exponential distribution mean value to calculate Off state duration

2 (default) | nonnegative scalar

Exponential distribution mean value to calculate the Off state duration, specified as a nonnegative scalar. This property must be expressed in seconds.

Dependencies

To enable this property, set the `OffTime` property to `[ ]`.

Data Types: `double`

`DataRate` — Packet generation rate during On state

5 (default) | positive scalar

Packet generation rate during the On state, specified as a positive scalar. This property must be expressed in Kbps. If the value of this property is low and the `PacketSize` is large, the object might not generate packets in the On state.

Data Types: `double`

`PacketSize` — Length of packet to be generated

`1500` (default) | positive scalar

Length of the packet to be generated, specified as a positive scalar. This property must be expressed in bytes. If the value of this property is greater than the `DataRate` property value, the object accumulates the data across multiple On times to generate a packet.

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate application packet

`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates an application packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in packet

1500-by-1 vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the `PacketSize` property value, the object truncates the application data. If the size of the application data is less than the `PacketSize` property value, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

generate Generate next FTP, On-Off, or VoIP application traffic packet

Examples

Generate On-Off Application Traffic Pattern

Create a default On-Off application traffic pattern object.

```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern.

```
[dt,packetSize] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern Using On State Mean Value

Create an On-Off application traffic pattern object, specifying the exponentially distributed mean value of the On state.

```
cfgOnOff = networkTrafficOnOff('OnExponentialMean',5);
```

Generate an On-Off application traffic pattern.

```
[dt,packetSize] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern and Data Packet

Create an On-Off application traffic pattern object to generate an On-Off data packet.

```
cfgOnOff = networkTrafficOnOff('GeneratePacket',true);
```

Generate an On-Off application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgOnOff);
```

Generate On-Off Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default On-Off application traffic pattern object.

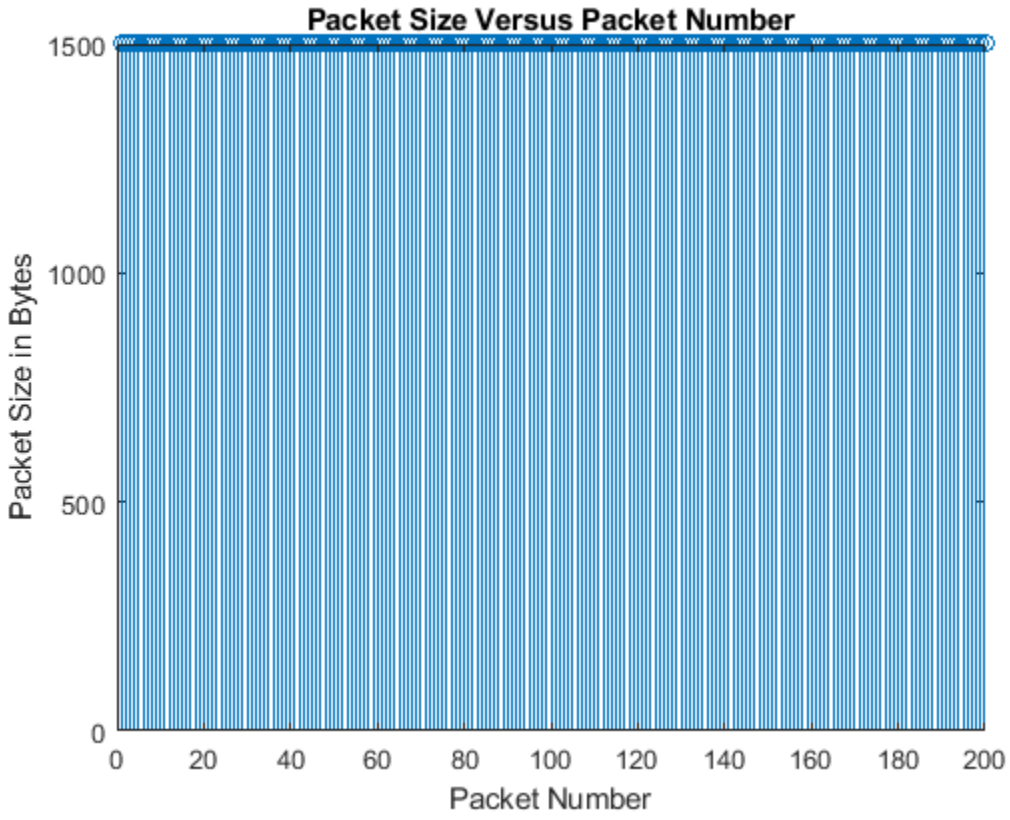
```
cfgOnOff = networkTrafficOnOff;
```

Generate an On-Off application traffic pattern with 200 On-Off packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgOnOff);
end
```

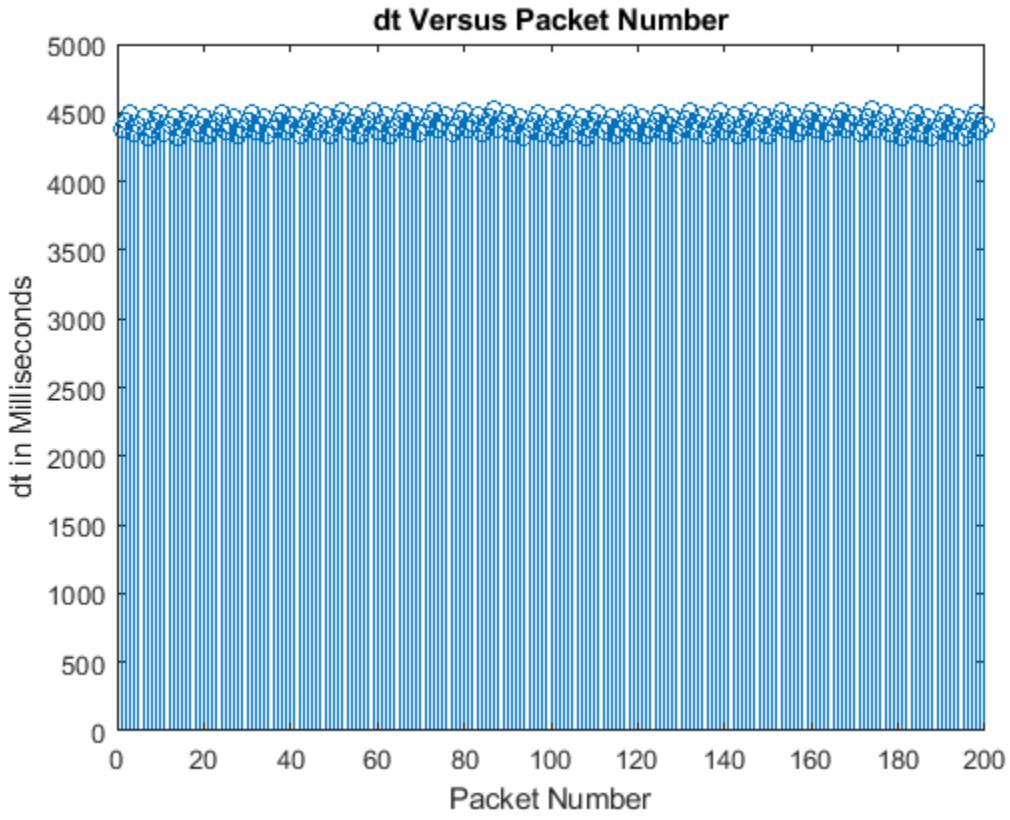
Visualize the On-Off packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the On-Off packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`networkTrafficFTP` | `networkTrafficVoIP`

Introduced in R2020b

networkTrafficVoIP

VoIP application traffic pattern generator

Description

The `networkTrafficVoIP` object specifies the configuration parameters to generate a voice over Internet protocol (VoIP) application traffic pattern based on the IEEE 802.11ax Evaluation Methodology.

You can use the VoIP application traffic pattern in 5G and WLAN (requires WLAN Toolbox) system-level simulations to accurately model the real-world data traffic.

Creation

Syntax

```
cfgVoIP = networkTrafficVoIP
cfgVoIP = networkTrafficVoIP(Name,Value)
```

Description

`cfgVoIP = networkTrafficVoIP` creates a default VoIP application traffic pattern object.

`cfgVoIP = networkTrafficVoIP(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'HasJitter',false` specifies that the VoIP application pattern does not model jitter.

Properties

`ExponentialMean` — Mean value for exponential distribution

`1250` (default) | nonnegative integer

Mean value for the exponential distribution, specified as a nonnegative integer. The object uses this property to calculate the exponentially distributed active or silent state duration in the VoIP traffic.

Data Types: `double`

`HasJitter` — Flag to indicate whether to model jitter

`true` or `1` (default) | `false` or `0`

Flag to indicate whether the object models jitter, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LaplaceScale` — Scale parameter for Laplace distribution

`5.11` (default) | scalar in the range [1, 100]

Scale parameter for the Laplace distribution, specified as a scalar in the range [1, 100]. The object uses this property and the `LaplaceMu` property to calculate the packet arrival delay jitter in milliseconds.

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`LaplaceMu` — Location parameter for Laplace distribution
`0` (default) | scalar in the range [0, 100]

Location parameter for the Laplace distribution, specified as a scalar in the range [0, 100]. The object uses this property and the `LaplaceScale` property to calculate packet arrival delay jitter in milliseconds.

Dependencies

To enable this property, set the `HasJitter` property to `1` (`true`).

Data Types: `double`

`GeneratePacket` — Flag to indicate whether to generate a VoIP packet
`false` or `0` (default) | `true` or `1`

Flag to indicate whether the object generates a VoIP packet, specified as a `logical` `1` (`true`) or `0` (`false`).

Data Types: `logical`

`ApplicationData` — Application data to be added in VoIP packet
36-element column vector of 1s (default) | column vector of integers in the range [0, 255]

Application data to be added in the VoIP packet, specified as a column vector of integers in the range [0, 255]. If the size of the application data is greater than the packet size, the object truncates the application data. If the size of the application data is less than the packet size, the object appends zeros.

Dependencies

To enable this property, set the `GeneratePacket` property to `1` (`true`).

Data Types: `double`

Object Functions

Specific to This Object

generate Generate next FTP, On-Off, or VoIP application traffic packet

Examples

Generate VoIP Application Traffic Pattern

Create a default VoIP application traffic pattern object.

```
cfgVoIP = networkTrafficVoIP;
```

Generate a VoIP application traffic pattern.

```
[dt,packetSize] = generate(cfgVoIP);
```

Generate VoIP Application Traffic Pattern Using Mean Value of Exponential Distribution

Create a VoIP application traffic pattern object, specifying the mean value of the exponential distribution.

```
cfgVoIP = networkTrafficVoIP('ExponentialMean',5);
```

Generate a VoIP application traffic pattern.

```
[dt,packetSize] = generate(cfgVoIP);
```

Generate VoIP Application Traffic Pattern and Data Packet

Create a VoIP application traffic pattern object to generate a VoIP data packet.

```
cfgVoIP = networkTrafficVoIP('GeneratePacket',true);
```

Generate a VoIP application traffic pattern and data packet.

```
[dt,packetSize,packet] = generate(cfgVoIP);
```

Generate VoIP Application Traffic Pattern to Visualize Packet Sizes and Intervals

Create a default VoIP application traffic pattern object.

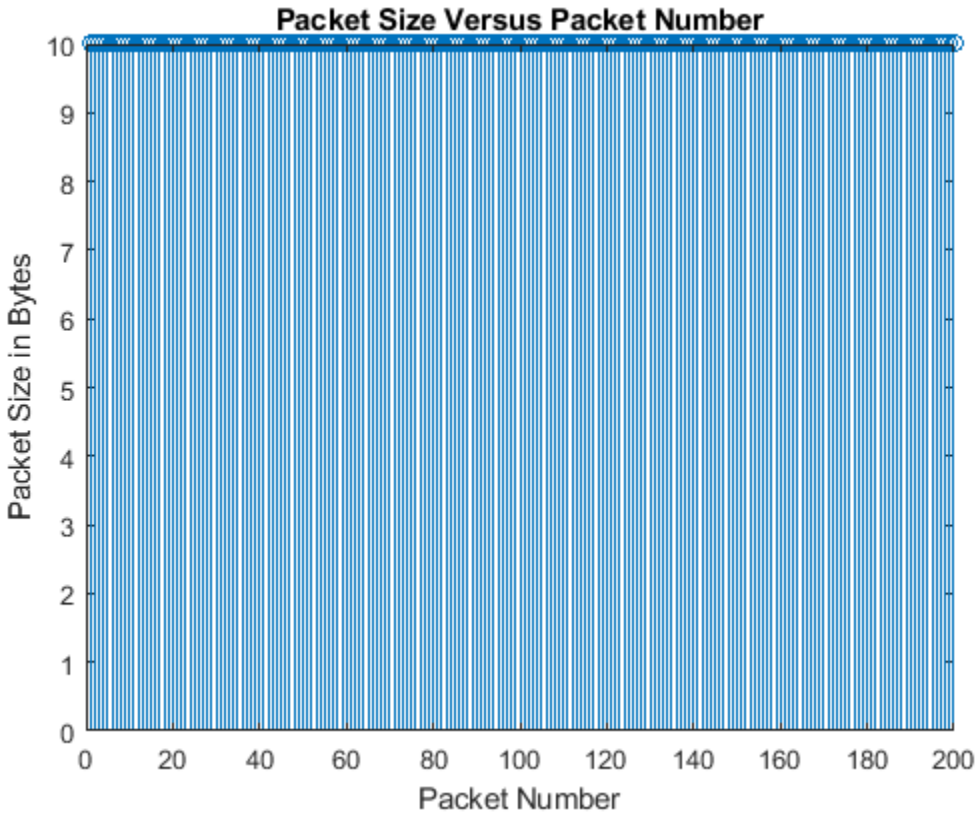
```
cfgVoIP = networkTrafficVoIP;
```

Generate a VoIP application traffic pattern with 200 VoIP packets.

```
for packetCount = 1:200
    [dt(packetCount),packetSize(packetCount)] = generate(cfgVoIP);
end
```

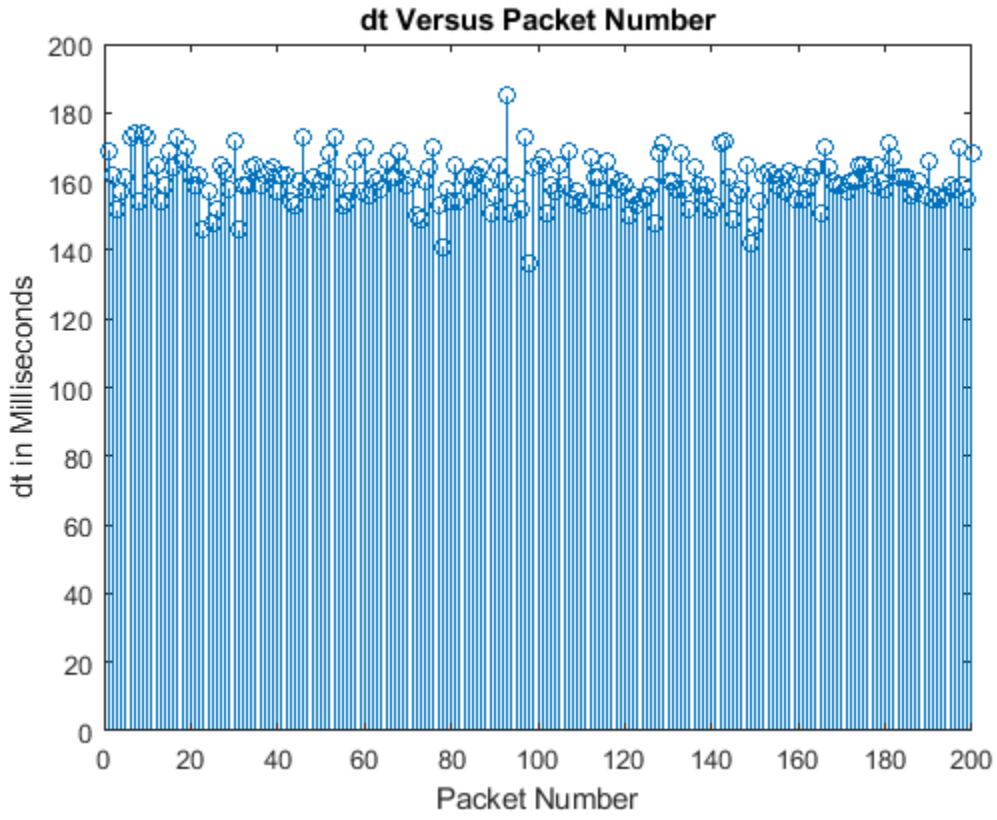
Visualize the VoIP packet sizes.

```
stem(packetSize);
title('Packet Size Versus Packet Number');
xlabel('Packet Number');
ylabel('Packet Size in Bytes');
```



Visualize the VoIP packet intervals.

```
figure;
stem(dt);
title('dt Versus Packet Number');
xlabel('Packet Number');
ylabel('dt in Milliseconds');
```



References

[1] IEEE 802.11-14/0571r12 . "11ax Evaluation Methodology". IEEE P802.11. Wireless LANs. https://www.ieee.org.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`networkTrafficFTP` | `networkTrafficOnOff`

Introduced in R2020b

nrCarrierConfig

Carrier configuration parameters

Description

The `nrCarrierConfig` object sets carrier configuration parameters for a specific OFDM numerology, as defined in TS 38.211 Sections 4.2, 4.3, and 4.4 [1].

The object defines the carrier subcarrier spacing, bandwidth, and offset parameters from point *A*, the center of subcarrier 0 in the common resource block 0 (CRB 0). For a 60 kHz subcarrier spacing, you can specify either normal or extended cyclic prefix. The read-only properties of this object provide the carrier resource grid time-domain dimensions. By default, the object specifies a 10 MHz carrier corresponding to 52 resource blocks (RBs) and 15 kHz subcarrier spacing. You can use the object in slot-oriented processing by specifying the current slot and frame numbers.

Creation

Syntax

```
carrier = nrCarrierConfig
carrier = nrCarrierConfig(Name,Value)
```

Description

`carrier = nrCarrierConfig` creates a carrier configuration object with default properties.

`carrier = nrCarrierConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'SubcarrierSpacing',30,'NSizeGrid',273` specifies a 100 MHz carrier corresponding to 273 RBs and 30 kHz subcarrier spacing. Unspecified properties take their default values.

Properties

`NCellID` — Physical layer cell identity
1 (default) | integer from 0 to 1007

Physical layer cell identity, specified as an integer from 0 to 1007.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing in kHz
`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`CyclicPrefix` — Cyclic prefix length

`'normal'` (default) | `'extended'`

Cyclic prefix length, specified as one of these options.

- `'normal'` — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- `'extended'` — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: `char` | `string`

`NSizeGrid` — Number of RBs in carrier resource grid

`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`NStartGrid` — Start of carrier resource grid relative to CRB 0

`0` (default) | integer from 0 to 2199

Start of carrier resource grid relative to CRB 0, specified as an integer from 0 to 2199. This property is the higher-layer parameter *offsetToCarrier*.

Data Types: `double`

`NSlot` — Slot number

`0` (default) | nonnegative integer

Slot number, specified as a nonnegative integer. You can set `NSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

`NFrame` — System frame number

`0` (default) | nonnegative integer

System frame number, specified as a nonnegative integer. You can set `NFrame` to a value larger than the maximum frame number 1023. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you may have to ensure that the property value is modulo 1024 in a calling code.

Data Types: `double`

`SymbolsPerSlot` — Number of OFDM symbols per slot

`14` (default) | `12`

This property is read-only.

Number of OFDM symbols per slot, specified as `14` for normal cyclic prefix or `12` for extended cyclic prefix. The object sets this property based on the `CyclicPrefix` property.

Data Types: `double`

`SlotsPerSubframe` — Number of slots per 1 ms subframe
1 (default) | 2 | 4 | 8 | 16

This property is read-only.

Number of slots per 1 ms subframe, specified as `1`, `2`, `4`, `8`, or `16`. The object sets this property based on the `SubcarrierSpacing` property values `15`, `30`, `60`, `120`, and `240`, respectively.

Data Types: `double`

`SlotsPerFrame` — Number of slots per 10 ms frame
`10` (default) | `20` | `40` | `80` | `160`

This property is read-only.

Number of slots per 10 ms frame, specified as `10`, `20`, `40`, `80`, or `160`. The object sets this property based on the `SubcarrierSpacing` property values `15`, `30`, `60`, `120`, and `240`, respectively.

Data Types: `double`

Examples

Generate CSI-RS Symbols and Indices for 10 MHz Carrier

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS configuration object with default properties.

```
csirs = nrCSIRSConfig;
```

Generate CSI-RS symbols of `single` data type.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,'OutputDataType','single');
```

Generate resource element indices for CSI-RS.

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs);
```

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers `3` and `5`, symbol locations `13` and `9`, and subcarrier locations `6` and `4`, respectively. For both resources, set the periodicity to `5`, offset to `1`, and density to `'one'`.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                 'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}


info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

```
[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                 'IndexStyle','subscript','OutputResourceFormat','cell')
```

```
ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1


info_ind.ResourceOrder
```

```
ans = 1×2

     2     1
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCSIRS` | `nrCSIRSIndices`

Objects

`nrCSIRSConfig`

Introduced in R2019b

nrCORESETConfig

Control-resource set (CORESET) configuration parameters

Description

The `nrCORESETConfig` object sets CORESET configuration parameters for the physical downlink control channel (PDCCH), as defined in TS 38.211 Section 7.3.2 [1]. Use this object when setting the `CORESET` property of the `nrPDCCHConfig` or `nrDLCarrierConfig` objects.

Creation

Syntax

```
crst = nrCORESETConfig
crst = nrCORESETConfig(Name,Value)
```

Description

`crst = nrCORESETConfig` creates a CORESET configuration object with default properties.

`crst = nrCORESETConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'REGBundleSize',3,'Duration',3` configures the CORESET with a duration of 3 OFDM symbols and resource-element group (REG) bundle size of 3. Unspecified properties take their default values.

Properties

`CORESETID` — CORESET ID

`1` (default) | integer from 0 to 11

CORESET ID, specified as an integer from 0 to 11. When this object and `nrSearchSpaceConfig` object specify the `SearchSpace` and `CORESET` properties, respectively, of the same `nrPDCCHConfig` object, the `CORESETID` properties of these objects must match.

The reference point for the demodulation reference signal (DM-RS) sequence-to-subcarrier resource mapping for CORESET ID 0 is the lowest physical resource block of the CORESET. All other CORESET ID values use the common resource block 0 for the DM-RS reference point.

Data Types: `double`

`Label` — Name of CORESET configuration

`'CORESET1'` (default) | character array | string scalar

Name of CORESET configuration, specified as a character array or string scalar. Use this property to set a mnemonic description to the CORESET configuration.

Data Types: `char` | `string`

`FrequencyResources` — Frequency-domain resources

`[1 1 1 1 1 1 1 1]` (default) | binary row vector

Frequency-domain resources, specified as a binary row vector. An element value of `1` indicates an allocated frequency resource of six resource blocks (RBs). An element value of `0` indicates no allocation. The maximum number of vector elements is 45. Grouping starts from the first RB group in the bandwidth part (BWP). The first vector element corresponds to the first RB group in the BWP.

This property determines the total number of RBs allocated in the frequency domain, which is given by *numRBs* = 6 × `sum(FrequencyResources)`.

Data Types: `double`

`Duration` — CORESET duration

2 (default) | 1 | 3

CORESET duration, in OFDM symbols, specified as `1`, `2`, or `3`.

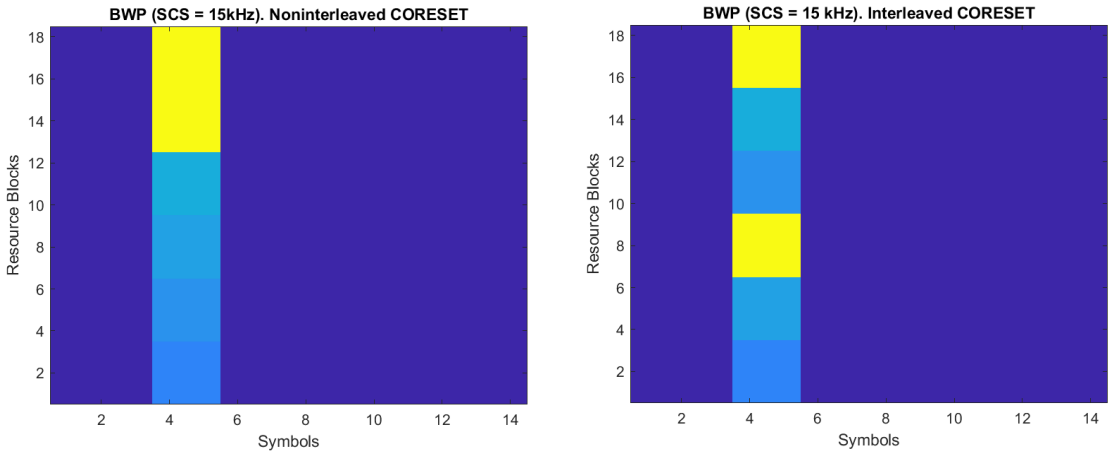
Data Types: `double`

`CCEREGMapping` — CCE-to-REG mapping

`'interleaved'` (default) | `'noninterleaved'`

Control-channel elements (CCE) to REG mapping, specified as `'interleaved'` or `'noninterleaved'`.

These diagrams illustrate the difference between noninterleaved and interleaved CORESETs. Both CORESETS are configured with the `Duration` property set to `2` and the `REGBundleSize` property set to `6`. The `InterleaverSize` property of the interleaved CORESET is set to `2`.



Data Types: `char` | `string`

`REGBundleSize` — Size of REG bundles

6 (default) | 2 | 3

Size of REG bundles, specified as `2`, `3`, or `6`.

- When the `Duration` property is set to `3`, set `REGBundleSize` to `3` or `6`.
- When `Duration` is set to `1` or `2`, set `REGBundleSize` to `2` or `6`.

The number of REGs, *numREGs*, depends on the total number of resource blocks, *numRBs*, allocated in the frequency domain and is given by

numREGs = *numRBs* × `Duration`, where *numRBs* = 6 × `sum(FrequencyResources)`.

When the `CCEREGMapping` property is set to `'interleaved'`, *numREGs* must be a multiple of `REGBundleSize` × `InterleaverSize`.

Dependencies

To enable this property, set the `CCEREGMapping` property to `'interleaved'`.

Data Types: `double`

`InterleaverSize` — Interleaver size

2 (default) | 3 | 6

Interleaver size for interleaved CCE-to-REG mapping, specified as 2, 3, or 6.

The number of REGs, *numREGs*, depends on the total number of resource blocks, *numRBs*, allocated in the frequency domain and is given by

numREGs = *numRBs* × `Duration`, where *numRBs* = 6 × `sum(FrequencyResources)`.

When the `CCEREGMapping` property is set to `'interleaved'`, *numREGs* must be a multiple of `REGBundleSize` × `InterleaverSize`.

Dependencies

To enable this property, set the `CCEREGMapping` property to `'interleaved'`.

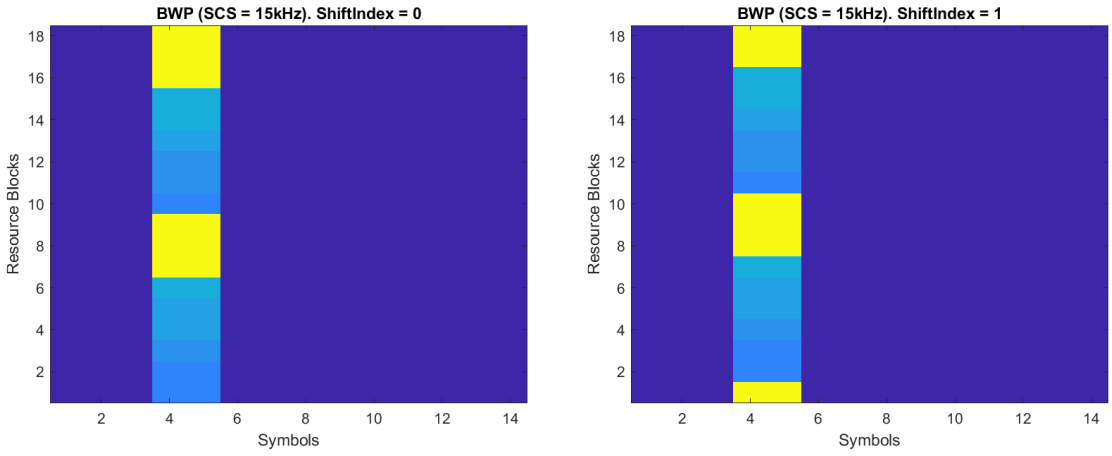
Data Types: `double`

`ShiftIndex` — Shift index

`0` (default) | integer from 0 to 274 | *NCellID* with integer value from 0 to 1007

Shift index, specified as an integer from 0 to 274 or the physical layer cell identity *NCellID* with integer value from 0 to 1007.

For example, these diagrams illustrate the effect of shift index on CORESET mapping. Both CORESETs are configured with the `Duration` property set to 2, the `REGBundleSize` property set to 2, and the `InterleaverSize` property set to 2.



Dependencies

To enable this property, set the `CCEREGMapping` property to `'interleaved'`.

Data Types: `double`

Examples

Generate PDCCH DM-RS Symbols and Indices

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH DM-RS symbols and indices for the specified carrier and PDCCH.

```
[~,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
```

Generate PDCCH and DM-RS Indices Relative to BWP Grid

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
```

```
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH resource element indices and DM-RS symbol indices using 1-based, subscript indexing form relative to the BWP grid.

```
[ind,~,dmrsInd] = nrPDCCHResources(carrier,pdcch,...
    'IndexOrientation','bwp','IndexStyle','subscript');
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPDCCH` | `nrPDCCHResources` | `nrPDCCHSpace`

Objects

`nrCarrierConfig` | `nrDLCarrierConfig` | `nrPDCCHConfig` | `nrSearchSpaceConfig`

Introduced in R2020a

nrCSIRSConfig

CSI-RS configuration parameters

Description

The `nrCSIRSConfig` object sets channel state information reference signal (CSI-RS) configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources, as defined in TS 38.211 Section 7.4.1.5 [1].

Creation

Syntax

```
csirs = nrCSIRSConfig
csirs = nrCSIRSConfig(Name,Value)
```

Description

`csirs = nrCSIRSConfig` creates a CSI-RS configuration object with default properties.

`csirs = nrCSIRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'CSIRSType', {'zp','nzp','zp'},'Density',{'one','dot5odd','three'},'SubcarrierLocations', {0,4,[0 4]}` specifies three CSI-RS resources with different frequency density values and different frequency-domain locations. Unspecified properties take their default values.

Properties

`CSIRSType` — Type of one or more CSI-RS resource configurations

`'nzp'` (default) | `'zp'` | cell array | string scalar | string array

Type of one or more CSI-RS resource configurations, specified as one of these options.

- `'nzp'` — Use this option to specify a single NZP-CSI-RS resource.
- `'zp'` — Use this option to specify a single ZP-CSI-RS resource.
- Cell array with elements `'nzp'` or `'zp'` — Use this option to specify multiple CSI-RS resources.

Alternatively, you can specify this property by using `"nzp"` and `"zp"` as string scalars or as elements of a string array.

The number of CSI-RS resource configurations is equal to the number of values provided for this property.

Data Types: `cell` | `string` | `char`

`CSIRSPeriod` — Slot periodicity and offset of CSI-RS resource

`'on'` (default) | `'off'` | vector of integers | cell array | string scalar | string array

Slot periodicity and offset of the CSI-RS resource, specified as one of these options.

For Single CSI-RS Resource

- `'on'` — Use this option to indicate that the resource is present in all slots.
- `'off'` — Use this option to indicate that the resource is absent in all slots.
- Vector of integers of the form [*Tcsi-rs Toffset*] — Use this option to specify slot periodicity *Tcsi-rs* and offset *Toffset* for scheduling the CSI-RS resource in specific slots.

  Tcsi-rs is 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, or 640. For a particular value of *Tcsi-rs*, the value of *Toffset* is in the range from 0 to *Tcsi-rs*–1.

For Multiple CSI-RS Resources

- Cell array with elements `'on'`, `'off'`, or [*Tcsi-rs Toffset*] — The number of elements in the cell array must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the cell array contains only one element, the specified value applies to all CSI-RS resources.

Alternatively, you can specify this property by using `"on"` and `"off"` as string scalars or as elements of a string array.

This property is the higher-layer parameter *CSI-ResourcePeriodicityAndOffset* or *slotConfig* defined in the *CSI-RS-CellMobility* IE.

Data Types: `cell` | `string` | `char` | `double`

`RowNumber` — Row number of CSI-RS resource
3 (default) | integer from 1 to 18 | vector of integers

Row number of CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 18

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 18 — The number of vector elements must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`Density` — Frequency density of CSI-RS resource
`'one'` (default) | `'three'` | `'dot5even'` | `'dot5odd'` | cell array | string scalar | string array

Frequency density of the CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- `'one'` — This option corresponds to $\rho = 1$ from the specified table.
- `'three'` — This option corresponds to $\rho = 3$ from the specified table.
- `'dot5even'` — This option corresponds to $\rho = 0.5$ from the specified table with even resource block (RB) allocation regarding the common resource block 0 (CRB 0).

- `'dot5odd'` — This option corresponds to ρ = 0.5 from the specified table with odd RB allocation regarding CRB 0.

For Multiple CSI-RS Resources

- Cell array of the character vectors `'one'`, `'three'`, `'dot5even'`, or `'dot5odd'` — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Alternatively, you can specify this property by using `"one"`, `"three"`, `"dot5even"`, and `"dot5odd"` as string scalars or as elements of a string array.

The value of ρ is the higher-layer parameter *density* defined in the *CSI-RS-ResourceMapping* IE or the *CSI-RS-CellMobility* IE.

Data Types: `cell` | `string` | `char`

`SymbolLocations` — Time-domain locations of CSI-RS resource
`0` (default) | integer from 0 to 13 | vector of integers | cell array

Time-domain locations of the CSI-RS resource (l_0 and l_1 values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 13 — This option corresponds to the l_0 value in the specified table.
- Vector of integers of the form [l_0 l_1] or [l_0; l_1], where l_0 and l_1 are the corresponding l_0 and l_1 values in the specified table — The l_1 values are required only in table rows 13, 14, 16, and 17. l_0 is an integer from 0 to 13, and l_1 is an integer from 2 to 12.

For Multiple CSI-RS Resources

- Cell array of l_0 values or vectors of the form [l_0 l_1] or [l_0; l_1] — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

The values of l_0 and l_1 are the higher-layer parameters *firstOFDMSymbolInTimeDomain* and *firstOFDMSymbolInTimeDomain2*, respectively, in the *CSI-RS-ResourceMapping* IE or *CSI-RS-Resource-Mobility* IE.

Data Types: `double`

`SubcarrierLocations` — Frequency-domain locations of CSI-RS resource
`0` (default) | numeric vector | cell array

Frequency-domain locations of the CSI-RS resource (k_i values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Numeric vector with elements 1, 2, 3, 4, or 6 — The vector elements correspond to the possible lengths of subcarrier locations.

For Multiple CSI-RS Resources

- Cell array of numeric vectors with elements 1, 2, 3, 4, or 6 — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`NumRB` — CSI-RS resource bandwidth
52 (default) | integer from 1 to 275 | vector of integers

CSI-RS resource bandwidth, in terms of the number of allocated RBs, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 275

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 275 — The number of vector elements must equal to one or the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *nrOfRBs* in *FrequencyOccupation* IE or *nrOfPRBs* in *CSI-RS-ResourceConfigMobility* IE.

Data Types: `double`

`RBOffset` — Starting RB index of CSI-RS resource allocation
`0` (default) | integer from 0 to 274 | vector of integers

Starting RB index of the CSI-RS resource allocation, relative to the carrier resource grid, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 274

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 274 — The number of vector elements must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

Data Types: `double`

`NID` — Scrambling identity
`0` (default) | integer from 0 to 1023 | vector of integers

Scrambling identity, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 1023

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 1023 — The number of vector elements must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *scramblingID* in *NZP-CSI-RS-Resource* IE or *sequenceGenerationConfig* in *CSI-RS-ResourceConfigMobility* IE.

When the `CSIRSType` property defines only ZP resources, this property is hidden.

Data Types: `double`

`NumCSIRSPorts` — Number of CSI-RS antenna ports
2 (default) | 1 | 4 | 8 | 12 | 16 | 24 | 32 | vector of integers

This property is read-only.

Number of CSI-RS antenna ports, specified as `1`, `2`, `4`, `8`, `12`, `16`, `24`, `32`, or a vector of integers from this list. The object sets this property based on the `RowNumber` property.

Data Types: `double`

`CDMType` — CDM type of CSI-RS resource
`'FD-CDM2'` (default) | `'noCDM'` | `'CDM4'` | `'CDM8'` | cell array

This property is read-only.

CDM type of CSI-RS resource, specified as `'noCDM'`, `'FD-CDM2'`, `'CDM4'`, `'CDM8'`, or a cell array of character vectors from this list. The object sets this property based on the `RowNumber` property.

Data Types: `char`

Examples

Generate ZP and NZP-CSI-RS Symbols and Indices

Create a carrier configuration object, specifying the slot number as `10`.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a CSI-RS resource configuration object for two periodic resources. Specify one NZP resource and one ZP resource with row numbers `3` and `5`, symbol locations `13` and `9`, and subcarrier locations `6` and `4`, respectively. For both resources, set the periodicity to `5`, offset to `1`, and density to `'one'`.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.RowNumber = [3 5];
csirs.Density = {'one','one'};
csirs.SymbolLocations = {13,9};
csirs.SubcarrierLocations = {6,4};
```

Generate CSI-RS symbols and indices for the specified carrier, CSI-RS resource configuration, and output formatting name-value pair arguments. Verify the format of the symbols and indices.

```
[sym,info_sym] = nrCSIRS(carrier,csirs,...
                 'OutputResourceFormat','cell')
```

```
sym=1×2 cell array
    {0x1 double}    {0x1 double}
```

```
info_sym = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}


[ind,info_ind] = nrCSIRSIndices(carrier,csirs,...
                 'IndexStyle','subscript','OutputResourceFormat','cell')

ind=1×2 cell array
    {0x3 uint32}    {0x3 uint32}


info_ind = struct with fields:
      ResourceOrder: [2 1]
           KBarLBar: {{1x1 cell}  {1x2 cell}}
    CDMGroupIndices: {[0]  [0 1]}
             KPrime: {[0 1]  [0 1]}
             LPrime: {[0]  [0]}
```

Verify that the generated outputs are in the order of ZP-CSI-RS resources followed by NZP-CSI-RS resources in terms of the specified `csirs.CSIRSType` indices.

```
info_sym.ResourceOrder
```

```
ans = 1×2

     2     1
```

```
info_ind.ResourceOrder
```

```
ans = 1×2

     2     1
```

Generate and Map CSI-RS Symbols Used for Tracking

Create a carrier configuration object with default properties.

```
carrier = nrCarrierConfig;
```

Create a CSI-RS resource configuration object with CSI-RS parameters set for tracking. Specify four periodic NZP-CSI-RS resources in two consecutive slots. Specify for each slot to contain two periodic NZP-CSI-RS resources with periodicity set to `20`. Set the offset for the first two resources to `0`. Set the offset for the next two resources to `1`. Set the row number to `1` and density to `'three'` for all resources.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp','nzp'};
csirs.CSIRSPeriod = {[20 0],[20 0],[20 1],[20 1]};
csirs.RowNumber = [1 1 1 1];
```

```
csirs.Density = {'three','three','three','three'};
csirs.SymbolLocations = {6,10,6,10};
csirs.SubcarrierLocations = {0,0,0,0};
```

Generate CSI-RS symbols and indices for the default slot number of the carrier configuration object (slot number 0).

```
ind0 = nrCSIRSIndices(carrier,csirs);
sym0 = nrCSIRS(carrier,csirs);
```

Map the symbols to a carrier grid of one slot duration.

```
gridSize = [12*carrier.NSizeGrid carrier.SymbolsPerSlot max(csirs.NumCSIRSPorts)];
slotgrid0 = complex(zeros(gridSize));
slotgrid0(ind0) = sym0;
```

Change the absolute slot number in the carrier configuration from `0` to `1`.

```
carrier.NSlot = 1;
```

Generate CSI-RS symbols and indices for slot number 1.

```
ind1 = nrCSIRSIndices(carrier,csirs);
sym1 = nrCSIRS(carrier,csirs);
```

Map the symbols to another carrier grid of one slot duration.

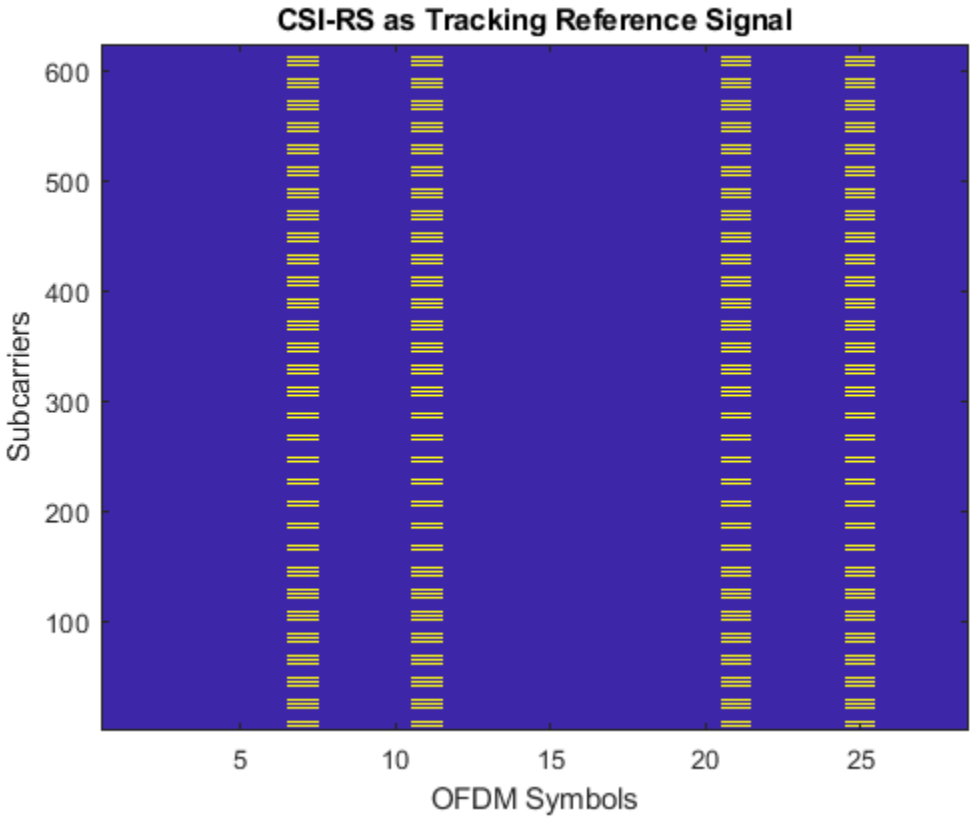
```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

Concatenate the two slots to form the final grid.

```
grid = [slotgrid0 slotgrid1];
```

Plot the grid.

```
imagesc(abs(grid(:,:,1)));
axis xy;
title('CSI-RS as Tracking Reference Signal');
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



References

[1] 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrCSIRS` | `nrCSIRSIndices`

Objects

`nrCarrierConfig`

Introduced in R2019b

nrDLCarrierConfig

5G downlink waveform configuration parameters

Description

The `nrDLCarrierConfig` object sets the parameters of a single-component-carrier 5G downlink waveform. Use this object to configure 5G downlink waveform generation when calling the `nrWaveformGenerator` function.

This object defines these aspects of the downlink waveform:

- Frequency range
- Channel bandwidth
- Cell identity
- Waveform duration
- Subcarrier spacing (SCS) carriers
- Bandwidth parts (BWPs)
- Synchronization signal (SS) burst
- Control-resource sets (CORESETs)
- Search spaces
- Physical downlink control channel (PDCCH) and PDCCH demodulation reference signal (DM-RS)
- Physical downlink shared channel (PDSCH), PDSCH DM-RS, and PDSCH phase-tracking reference signal (PT-RS)
- Channel state information reference signal (CSI-RS)

Creation

Syntax

```
cfgDL = nrDLCarrierConfig
cfgDL = nrDLCarrierConfig(Name,Value)
```

Description

`cfgDL = nrDLCarrierConfig` creates a default single-component-carrier 5G downlink waveform configuration object.

`cfgDL = nrDLCarrierConfig(Name,Value)` sets properties on page 3-37 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'FrequencyRange','FR2'` specifies a downlink waveform for frequency range 2 (FR2).

Properties

`FrequencyRange` — Frequency range

`'FR1'` (default) | `'FR2'`

Frequency range, specified as one of these values.

- `'FR1'` for frequency range 1 (FR1)
- `'FR2'` for frequency range 2 (FR2)

Data Types: `char` | `string`

`ChannelBandwidth` — Downlink channel bandwidth

`50` (default) | `5` | `10` | `15` | `20` | `25` | `30` | `40` | `50` | `60` | `80` | `90` | `100` | `200` | `400`

Downlink channel bandwidth, in MHz, specified as one of these values.

- `5`, `10`, `15`, `20`, `25`, `30`, `40`, `50`, `60`, `80`, `90`, or `100` for FR1
- `50`, `100`, `200`, or `400` for FR2

Set the frequency range with the `FrequencyRange` property.

Data Types: `double`

`NCellID` — Physical layer cell identity

`1` (default) | integer from 0 to 1007

Physical layer cell identity, specified as an integer from 0 to 1007.

Data Types: `double`

`NumSubframes` — Waveform duration

`10` (default) | positive integer

Waveform duration, in number of subframes, specified as a positive integer.

Data Types: `double`

`WindowingPercent` — Windowing percentage relative to FFT length

`0` (default) | real scalar in the range [0 50] | 1-by-6 vector of real numbers in the range [0 50] | `[]`

Windowing percentage relative to fast Fourier transform (FFT) length, specified as one of these values.

- Real scalar in the range [0 50] — The object sets the same windowing percentage for all combinations of SCS and cyclic prefix.
- [*w1 w2 w3 w4 w5 w6*] real vector with element values in the range [0 50] — The object sets individual windowing percentage for the different SCS and cyclic prefix combinations.
  - *w1* specifies the windowing percentage for 15 kHz SCS
  - *w2* specifies the windowing percentage for 30 kHz SCS
  - *w3* specifies the windowing percentage for 60 kHz SCS and normal cyclic prefix
  - *w4* specifies the windowing percentage for 60 kHz SCS and extended cyclic prefix
  - *w5* specifies the windowing percentage for 120 kHz SCS

- *w6* specifies the windowing percentage for 240 kHz SCS

- `[]` — The object automatically selects windowing percentage based on other object properties. For more information, see `nrOFDMModulate`.

This property configures the number of time-domain samples, as a percentage of the FFT length, over which windowing and overlapping of OFDM symbols take place.

Data Types: `double`

`SCSCarriers` — One or more SCS carrier configurations
cell array consisting of a default `nrSCSCarrierConfig` object (default) | cell array of `nrSCSCarrierConfig` objects

One or more SCS carrier configurations, specified as a cell array of `nrSCSCarrierConfig` objects. Because this property configures the subcarrier spacing and grid size of each numerology, each `nrSCSCarrierConfig` object in the cell array must have a unique `SubcarrierSpacing` property value.

`BandwidthParts` — One or more BWP configurations
cell array consisting of a default `nrWavegenBWPConfig` object (default) | cell array of `nrWavegenBWPConfig` objects

One or more BWP configurations, specified as a cell array of `nrWavegenBWPConfig` objects.

`SSBurst` — SS burst configuration
default `nrWavegenSSBurstConfig` object (default) | `nrWavegenSSBurstConfig` object

SS burst configuration, specified as an `nrWavegenSSBurstConfig` object. Use this property to configure the SS burst and blocks.

`CORESET` — One or more CORESET configurations
cell array consisting of a default `nrCORESETConfig` object (default) | cell array of `nrCORESETConfig` objects

One or more CORESET configurations, specified as a cell array of `nrCORESETConfig` objects. Use this property to specify different CORESET configurations for multiple search spaces and PDCCH.

`SearchSpaces` — One or more search space set configurations
cell array consisting of a default `nrSearchSpaceConfig` object (default) | cell array of `nrSearchSpaceConfig` objects

One or more search space set configurations, specified as a cell array of `nrSearchSpaceConfig` objects. Use this property to specify different search space set configurations for linking to a CORESET and for multiple PDCCH.

`PDCCH` — One or more PDCCH configurations
cell array consisting of a default `nrWavegenPDCCHConfig` object (default) | cell array of `nrWavegenPDCCHConfig` objects

One or more PDCCH configurations, specified as a cell array of `nrWavegenPDCCHConfig` objects. Use this property to configure different PDCCH and associated DM-RS.

`PDSCH` — One or more PDSCH configurations
cell array consisting of a default `nrWavegenPDSCHConfig` object (default) | cell array of `nrWavegenPDSCHConfig` objects

One or more PDSCH configurations, specified as a cell array of `nrWavegenPDSCHConfig` objects. Use this property to configure different PDSCH and associated DM-RS and PT-RS.

`CSIRS` — One or more CSI-RS configurations
cell array consisting of a default `nrWavegenCSIRSConfig` object (default) | cell array of `nrWavegenCSIRSConfig` objects

One or more CSI-RS configurations, specified as a cell array of `nrWavegenCSIRSConfig` objects.

Examples

Configure and Generate Single-User 5G Downlink Waveform

Create an SCS carrier configuration object with the default SCS of 15 kHz and 100 resource blocks.

```
carrier = nrSCSCarrierConfig('NSizeGrid',100);
```

Create a customized BWP configuration object for the SCS carrier.

```
bwp = nrWavegenBWPConfig('NStartBWP',carrier.NStartGrid+10);
```

Create an SS burst configuration object with block pattern Case A.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create a PDCCH configuration object, specifying an aggregation of size two and the fourth candidate for the PDCCH instance.

```
pdcch = nrWavegenPDCCHConfig('AggregationLevel',2,'AllocatedCandidate',4);
```

Create a CORESET configuration object, specifying four frequency resources and a duration of three OFDM symbols.

```
coreset = nrCORESETConfig;
coreset.FrequencyResources = [1 1 1 1];
coreset.Duration = 3;
```

Create a search space set configuration object, specifying two aggregation levels.

```
ss = nrSearchSpaceConfig;
ss.NumCandidates = [8 4 0 0 0];
```

Create a PDSCH configuration object, specifying the modulation scheme and the target code rate. Enable the PDSCH PT-RS.

```
pdsch = nrWavegenPDSCHConfig( ...
    'Modulation','16QAM','TargetCodeRate',658/1024,'EnablePTRS',true);
```

Create a PDSCH DM-RS and a PDSCH PT-RS configuration object with the specified property values.

```
dmrs = nrPDSCHDMRSConfig('DMRSTypeAPosition',3);
pdsch.DMRS = dmrs;
ptrs = nrPDSCHPTRSConfig('TimeDensity',2);
pdsch.PTRS = ptrs;
```

Create a CSI-RS configuration object with the specified property values.

```
csirs = nrWavegenCSIRSConfig('RowNumber',4,'RBOffset',10);
```

Create a single-user 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',{carrier}, ...
    'BandwidthParts',{bwp}, ...
    'SSBurst',ssb, ...
    'CORESET',{coreset}, ...
    'SearchSpaces',{ss}, ...
    'PDCCH',{pdcch}, ...
    'PDSCH',{pdsch}, ...
    'CSIRS',{csirs});
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

Configure and Generate Multiuser 5G Downlink Waveform

Create two SCS carrier configuration objects with mixed numerologies and custom numbers of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two custom BWP configuration objects, one for each of the carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create an SS burst configuration object with block pattern Case A, corresponding to an SCS of 15 kHz.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case A');
```

Create two PDCCH configuration objects.

```
pdcch = {
    nrWavegenPDCCHConfig('SearchSpaceID',1,'BandwidthPartID',1,'RNTI',1,'DMRSScramblingID',1), .
    nrWavegenPDCCHConfig('SearchSpaceID',2,'BandwidthPartID',2,'RNTI',2,'DMRSScramblingID',2, ..
        'AggregationLevel',4)};
```

Create two CORESET configuration objects and two search space set configuration objects for the two PDCCH.

```
coreset = {
    nrCORESETConfig('CORESETID',1,'FrequencyResources',[1 1 1 1 1 0 0 0 0 0 1],'Duration',3), ..
```

```
    nrCORESETConfig('CORESETID',2,'FrequencyResources',[0 0 0 0 0 0 0 0 1 1])};

ss = {
    nrSearchSpaceConfig('SearchSpaceID',1,'CORESETID',1,'StartSymbolWithinSlot',4), ...
    nrSearchSpaceConfig('SearchSpaceID',2,'CORESETID',2,'NumCandidates',[8 8 4 0 0])};
```

Create two PDSCH configuration objects with mixed modulation schemes.

```
pdsch = {
    nrWavegenPDSCHConfig('BandwidthPartID',1,'Modulation','16QAM','RNTI',1,'NID',1), ...
    nrWavegenPDSCHConfig('BandwidthPartID',2,'Modulation','QPSK','RNTI',2,'NID',2, ...
            'PRBSet', 50:59)};
```

Create two CSI-RS configuration objects.

```
 csirs = {
     nrWavegenCSIRSConfig('BandwidthPartID',1,'RowNumber',2,'RBOffset',10), ...
     nrWavegenCSIRSConfig('BandwidthPartID',2,'Density','three','RowNumber',4)};
```

Create a multiuser 5G downlink waveform configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'FrequencyRange','FR1', ...
    'ChannelBandwidth',40, ...
    'NumSubframes',20, ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'SSBurst',ssb, ...
    'CORESET',coreset, ...
    'SearchSpaces',ss, ...
    'PDCCH',pdcch, ...
    'PDSCH',pdsch, ...
    'CSIRS',csirs);
```

Generate a 5G downlink waveform using the specified configuration.

```
waveform = nrWaveformGenerator(cfgDL);
```

See Also

Functions
nrWaveformGenerator

Objects
nrCORESETConfig | nrPDSCHDMRSConfig | nrPDSCHPTRSConfig | nrSCSCarrierConfig | nrSearchSpaceConfig | nrWavegenBWPConfig | nrWavegenCSIRSConfig | nrWavegenPDCCHConfig | nrWavegenPDSCHConfig | nrWavegenSSBurstConfig

Introduced in R2020b

nrPDCCHConfig

PDCCH configuration parameters

Description

The `nrPDCCHConfig` object sets physical downlink control channel (PDCCH) configuration parameters, as defined in TS 38.211 Section 7.3.2 [1] and TS 38.213 Section 10 [2].

Creation

Syntax

```
pdcch = nrPDCCHConfig
pdcch = nrPDCCHConfig(Name,Value)
```

Description

`pdcch = nrPDCCHConfig` creates a PDCCH configuration object with default properties.

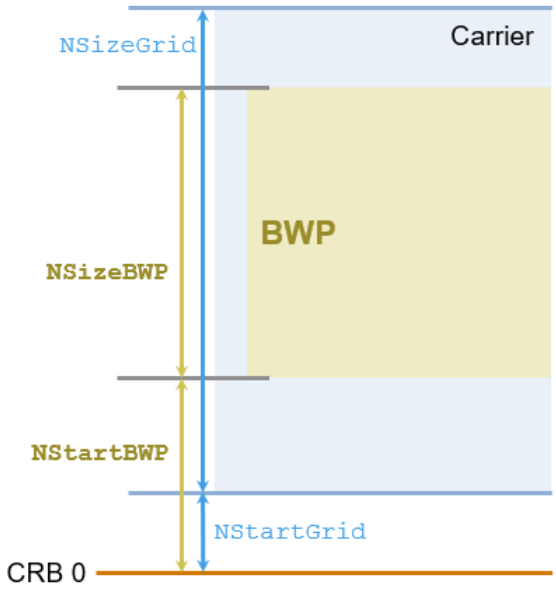
`pdcch = nrPDCCHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NSizeBWP',36` configures the PDCCH with a bandwidth part (BWP) consisting of 36 resource blocks (RBs). Unspecified properties take their default values.

Properties

`NStartBWP` — Start of BWP resource grid relative to CRB 0

`0` (default) | nonnegative integer

Start of BWP resource grid relative to the common resource block 0 (CRB 0), specified as a nonnegative integer. Set this property relative to the carrier such that the property value is in this range: `NStartGrid` ≤ `NStartBWP` < (`NStartGrid` + `NSizeGrid`), where `NStartGrid` and `NSizeGrid` are properties of the carrier configuration object `nrCarrierConfig`. This figure shows where in the carrier the BWP is located in terms of this property and the `NSizeBWP` property.



Data Types: `double`

`NSizeBWP` — Number of RBs in BWP resource grid
48 (default) | integer from 1 to 275

Number of RBs in BWP resource grid, specified as an integer from 1 to 275. This property must be less than or equal to the size of the carrier, which is specified by the `NSizeGrid` property of the carrier configuration object `nrCarrierConfig`.

Data Types: `double`

`CORESET` — CORESET configuration
`nrCORESETConfig` object with default properties (default)

Control-resource set (CORESET) configuration, specified as an `nrCORESETConfig` object.

`SearchSpace` — Search space set configuration
`nrSearchSpaceConfig` object with default properties (default)

Search space set configuration, specified as an `nrSearchSpaceConfig` object.

`RNTI` — Radio network temporary identifier
1 (default) | integer from 0 to 65,519

Radio network temporary identifier (RNTI), specified as an integer from 0 to 65,519.

- If the higher layer parameter *pdcch-DMRS-ScramblingID* is configured, `RNTI` is the cell radio network temporary identifier (C-RNTI) with integer value from 1 to 65,519.
- If *pdcch-DMRS-ScramblingID* is not configured, `RNTI` is `0`.

Data Types: `double`

`DMRSScramblingID` — PDCCH DM-RS scrambling identity
2 (default) | integer from 0 to 65,535 | [ ]

PDCCH demodulation reference signal (DM-RS) scrambling identity, specified as an integer from 0 to 65,535 if the higher layer parameter *pdcch-DMRS-ScramblingID* is configured or as `[]` if *pdcch-DMRS-ScramblingID* is not configured. When you specify this property as `[]`, the object sets the PDCCH DM-RS scrambling identity to the physical layer cell identity specified by the `NCellID` property of the carrier.

Data Types: `double`

`AggregationLevel` — PDCCH aggregation level
8 (default) | 1 | 2 | 4 | 16

PDCCH aggregation level, specified as 1, 2, 4, 8, or 16.

Data Types: `double`

`AllocatedCandidate` — Candidate used for PDCCH instance
1 (default) | integer from 1 to 8

Candidate used for PDCCH instance, specified as an integer from 1 to 8. The value of this property is an index from the set of candidates specified for the aggregation level by the `SearchSpace.NumCandidates` property.

Data Types: `double`

Examples

Generate PDCCH DM-RS Symbols and Indices

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH DM-RS symbols and indices for the specified carrier and PDCCH.

```
[~,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
```

Generate PDCCH and DM-RS Indices Relative to BWP Grid

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the PDCCH with the specified bandwidth part and CORESET.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.AggregationLevel = 16;
```

Generate PDCCH resource element indices and DM-RS symbol indices using 1-based, subscript indexing form relative to the BWP grid.

```
[ind,~,dmrsInd] = nrPDCCHResources(carrier,pdcch,...
    'IndexOrientation','bwp','IndexStyle','subscript');
```

Generate PDCCH DM-RS Symbols for All Candidates and Aggregation Levels

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
```

```
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS symbols for all candidates and aggregation levels.

```
[~,allDMRS] = nrPDCCHSpace(carrier,pdcch)
```

```
allDMRS=5×1 cell array
    { 18x5 double}
    { 36x5 double}
    { 72x3 double}
    {144x2 double}
    {288x1 double}
```

Verify that the number of generated candidates for the PDCCH DM-RS symbols at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRS{1},2) ...
    size(allDMRS{2},2) ...
    size(allDMRS{3},2) ...
    size(allDMRS{4},2) ...
    size(allDMRS{5},2)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPDCCH` | `nrPDCCHResources` | `nrPDCCHSpace`

Objects

`nrCORESETConfig` | `nrCarrierConfig` | `nrSearchSpaceConfig`

Introduced in R2020a

nrPDSCHConfig

PDSCH configuration parameters

Description

The `nrPDSCHConfig` object sets physical downlink shared channel (PDSCH) configuration parameters, as defined in TS 38.211 Sections 7.3.1, 7.4.1.1, and 7.4.1.2 [1].

This object defines all the properties involved in the PDSCH processing chain, including scrambling, symbol modulation, layer mapping, virtual resource blocks (VRB) to physical resource blocks (PRB) interleaving, and resource element (RE) mapping with the reserved resources patterns. The object also contains properties of the associated physical reference signals, such as demodulation reference signal (DM-RS) and phase tracking reference signal (PT-RS).

By default, the object configures a physical downlink shared channel occupying a 10 MHz bandwidth at, 15 kHz subcarrier spacing (52 resource blocks) and spanning over 14 OFDM symbols in a slot.

Creation

Syntax

```
pdsch = nrPDSCHConfig
pdsch = nrPDSCHConfig(Name,Value)
```

Description

`pdsch = nrPDSCHConfig` creates a PDSCH configuration object with default properties.

`pdsch = nrPDSCHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NSizeBWP',200,'NumLayers',7` specifies 200 PRBs in the bandwidth part (BWP) and 7 transmission layers. Unspecified properties take their default values.

Properties

`NSizeBWP` — Number of PRBs in BWP

`[]` (default) | integer from 1 to 275

Number of PRBs in BWP, specified as an integer from 1 to 275. Use `[]` to set this property to the `NSizeGrid` property of the `nrCarrierConfig` object.

Data Types: `double`

`NStartBWP` — Starting PRB index of BWP relative to CRB 0

`[]` (default) | integer from 0 to 2473

Starting PRB index of BWP relative to common resource block (CRB) 0, specified as an integer from 0 to 2473. Use `[]` to set this property to the `NStartGrid` property of the `nrCarrierConfig` object.

Data Types: `double`

`ReservedPRB` — Reserved PRBs and OFDM symbols pattern in BWP
default `nrPDSCHReservedConfig` object (default) | cell array of `nrPDSCHReservedConfig` objects

Reserved PRBs and OFDM symbols pattern in the BWP, specified as a cell array of `nrPDSCHReservedConfig` objects.

Data Types: `cell`

`ReservedRE` — Reserved RE indices within BWP
`[]` (default) | vector of nonnegative integers

Reserved RE indices within the BWP, specified as a vector of nonnegative integers. This property specifies RE indices (0-based) that are unavailable for a PDSCH due to the channel state information reference signal (CSI-RS) or cell-specific reference signal being present in a particular slot.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. For one codeword, specify the modulation scheme as a character vector or string scalar. If two codewords are present (`NumLayers` > 4), the same modulation scheme applies to both codewords or you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Example: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]` specifies different modulation schemes for two codewords.

Data Types: `char` | `string` | `cell`

`NumLayers` — Number of transmission layers
`1` (default) | integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8.

- For one codeword, specify an integer from 1 to 4.
- For two codewords, specify an integer from 5 to 8.

Data Types: `double`

`MappingType` — Mapping type
`'A'` (default) | `'B'`

Mapping type of the physical shared channel, specified as `'A'` or `'B'`.

Data Types: `char` | `string`

`SymbolAllocation` — OFDM symbol allocation

`[0 14]` (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When this property is set to `[]` or second element in this two-element vector of nonnegative integers is `0`, the symbol allocation is empty.

Data Types: `double`

`PRBSet` — PRB allocation

`[0:51]` (default) | vector of integers from 0 to 274

PRB allocation of the PDSCH in the BWP, specified as a vector of integers from 0 to 274.

Data Types: `double`

`VRBToPRBInterleaving` — Enable VRB-to-PRB interleaving

`0` (default) | `1`

Enable VRB-to-PRB interleaving, specified as one of these values.

- `0` — Disable VRB-to-PRB interleaving.
- `1` — Enable VRB-to-PRB interleaving.

Data Types: `double` | `logical`

`VRBBundleSize` — VRB bundle size

`2` (default) | `4`

VRB bundle size, in terms of the number of PRBs for VRB-to-PRB interleaving, specified as `2` or `4`.

Dependencies

To enable this property, set the `VRBToPRBInterleaving` property to `1`.

Data Types: `double`

`NID` — PDSCH scrambling identity

`[]` (default) | integer from 0 to 1023

PDSCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher layer parameter *dataScramblingIdentityPDSCH* is configured, `NID` must be in the range from 0 to 1023.
- If the higher layer parameter *dataScramblingIdentityPDSCH* is not configured, `NID` must be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PDSCH scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier.

Data Types: `double`

RNTI — Radio network temporary identifier
1 (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

DMRS — PDSCH DM-RS configuration parameters
`nrPDSCHDMRSConfig` object with default properties (default) | `nrPDSCHDMRSConfig` object

PDSCH DM-RS configuration parameters, specified as an `nrPDSCHDMRSConfig` object. This property relates to the demodulation reference signal configuration and contains all properties of the specified `nrPDSCHDMRSConfig` object.

EnablePTRS — Enable PT-RS
`0` (default) | `1`

Enable PT-RS, specified as one of these values.

- `0` — Disable PT-RS configuration.
- `1` — Enable PT-RS configuration.

Data Types: `double` | `logical`

PTRS — PDSCH PT-RS configuration parameters
`nrPDSCHPTRSConfig` object with default properties (default) | `nrPDSCHPTRSConfig` object

PDSCH PT-RS configuration, specified as an `nrPDSCHPTRSConfig` object. This property relates to the phase tracking reference signal configuration and contains all properties of the specified `nrPDSCHPTRSConfig` object.

NumCodewords — Number of codewords
1 (default) | 2

This property is read-only.

Number of codewords, specified as a `1` or `2`. This property is updated based on the `DMRSPortSet` property of `nrPDSCHDMRSConfig` object. Use `NumLayers` property to calculate the number of codewords, when `DMRSPortSet` property is empty.

Data Types: `double`

Examples

Create PDSCH Configuration Object

Create a physical downlink shared channel (PDSCH) configuration object that occupies a bandwidth of 10 MHz bandwidth with 15 kHz subcarrier spacing.

Specify 52 PRBs in the bandwidth part (BWP), a scrambling identity of 750, and a 16-QAM modulation scheme. Enable VRB-to-PRB interleaving and PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.NSizeBWP = 52;
```

```
pdsch.NID = 750;
pdsch.Modulation = '16QAM';
pdsch.VRBToPRBInterleaving = 1;
pdsch.EnablePTRS = 1;
disp(pdsch)
```

```
  nrPDSCHConfig with properties:

               NSizeBWP: 52
              NStartBWP: []
            ReservedPRB: {[1x1 nrPDSCHReservedConfig]}
             ReservedRE: []
             Modulation: '16QAM'
              NumLayers: 1
            MappingType: 'A'
        SymbolAllocation: [0 14]
                 PRBSet: [1x52 double]
    VRBToPRBInterleaving: 1
           VRBBundleSize: 2
                     NID: 750
                    RNTI: 1
                    DMRS: [1x1 nrPDSCHDMRSConfig]
              EnablePTRS: 1
                    PTRS: [1x1 nrPDSCHPTRSConfig]

   Read-only properties:
            NumCodewords: 1
```

Create PDSCH Object to Configure Two Reserved PRB Patterns

Create a PDSCH configuration object with two reserved PRB patterns.

```
pdsch = nrPDSCHConfig('ReservedPRB',{nrPDSCHReservedConfig,nrPDSCHReservedConfig});
```

For each reserved PRB pattern, specify the reserved PRB indices in the BWP, the OFDM symbols associated with those reserved PRBs, and the period for the number of slots in the pattern.

```
pdsch.ReservedPRB{1}.PRBSet = (0:15);
pdsch.ReservedPRB{1}.SymbolSet = (5:6);
pdsch.ReservedPRB{1}.Period = 5;
pdsch.ReservedPRB{2}.PRBSet = (0:23);
pdsch.ReservedPRB{2}.SymbolSet = [2:4 7:9];
pdsch.ReservedPRB{2}.Period = 3;
```

Display the two PRB patterns.

```
PRBPattern1 = pdsch.ReservedPRB{1}
```

```
PRBPattern1 =
  nrPDSCHReservedConfig with properties:

       PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
    SymbolSet: [5 6]
       Period: 5
```

```
PRBPattern2 = pdsch.ReservedPRB{2}

PRBPattern2 =
  nrPDSCHReservedConfig with properties:

       PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
    SymbolSet: [2 3 4 7 8 9]
       Period: 3
```

Generate PDSCH Symbols and Indices

Create a carrier configuration object with default properties. Specify the physical layer cell identity as 42 and slot number as 10.

```
carrier = nrCarrierConfig;
carrier.NCellID = 42;
carrier.NSlot = 10;
```

Create a PDSCH configuration object with a 16-QAM modulation scheme. Set the radio network temporary identifier to 1005, size of the BWP to 25, starting PRB index of the BWP to 10, and PRB set to occupy the whole BWP.

```
pdsch = nrPDSCHConfig;
pdsch.Modulation = '16QAM';
pdsch.RNTI = 1005;
pdsch.NID = []; % Set NID equal to the NCellID property of carrier
pdsch.NSizeBWP = 25;
pdsch.NStartBWP = 10;
pdsch.PRBSet = 0:pdsch.NSizeBWP-1;
```

Generate PDSCH indices in subscript form and set the index orientation to bandwidth part.

```
[ind,info] = nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')

ind = 3900x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
    ⋮


info = struct with fields:
                G: 15600
               Gd: 3900
        NREPerPRB: 156
    DMRSSymbolSet: 2
```

```
    PTRSSymbolSet: [1x0 double]
```

Generate PDSCH symbols of data type single.

```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPDSCH(carrier,pdsch,cws,'OutputDataType','single')
```

```
sym = 3900x1 single column vector

  -0.9487 + 0.9487i
  -0.9487 - 0.9487i
  -0.3162 - 0.9487i
   0.9487 - 0.3162i
  -0.9487 + 0.3162i
   0.3162 + 0.9487i
   0.3162 + 0.9487i
  -0.3162 + 0.3162i
   0.3162 + 0.3162i
   0.9487 - 0.3162i
      ⋮
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPDSCH` | `nrPDSCHDMRS` | `nrPDSCHDMRSIndices` | `nrPDSCHDecode` | `nrPDSCHIndices` | `nrPDSCHPTRS` | `nrPDSCHPTRSIndices`

Objects

`nrCarrierConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig` | `nrPDSCHReservedConfig`

Introduced in R2020a

nrPDSCHDMRSConfig

PDSCH DM-RS configuration parameters

Description

The `nrPDSCHDMRSConfig` object sets demodulation reference signal (DM-RS) configuration parameters for a physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.1 [1]. Use this object when setting the `DMRS` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects.

The object defines the properties of PDSCH DM-RS symbols and indices generation and the resource elements pattern not used for data in DM-RS symbol locations. The read-only properties of this object provide the DM-RS subcarrier locations within a resource block (RB), code division multiplexing (CDM) groups, and time and frequency weights for DM-RS symbols. By default, the object specifies a single symbol DM-RS at symbol index 2 (0-based) with configuration type 1 and antenna port 0.

Creation

Syntax

```
dmrs = nrPDSCHDMRSConfig
dmrs = nrPDSCHDMRSConfig(Name,Value)
```

Description

`dmrs = nrPDSCHDMRSConfig` creates a DM-RS configuration object for a PDSCH with default properties.

`dmrs = nrPDSCHDMRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'DMRSConfigurationType',1,'DMRSLength',2` specifies a double-symbol DM-RS with configuration type 1. Unspecified properties take their default values.

Properties

`DMRSConfigurationType` — DM-RS configuration type
`1` (default) | `2`

DM-RS configuration type, specified as `1` or `2`. This property is the higher-layer parameter *dmrs-Type*.

Data Types: `double`

`DMRSReferencePoint` — Reference point for DM-RS sequence to subcarrier resource mapping
`CRB0` (default) | `PRB0`

Reference point for the DM-RS sequence to subcarrier resource mapping, specified as one of these options.

- `PRB0` — When the reference point is subcarrier 0 of the physical resource block 0 (PRB 0) of the bandwidth part (BWP). Use this option when PDSCH is signalled by control resource set 0 (CORESET 0). For this case, the BWP parameters must align with CORSET 0.
- `CRB0` — When the reference point is subcarrier 0 of the common resource block 0 (CRB 0)

Data Types: `char` | `string`

`DMRSTypeAPosition` — Position of first DM-RS OFDM symbol
2 (default) | 3

Position of first DM-RS OFDM symbol, provided by higher layer parameter *dmrs-TypeA-Position*, specified as `2` or `3`.

This property only applies when the `MappingType` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects is set to `'A'`.

Data Types: `double`

`DMRSAdditionalPosition` — Maximum number of DM-RS additional positions
`0` (default) | `1` | `2` | `3`

Maximum number of DM-RS additional positions, specified as `0`, `1`, `2`, or `3`. This property is the higher layer parameter *dmrs-AdditionalPosition*.

Data Types: `double`

`DMRSLength` — Number of consecutive front-loaded DM-RS OFDM symbols
1 (default) | 2

Number of consecutive front-loaded DM-RS OFDM symbols, specified as `1` (single-symbol DM-RS) or `2` (double-symbol DM-RS).

Data Types: `double`

`CustomSymbolSet` — DM-RS OFDM symbol locations
`[]` (default) | integer from 0 to 13 | vector of nonnegative integers

DM-RS OFDM symbol locations that are 0-based, specified as one of these options.

- Integer from 0 to 13 — For one DM-RS symbol
- Vector of nonnegative integers from 0 to 13 — For multiple DM-RS symbols

Each input symbol location is assumed to be a single-symbol DM-RS within the physical shared channel symbol allocation.

The default value, `[]`, corresponds to the DM-RS symbol locations, as defined in TS 38.211 Table 7.4.1.1.2-3 or 7.4.1.1.2-4. Setting this property overrides the corresponding DM-RS symbol locations in these standard lookup tables.

Data Types: `double`

`DMRSPortSet` — DM-RS antenna ports
`[]` (default) | integer scalar | vector of nonnegative integers

DM-RS antenna ports, specified as one of these options.

- Integer from 0 to 11 — For a single antenna port
- Vector of nonnegative integers from 0 to 11 — For multiple antenna ports

Nominal antenna ports supported depend on `DMRSLength` and `DMRSConfigurationType` property values.

DMRSLength Value	DMRSConfigurationType Value	Nominal Range of Antenna Ports Supported
1	1	[0, 3]
	2	[0, 5]
2	1	[0, 7]
	2	[0, 11]

The default value, `[]`, implies that `DMRSPortSet` is in the range from 0 to `NumLayers`– 1, where `NumLayers` is a property of `nrPDSCHConfig` or `nrWavegenPDSCHConfig`.

Data Types: `double`

`NIDNSCID` — DM-RS scrambling identity
`[]` (default) | integer from 0 to 65,535

DM-RS scrambling identity, specified as one of these options.

- Integer from 0 to 65,535 — Use this option when the higher layer parameter *scramblingID0/ scramblingID1* is configured.
- `[]` — Use this option when *scramblingID0/scramblingID1* is not configured. In this case, the object sets the DM-RS scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier.

Data Types: `double`

`NSCID` — DM-RS scrambling initialization
`0` (default) | `1`

DM-RS scrambling initialization, specified as `0` or `1`.

Data Types: `double`

`NumCDMGroupsWithoutData` — Number of DM-RS CDM groups without data
2 (default) | 1 | 3

Number of DM-RS CDM groups without data, specified as `1`, `2`, or `3`.

Each value indicates a different set of CDM group numbers, according to TS 38.214 Section 5.1.6.2.

- `1` — CDM group number 0
- `2` — CDM group numbers 0 and 1
- `3` — CDM group numbers 0, 1, and 2

Data Types: `double`

`CDMGroups` — CDM group numbers corresponding to each DM-RS port
`0` (default) | integer from 0 to 2 | row vector of integers

This property is read-only.

CDM group numbers corresponding to each DM-RS port, specified as one of these options.

- Integer from 0 to 2 — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from 0 to 2 — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to a CDM group number for that port.

Value of this property depends on the `DMRSConfigurationType` property according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`DeltaShifts` — Delta shifts corresponding to each CDM group

`0` (default) | integer from the set {0, 1, 2, 4} | row vector of integers

This property is read-only.

Delta shifts corresponding to each CDM group, specified as one of these options.

- Integer from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to the delta shift to be applied for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`FrequencyWeights` — Frequency weights

`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Frequency weights for the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`TimeWeights` — Time weights

`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Time weights for to the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 7.4.1.1.2-1 or 7.4.1.1.2-2 [1].

Data Types: `double`

`DMRSSubcarrierLocations` — Subcarrier locations in RB for each port
`[0; 2; 4; 6; 8; 10]` (default) | column vector of integers | matrix of integers

This property is read-only.

Subcarrier locations in an RB for each port, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the subcarrier locations for that port.

Data Types: `double`

`CDMLengths` — CDM arrangement for reference signals
`[1 1]` (default) | two-element row vector

This property is read-only.

CDM arrangement for reference signals, specified as the comma-separated pair consisting of `'CDMLengths'` and a two-element row vector of nonnegative integers [*FD TD*]. Array elements *FD* and *TD* specify the length of CDM despreading in the frequency domain (FD-CDM) and time domain (TD-CDM), respectively. A value of `1` for an element specifies no CDM.

Data Types: `double`

Examples

Create PDSCH DM-RS Object

Create a physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS) object.

Specify a single-symbol DMRS with configuration type as 2, number of DM-RS additional positions as 2, and antenna ports as 0, 1, and 3.

View the corresponding properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 1;
dmrs.DMRSAdditionalPosition = 2;
dmrs.DMRSPortSet = [0 1 3];
dmrs
```

```
dmrs =
  nrPDSCHDMRSConfig with properties:

      DMRSConfigurationType: 2
         DMRSReferencePoint: 'CRB0'
          DMRSTypeAPosition: 2
     DMRSAdditionalPosition: 2
                 DMRSLength: 1
            CustomSymbolSet: []
                DMRSPortSet: [0 1 3]
                   NIDNSCID: []
                      NSCID: 0
    NumCDMGroupsWithoutData: 2

   Read-only properties:
                  CDMGroups: [0 0 1]
                 DeltaShifts: [0 0 2]
           FrequencyWeights: [2x3 double]
                TimeWeights: [2x3 double]
    DMRSSubcarrierLocations: [4x3 double]
                 CDMLengths: [2 1]
```

Generate PDSCH DM-RS Symbols and Indices

Create a carrier configuration object specifying the slot number as 10.

```
carrier = nrCarrierConfig('NSlot',10);
```

Create a physical downlink shared channel (PDSCH) configuration object, `pdsch`, with physical resource blocks (PRBs) allocated from 0 to 30.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:30;
```

Create a PDSCH demodulation reference signal (DM-RS) object, `dmrs`, with specified properties.

```
dmrs = nrPDSCHDMRSConfig;
dmrs.DMRSConfigurationType = 2;
dmrs.DMRSLength = 2;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 5;
dmrs.NIDNSCID = 10;
dmrs.NSCID = 0;
```

Assign the PDSCH DM-RS configuration object to DMRS property of PDSCH configuration object.

```
pdsch.DMRS = dmrs;
```

Generate PDSCH DM-RS symbols and indices for the specified carrier, PDSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPDSCHDMRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 496x1 single column vector
```

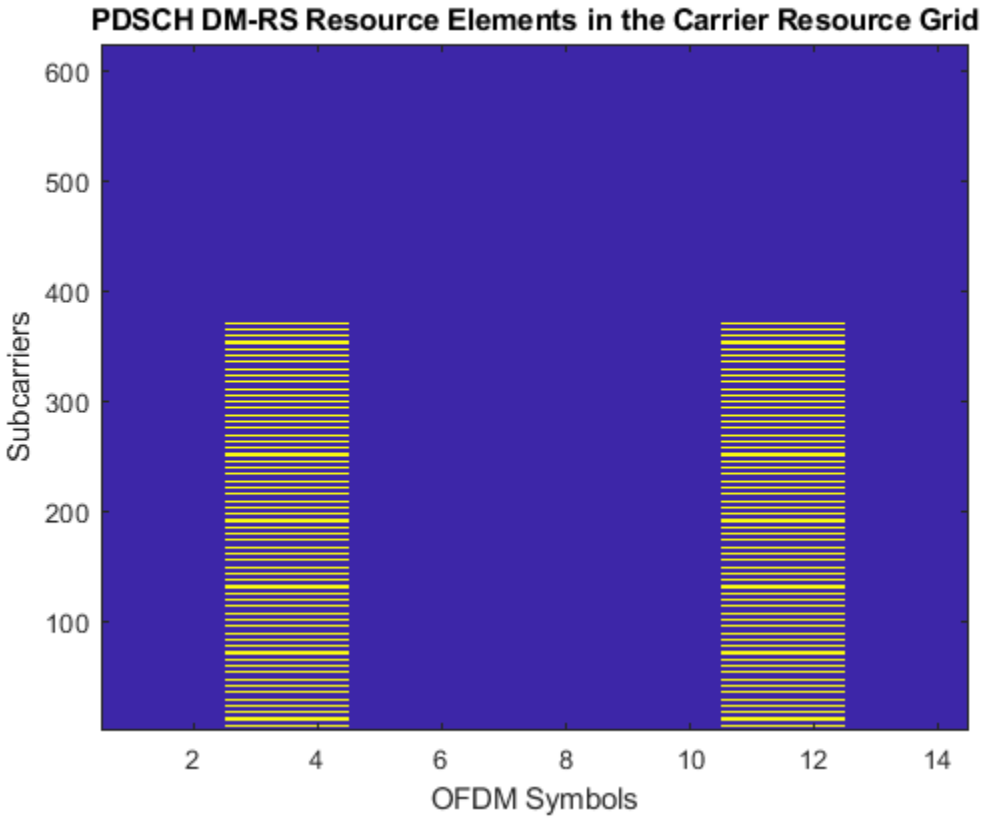
```
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
      ⋮
```

```
ind = nrPDSCHDMRSIndices(carrier,pdsch,'IndexBase','0based','IndexOrientation','carrier')
```

```
ind = 496x1 uint32 column vector

   1252
   1253
   1258
   1259
   1264
   1265
   1270
   1271
   1276
   1277
      ⋮
```

Display the generated DM-RS symbols on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pdsch.NumLayers]));
grid(ind+1) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PDSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPDSCH` | `nrPDSCHDMRS` | `nrPDSCHDMRSIndices`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHPTRSConfig`

Introduced in R2020a

nrPDSCHPTRSConfig

PDSCH PT-RS configuration parameters

Description

The `nrPDSCHPTRSConfig` object sets phase tracking reference signal (PT-RS) configuration parameters for a physical downlink shared channel (PDSCH), as defined in TS 38.211 Section 7.4.1.2 [1]. By default, the object defines the PT-RS with time density 1, frequency density 2, resource element offset '00' and PTRS port set [ ]. Use this object when setting the `PTRS` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects.

Creation

Syntax

```
ptrs = nrPDSCHPTRSConfig
ptrs = nrPDSCHPTRSConfig(Name,Value)
```

Description

`ptrs = nrPDSCHPTRSConfig` creates a PT-RS configuration object for a PDSCH with default properties.

`ptrs = nrPDSCHPTRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'TimeDensity',2,'FrequencyDensity',4` sets the time density to 2 and frequency density to 4. Unspecified properties take their default values.

Properties

`TimeDensity` — PT-RS time density
1 (default) | 2 | 4

PT-RS time density, specified as 1, 2 or 4. This property is the higher layer parameter *timeDensity*.

Data Types: `double`

`FrequencyDensity` — PT-RS frequency density
2 (default) | 4

PT-RS frequency density, specified as 2 or 4. This property is the higher layer parameter *frequencyDensity*.

Data Types: `double`

`REOffset` — Resource element offset
`'00'` (default) | `'01'` | `'10'` | `'11'`

Resource element offset with a specific subcarrier offset, specified as `'00'`, `'01'`,`'10'`, or `'11'`. This property is the higher layer parameter *resourceElementOffset*.

Data Types: `char` | `string`

`PTRSPortSet` — PT-RS antenna port set
`[]` (default) | nonnegative integer

PT-RS antenna port set, specified as a nonnegative integer. Specify `[]` to set this property to the lowest value in the `DMRSPortSet` property of `nrPDSCHDMRSConfig` object. This usage of `[]` value is applicable only when `nrPDSCHPTRSConfig` object is used as a property of `nrPDSCHConfig` or `nrWavegenPDSCHConfig`.

Data Types: `double`

Examples

Create PDSCH PT-RS Object

Create a PT-RS configuration object for a PDSCH. Set the time density to 2, frequency density to 4, and resource element offset to '10'.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
disp(ptrs)
```

```
  nrPDSCHPTRSConfig with properties:

         TimeDensity: 2
    FrequencyDensity: 4
           REOffset: '10'
         PTRSPortSet: []
```

Generate PDSCH PT-RS Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier.

```
carrier = nrCarrierConfig;
```

Create a default PDSCH configuration object, and then enable the PT-RS configuration.

```
pdsch = nrPDSCHConfig;
pdsch.EnablePTRS = 1;
```

Create a PDSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPDSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '10';
```

Assign the PDSCH PT-RS configuration object to PTRS property of PDSCH configuration object.

```
pdsch.PTRS = ptrs;
```

Generate PDSCH PT-RS symbols of data type single.

```
sym = nrPDSCHPTRS(carrier,pdsch,'OutputDataType','single')
```

```
sym = 78x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
      ⋮
```

Generate PDSCH PT-RS indices in subscript form and set the index orientation to bandwidth part.

```
ind = nrPDSCHPTRSIndices(carrier,pdsch,'IndexStyle','subscript','IndexOrientation','bwp')
```

```
ind = 78x3 uint32 matrix

    19     1     1
    67     1     1
   115     1     1
   163     1     1
   211     1     1
   259     1     1
   307     1     1
   355     1     1
   403     1     1
   451     1     1
      ⋮
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
nrPDSCHIndices | nrPDSCHPTRS | nrPDSCHPTRSIndices

Objects
nrCarrierConfig | nrPDSCHConfig | nrPDSCHDMRSConfig | nrPDSCHReservedConfig

Introduced in R2020a

nrPDSCHReservedConfig

PDSCH reserved PRB configuration parameters

Description

The `nrPDSCHReservedConfig` object sets physical downlink shared channel (PDSCH) reserved physical resource block (PRB) configuration parameters, as defined in TS 38.214 Section 5.1.4.1 [1].

The object configures the reserved PRB pattern for the PDSCH. By default, the object configures the empty reserved PRB pattern.

Creation

Syntax

```
reservedPRB = nrPDSCHReservedConfig
reservedPRB = nrPDSCHReservedConfig(Name,Value)
```

Description

`reservedPRB = nrPDSCHReservedConfig` creates a PDSCH reserved PRB configuration object with default properties.

`reservedPRB = nrPDSCHReservedConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'SymbolSet', (0:3),'Period',5` specifies the OFDM symbols associated with reserved PRBs as (0:3) and the period of the total number of slots in the pattern as 5. Unspecified properties take their default values.

Properties

PRBSet — Reserved PRB indices within BWP

`[]` (default) | vector of nonnegative integer

Reserved PRB indices within the bandwidth part (BWP), specified as a nonnegative integer vector. The PRB indices are 0-based. If this property is `[]` value, no reserved PRBs are defined.

Data Types: `double`

SymbolSet — OFDM symbols associated with reserved PRBs

`[]` (default) | vector of nonnegative integer

OFDM symbols associated with reserved PRBs spanning over one or more slots, specified a nonnegative integer vector. The symbol indices are 0-based. If this property is `[]` value, no reserved OFDM symbols are defined.

Data Types: `double`

`Period` — Period of number of slots in pattern

`[]` (default) | positive integer

Period of the number of slots in the pattern, specified as a positive integer. The SymbolSet property specifies provides the entire OFDM symbols pattern. This pattern repeats itself for every `Period` slots.

If this property is `[]` value, the OFDM symbols pattern does not cyclically repeat itself.

Data Types: `double`

Examples

Create PDSCH Object to Configure Two Reserved PRB Patterns

Create a PDSCH configuration object with two reserved PRB patterns.

```
pdsch = nrPDSCHConfig('ReservedPRB',{nrPDSCHReservedConfig,nrPDSCHReservedConfig});
```

For each reserved PRB pattern, specify the reserved PRB indices in the BWP, the OFDM symbols associated with those reserved PRBs, and the period for the number of slots in the pattern.

```
pdsch.ReservedPRB{1}.PRBSet = (0:15);
pdsch.ReservedPRB{1}.SymbolSet = (5:6);
pdsch.ReservedPRB{1}.Period = 5;
pdsch.ReservedPRB{2}.PRBSet = (0:23);
pdsch.ReservedPRB{2}.SymbolSet = [2:4 7:9];
pdsch.ReservedPRB{2}.Period = 3;
```

Display the two PRB patterns.

```
PRBPattern1 = pdsch.ReservedPRB{1}
```

```
PRBPattern1 =
  nrPDSCHReservedConfig with properties:

       PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
    SymbolSet: [5 6]
       Period: 5
```

```
PRBPattern2 = pdsch.ReservedPRB{2}
```

```
PRBPattern2 =
  nrPDSCHReservedConfig with properties:

       PRBSet: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
    SymbolSet: [2 3 4 7 8 9]
       Period: 3
```

References

[1] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPDSCH` | `nrPDSCHDecode`

Objects

`nrCarrierConfig` | `nrPDSCHConfig` | `nrPDSCHDMRSConfig` | `nrPDSCHPTRSConfig`

Introduced in R2020a

nrPRACHConfig

PRACH configuration parameters

Description

The `nrPRACHConfig` object sets physical random access channel (PRACH) configuration parameters for a PRACH preamble, as defined in TS 38.211 Section 5.3.2 and Section 6.3.3 [1].

Creation

Syntax

```
prach = nrPRACHConfig
prach = nrPRACHConfig(Name,Value)
```

Description

`prach = nrPRACHConfig` creates a PRACH configuration object with default properties.

`prach = nrPRACHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'ConfigurationIndex',106,'SubcarrierSpacing',30` specifies the time resource and subcarrier spacing for the PRACH preamble. Unspecified properties take their default values.

Properties

Configurable PRACH Properties

`FrequencyRange` — Frequency range
`'FR1'` (default) | `'FR2'`

Frequency range, specified as `'FR1'` or `'FR2'`.

Use this property together with the `DuplexMode` property to specify these PRACH configuration tables from TS 38.211.

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`DuplexMode` — Duplex mode for uplink transmission
`'FDD'` (default) | `'TDD'` | `'SUL'`

Duplex mode for uplink transmission, specified as one of these values:

- `'FDD'` — Use this value to specify frequency division duplex (FDD) mode for paired spectrum.
- `'TDD'` — Use this value to specify time division duplex (TDD) mode for unpaired spectrum.
- `'SUL'` — Use this value to specify supplementary uplink.

Use this property together with the `FrequencyRange` property to specify these PRACH configuration tables from TS 38.211:

- To specify Table 6.3.3.2-2, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'FDD'` or `'SUL'`.
- To specify Table 6.3.3.2-3, set `FrequencyRange` to `'FR1'` and `DuplexMode` to `'TDD'`.
- To specify Table 6.3.3.2-4, set `FrequencyRange` to `'FR2'` and `DuplexMode` to `'TDD'`.

Data Types: `char` | `string`

`ConfigurationIndex` — Time resource of PRACH preamble
`27` (default) | integer from 0 to 255

Time resource of PRACH preamble, specified as an integer from 0 to 255. This property specifies a configuration index from Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211. Properties `FrequencyRange` and `DuplexMode` determine the actual configuration table to consider.

This property is the higher layer parameter *prach-ConfigurationIndex*.

Data Types: `double`

`SubcarrierSpacing` — Subcarrier spacing for PRACH in kHz
`1.25` (default) | `5` | `15` | `30` | `60` | `120`

Subcarrier spacing for the PRACH in kHz, specified as `1.25`, `5`, `15`, or `30` for frequency range FR1 and `60` or `120` for frequency range FR2.

Set this property in relation to the preamble format property `Format`. To identify valid preamble format and subcarrier spacing combinations, see the `LongPreambleFormats` and `ShortPreambleFormats` fields of the `Tables` property. For more information, see Table 6.3.3.1-1 for long preambles and Table 6.3.3.1-2 for short preambles.

Data Types: `double`

`SequenceIndex` — Logical root sequence index
`0` (default) | integer from 0 to 837

Logical root sequence index, specified as an integer from 0 to 837.

This property corresponds to parameter *i* in TS 38.211 Tables 6.3.3.1-3 and 6.3.3.1-4 and is the higher layer parameter *prach-RootSequenceIndex*.

Data Types: `double`

`PreambleIndex` — Preamble index within cell
`0` (default) | integer from 0 to 63

Preamble index within the cell, specified as an integer from 0 to 63.

This property is the higher layer parameter *ra-PreambleIndex*.

Data Types: `double`

`RestrictedSet` — Type of restricted set
`'UnrestrictedSet'` (default) | `'RestrictedSetTypeA'` | `'RestrictedSetTypeB'`

Type of restricted set, specified as `'UnrestrictedSet'`, `'RestrictedSetTypeA'`, or `'RestrictedSetTypeB'`. Set this property in relation to the cyclic shift configuration index property `ZeroCorrelationZone`, as defined by N_{CS} in Tables 6.3.3.1-5 to 6.3.3.1-7 from TS 38.211.

Data Types: `char` | `string`

`ZeroCorrelationZone` — Cyclic shift configuration index
`0` (default) | integer from 0 to 15

Cyclic shift configuration index, specified as an integer from 0 to 15. Use this property together with the `RestrictedSet` and `SubcarrierSpacing` properties to retrieve the number of cyclic shifts for the sequence generation. For more information, see TS 38.211 Tables 6.3.3.1-5 to 6.3.3.1-7.

Data Types: `double`

`RBOffset` — Starting RB index of initial uplink BWP
`0` (default) | integer from 0 to 274

Starting resource block (RB) index of the initial uplink bandwidth part (BWP), relative to the carrier resource grid, specified as an integer from 0 to 274.

Data Types: `double`

`FrequencyStart` — Offset of lowest PRACH transmission occasion
`0` (default) | integer from 0 to 274

Offset of lowest PRACH transmission occasion, in frequency domain, relative to the physical resource block (PRB) 0, specified as an integer from 0 to 274.

This property corresponds to parameter n_{RA}^{start} in TS 38.211 Section 5.3.2 and is the higher layer parameter *msg1-FrequencyStart*.

Data Types: `double`

`FrequencyIndex` — Index of PRACH transmission occasion
`0` (default) | integer from 0 to 7

Index of PRACH transmission occasion, in frequency domain, specified as an integer from 0 to 7. The frequency index must be in the range from 0 to $M - 1$, where M is 1, 2, 4, or 8.

This property corresponds to parameter n_{RA} in TS 38.211 Sections 5.3.2 and 6.3.3.2 and is the higher layer parameter *msg1-FDM* defined in TS 38.331 Section 6.3.2.

Data Types: `double`

`TimeIndex` — Index of PRACH transmission occasion
`0` (default) | integer from 0 to 6

Index of the PRACH transmission occasion, in time domain, specified as an integer from 0 to 6. Set this property in relation to the length of the Zadoff-Chu preamble sequence, referred to as L_{RA} in TS 38.211 Section 6.3.3.

- When the `LRA` property is 839, `TimeIndex` must be `0`.

- When the `LRA` property is 139, `TimeIndex` must be in the range from 0 ≤ to `NumTimeOccasions` − 1.

This property corresponds to parameter n_t^{RA} in TS 38.211 Section 5.3.2.

Data Types: `double`

`ActivePRACHSlot` — Position of active PRACH slot within subframe or 60 kHz slot
`0` (default) | `1`

Position of active PRACH slot within a subframe (for FR1) or a 60 kHz slot (for FR2), specified as `0` or `1`.

- If the `SubcarrierSpacing` property is set to `1.25`, `5`, `15`, or `60`, then `ActivePRACHSlot` must be `0`.
- If `SubcarrierSpacing` is set to `30` or `120`, then `ActivePRACHSlot` must be set based on configuration tables TS 38.211 Table 6.3.3.2-2 to Table 6.3.3.2-4.

To specify the frequency range of the carrier as FR1 or FR2, use the `FrequencyRange` property.

This property corresponds to parameter n_{slot}^{RA} in TS 38.211 Section 5.3.2.

Data Types: `double`

`NPRACHSlot` — PRACH slot number
`0` (default) | nonnegative integer

PRACH slot number, specified as a nonnegative integer. You can set `NPRACHSlot` to a value larger than the number of slots per frame. For example, you can set this value using transmission loop counters in a MATLAB simulation. In this case, you might have to ensure that the property value is modulo the number of slots per frame in a calling code.

Data Types: `double`

Nonconfigurable PRACH Properties

The object automatically sets these properties based on configurable PRACH property values by using the configuration tables from TS 38.211 Section 6.3.3.

`Format` — Preamble format
`'0'` | `'1'` | `'2'` | `'3'` | `'A1'` | `'A2'` | `'A3'` | `'B1'` | `'B2'` | `'B3'` | `'B4'` | `'C0'` | `'C2'`

This property is read-only.

Preamble format, defined in TS 38.211 Tables 6.3.3.1-1 and 6.3.3.1-2, returned as `'0'`, `'1'`, `'2'`, `'3'`, `'A1'`, `'A2'`, `'A3'`, `'B1'`, `'B2'`, `'B3'`, `'B4'`, `'C0'`, or `'C2'`.

For short preamble format C0, each preamble has one active sequence period. Because the preamble spans two OFDM symbols, including the guard and the cyclic prefix, the grid related to format C0 has 7 OFDM symbols instead of 14.

Data Types: `char` | `string`

`LRA` — Length of Zadoff-Chu preamble sequence
`839` | `139`

This property is read-only.

Length of Zadoff-Chu preamble sequence, returned as `839` for long preambles or `139` for short preambles.

Data Types: `double`

`NumTimeOccasions` — Number of time-domain PRACH occasions within PRACH slot
integer from 0 to 7

This property is read-only.

Number of time-domain PRACH occasions within a PRACH slot, returned as an integer from 0 to 7. For long preambles, `NumTimeOccasions` is always 1. For more details, see TS 38.211 Section 5.3.2.

This property corresponds to parameter $N_t^{RA,slot}$ in TS 38.211 Tables 6.3.3.2-2 to 6.3.3.2-4.

Data Types: `double`

`PRACHDuration` — Number of OFDM symbols in PRACH slot grid
integer from 1 to 12

This property is read-only.

Number of OFDM symbols in the PRACH slot grid, corresponding to one transmission occasion, returned as an integer from 1 to 12.

This property corresponds to parameter $N_{dur}^{RA,slot}$ in TS 38.211 Tables 6.3.3.2-2 to 6.3.3.2-4. For format C0, because the grid has 7 OFDM symbols instead of 14, the object sets `PRACHDuration` to $N_{dur}^{RA,slot}$ / 2.

For long preamble formats 0 and 1, `PRACHDuration` is `1` and `2`, respectively. For long preamble formats 2 and 3, `PRACHDuration` is `4`. For more information on long preamble formats, see Table 6.3.3.1-1.

Data Types: `double`

`SymbolLocation` — First OFDM symbol location in current PRACH occasion
integer from 0 to 26

This property is read-only.

First OFDM symbol location in current PRACH occasion within a slot, returned as an integer from 0 to 26. If the `ActivePRACHSlot` property is set to `1`, this location can fall outside a PRACH slot.

This property corresponds to parameter l in TS 38.211 Section 5.3.2 with these exceptions.

- For format C0, because the grid has 7 OFDM symbols instead of 14, the object sets `SymbolLocation` to l / 2.
- For long preamble formats characterized by starting symbol location 7 in Table 6.3.3.2-3, the object sets the `SymbolLocation` to `0`.

Data Types: `double`

`SubframesPerPRACHSlot` — Total number of subframes per nominal PRACH slot

`0.125` | `0.25` | `0.5` | `1` | `3` | `4`

This property is read-only.

Total number of subframes per nominal PRACH slot, returned as `0.125`, `0.25`, `0.5`, `1`, `3`, or `4`.

Data Types: `double`

`PRACHSlotsPerPeriod` — Number of PRACH slots per overall period

`5` | `10` | `20` | `40` | `80` | `160` | `320` | `640`

This property is read-only.

Number of PRACH slots per overall period, returned as `5`, `10`, `20`, `40`, `80`, `160`, `320`, or `640`. The overall period spans an integer multiple of *x* frames, where Tables 6.3.3.2-2, 6.3.3.2-3, and 6.3.3.2-4 of [1] define *x*.

Data Types: `double`

PRACH Lookup Tables

`Tables` — PRACH configuration tables

constant structure

This property is read-only.

PRACH configuration tables, from TS 38.211 Section 6.3.3, returned as a constant structure containing these fields:

Fields	Values	Description
`LongPreambleFormats`	4-by-6 table	Table 6.3.3.1-1: Long PRACH preamble formats
`ShortPreambleFormats`	9-by-6 table	Table 6.3.3.1-2: Short PRACH preamble formats
`NCSFormat012`	16-by-4 table	Table 6.3.3.1-5: N_{CS} for long preamble formats with 1.25 kHz subcarrier spacing
`NCSFormat3`	16-by-4 table	Table 6.3.3.1-6: N_{CS} for long preamble formats with 5 kHz subcarrier spacing
`NCSFormatABC`	16-by-4 table	Table 6.3.3.1-7: N_{CS} for short preamble formats
`SupportedSCSCombinations`	16-by-5 table	Table 6.3.3.2-1: Supported combinations of subcarrier spacing for the PRACH and the physical uplink shared channel (PUSCH)
`ConfigurationsFR1PairedSUL`	256-by-9 table	Table 6.3.3.2-2: PRACH configurations for FR1 and paired spectrum or FR1 and supplementary uplink
`ConfigurationsFR1Unpaired`	256-by-9 table	Table 6.3.3.2-3: PRACH configurations for FR1 and unpaired spectrum
`ConfigurationsFR2`	256-by-9 table	Table 6.3.3.2-4: PRACH configurations for FR2 and unpaired spectrum

Invalid PRACH Configurations

Based on the configuration tables in TS 38.211 Section 6.3.3, these property setting combinations and scenarios lead to invalid PRACH configurations.

- Setting `'FrequencyRange'` to `'FR2'` and `'DuplexMode'` to `'FDD'` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'DuplexMode'` to `'SUL'` is invalid.
- Setting `'FrequencyRange'` to `'FR1'` and `'SubcarrierSpacing'` to `60` is invalid.
- Setting `'FrequencyRange'` to `'FR1'` and `'SubcarrierSpacing'` to `120` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `1.25` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `5` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `15` is invalid.
- Setting `'FrequencyRange'` to `'FR2'` and `'SubcarrierSpacing'` to `30` is invalid.
- Any combination of properties `Format` and `SubcarrierSpacing` not listed in Table 6.3.3.1-1 for long preambles or Table 6.3.3.1-2 for short preambles is invalid. You can identify valid combinations in the `LongPreambleFormats` and `ShortPreambleFormats` fields of the `Tables` property.
- Any combination of properties `ZeroCorrelationZone` and `RestrictedSet` not listed in Tables 6.3.3.1-5, 6.3.3.1-6, and 6.3.3.1-7 is invalid. You can identify valid combinations in the `NCSFormat012`, `NCSFormat3`, and `NCSFormatABC` fields, respectively, of the `Tables` property.
- Any combination of properties `ActivePRACHSlot`, `FrequencyRange`, `DuplexMode`, `ConfigurationIndex`, and `SubcarrierSpacing` not covered in Section 5.3.2 is invalid.
- Any configuration where `TimeIndex` ≥ `NumTimeOccasions` is invalid.

Examples

Configure PRACH Preamble Format

Create a PRACH configuration object with default properties. The default configuration object defines a PRACH configuration with long preamble format 0, based on TS 38.211 Table 6.3.3.2-2.

```
prach = nrPRACHConfig;
```

To consider a different PRACH configuration table as a basis, for example Table 6.3.3.2-3 for FR1 and unpaired spectrum, update the duplex mode property.

```
prach.DuplexMode = 'TDD';
```

To change the PRACH preamble format, you must update the `ConfigurationIndex` property of the object based on Table 6.3.3.2-3. To lookup a suitable value, access this table through the `ConfigurationsFR1Unpaired` field of the `Tables` property.

```
prach.Tables.ConfigurationsFR1Unpaired(:,:)
```

```
ans=256×9 table
    ConfigurationIndex    PreambleFormat    x       y      SubframeNumber    StartingSymbol    PR
    __________________    ______________    __    _____    ______________    ______________    __

            0                {'0'}          16    {[1]}     {[       9]}           0
            1                {'0'}           8    {[1]}     {[       9]}           0
```

```
    2                {'0'}          4    {[1]}     {[       9]}           0
    3                {'0'}          2    {[0]}     {[       9]}           0
    4                {'0'}          2    {[1]}     {[       9]}           0
    5                {'0'}          2    {[0]}     {[       4]}           0
    6                {'0'}          2    {[1]}     {[       4]}           0
    7                {'0'}          1    {[0]}     {[       9]}           0
    8                {'0'}          1    {[0]}     {[       8]}           0
    9                {'0'}          1    {[0]}     {[       7]}           0
   10                {'0'}          1    {[0]}     {[       6]}           0
   11                {'0'}          1    {[0]}     {[       5]}           0
   12                {'0'}          1    {[0]}     {[       4]}           0
   13                {'0'}          1    {[0]}     {[       3]}           0
   14                {'0'}          1    {[0]}     {[       2]}           0
   15                {'0'}          1    {[0]}     {1x2 double}           0
```

To change the preamble from format 0 to format A1, set the `ConfigurationIndex` property to any value from 67 to 86.

```
prach.ConfigurationIndex = 86;
```

Verify that the object updates the preamble format correctly.

```
isequal(prach.Format,'A1')
```

```
ans = logical
   1
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPRACH` | `nrPRACHGrid` | `nrPRACHIndices`

Topics

“5G NR PRACH Configuration”

Introduced in R2020a

nrPUSCHConfig

PUSCH configuration parameters

Description

The `nrPUSCHConfig` object sets physical uplink shared channel (PUSCH) configuration parameters, as defined in TS 38.211 Sections 6.3.1, 6.4.1.1, and 6.4.1.2 [1].This object bundles all the properties involved in the PUSCH processing chain, including scrambling, symbol modulation, layer mapping, transform precoding, MIMO precoding, and resource element mapping. The object also contains properties to determine the number of resources for the uplink control information (UCI) multiplexing and associated physical reference signals, such as demodulation reference signal (DM-RS) and phase tracking reference signal (PT-RS).

By default, the object configures the physical uplink shared channel with cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM) occupying a 10 MHz bandwidth, at 15 kHz subcarrier spacing (52 resource blocks) and spanning over 14 OFDM symbols in a slot.

Creation

Syntax

```
pusch = nrPUSCHConfig
pusch = nrPUSCHConfig(Name,Value)
```

Description

`pusch = nrPUSCHConfig` creates a PUSCH configuration object with default properties.

`pusch = nrPUSCHConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NSizeBWP',200,'NumLayers',4` specifies 200 physical resource blocks (PRB) in the bandwidth part (BWP) and 4 transmission layers. Unspecified properties take their default values.

Properties

NSizeBWP — Number of PRBs in BWP

`[]` (default) | integer from 1 to 275

Number of PRBs in bandwidth part (BWP), specified as an integer from 1 to 275. Use `[]` to set this property to the `NSizeGrid` property of the `nrCarrierConfig` object.

Data Types: `double`

NStartBWP — Starting PRB index of BWP relative to CRB 0

`[]` (default) | integer from 0 to 2473

Starting PRB index of BWP relative to common resource block 0 (CRB 0), specified as an integer from 0 to 2473. Use `[]` to set this property to the `NStartGrid` property of the `nrCarrierConfig` object.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'pi/2-BPSK'` | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar

Modulation scheme, specified as `'QPSK'`, `'pi/2-BPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, or a character array.

Modulation Scheme	Number of Bits Per Symbol
`'pi/2-BPSK'`	1
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Data Types: `char` | `string`

`NumLayers` — Number of transmission layers
`1` (default) | `2` | `3` | `4`

Number of transmission layers, specified as `1`, `2`, `3`, or `4`.

Data Types: `double`

`MappingType` — Mapping type
`'A'` (default) | `'B'`

Mapping type of the physical shared channel, specified as `'A'` or `'B'`.

Data Types: `char` | `string`

`SymbolAllocation` — OFDM symbol allocation
`[0 14]` (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

When this property is set to `[]` or second element in this two-element vector of nonnegative integers is `0`, the symbol allocation is empty.

Data Types: `double`

`PRBSet` — PRB allocation
`[0:51]` (default) | vector of nonnegative integers from 0 to 274

Physical resource block (PRB) allocation of PUSCH within the BWP, specified as a vector of nonnegative integers from 0 to 274.

Data Types: `double`

`TransformPrecoding` — Transform precoding flag
`0` (default) | `1`

Transform precoding flag, specified as one of these values.

- `0` — The transform precoding is disabled and waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).
- `1` — The transform precoding is enabled and waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

Data Types: `double` | `logical`

`TransmissionScheme` — PUSCH transmission scheme

`'nonCodebook'` (default) | `'codebook'`

PUSCH transmission scheme, specified as `'nonCodebook'` or `'codebook'`.

Data Types: `char` | `string`

`NumAntennaPorts` — Number of antenna ports

`1` (default) | `2` | `4`

Number of antenna ports, specified as `1`, `2`, or `4`. This value must be greater than or equal to the `NumLayers` property.

Dependencies

This property is applicable only when `TransmissionScheme` is set to `'codebook'`.

Data Types: `double`

`TPMI` — Transmitted precoding matrix indicator

`0` (default) | integer from 0 to 27

Transmitted precoding matrix indicator, specified as an integer from 0 to 27.

Dependencies

This property is applicable only when `TransmissionScheme` is set to `'codebook'`.

Data Types: `double`

`FrequencyHopping` — Frequency hopping

`'neither'` (default) | `'intraSlot'` | `'interSlot'`

Frequency hopping for the physical uplink shared channel, specified as `'neither'`, `'intraSlot'`, or `'interSlot'`.

Data Types: `char` | `string`

`SecondHopStartPRB` — Starting PRB index of second hop

`1` (default) | integer from 0 to 274

Starting PRB index of the second hop relative to the BWP, specified as an integer from 0 to 274.

Dependencies

This property is applicable only when `FrequencyHopping` is set to `'intraSlot'`, or `'interSlot'`.

Data Types: `double`

`BetaOffsetACK` — Beta offset factor of HARQ-ACK

`20` (default) | positive integer

Beta offset factor of the hybrid automatic repeat request acknowledgment (HARQ-ACK), specified as a positive integer. This property is used to determine the number of resources for multiplexing HARQ-ACK. The nominal value is one of the entry from the Table 9.3-1 of TS 38.213.

Data Types: `double`

`BetaOffsetCSI1` — Beta offset factor of CSI part 1
`6.25` (default) | positive integer

Beta offset factor of the channel state information (CSI) part 1, specified as a positive integer. This property is used to determine the number of resources for multiplexing CSI part 1. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

`BetaOffsetCSI2` — Beta offset factor of CSI part 2
`6.25` (default) | positive integer

Beta offset factor of the CSI part 2, specified as a positive integer. This property is used to determine the number of resources for multiplexing CSI part 2. The nominal value is one of the entry from the Table 9.3-2 of TS 38.213.

Data Types: `double`

`UCIScaling` — Scaling factor
`1` (default) | scalar in the range (0, 1)

Scaling factor to limit the number of the resource elements allocated for the uplink channel information (UCI) on the PUSCH, specified as a scalar in the range (0, 1). The nominal value is 0.5, 0.65, 0.8, or 1.

Data Types: `double`

`NID` — Scrambling identity
`[]` (default) | integer from 0 to 1023

Scrambling identity, specified as an integer from 0 to 1023. Use `[]` to set this property to the `NCellID` property of the `nrCarrierConfig` object.

Data Types: `double`

`RNTI` — Radio network temporary identifier
`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`DMRS` — PUSCH DM-RS configuration parameters
`nrPUSCHDMRSConfig` object with default properties (default) | `nrPUSCHDMRSConfig` object

PUSCH demodulation reference signal (DM-RS) configuration parameters, specified as a `nrPUSCHDMRSConfig` configuration object. This property relates to the demodulation reference signal configuration and contains all properties of the specified `nrPUSCHDMRSConfig` object.

`EnablePTRS` — Enable PT-RS
`0` (default) | `1`

Enable PT-RS, specified as one of these values.

- `0` — Disable PT-RS configuration.
- `1` — Enable PT-RS configuration.

Data Types: `double` | `logical`

PTRS — PUSCH PT-RS configuration parameters
`nrPUSCHPTRSConfig` object with default properties (default) | `nrPUSCHPTRSConfig` object

PUSCH phase tracking reference signal (PT-RS) configuration, specified as a `nrPUSCHPTRSConfig` configuration object. This property relates to the phase tracking reference signal configuration and contains all properties of the specified `nrPUSCHPTRSConfig` object.

Examples

Create PUSCH Configuration Object

Create a physical uplink shared channel configuration object with 'codebook' transmission scheme.

Specify the size of the bandwidth part as 52, scrambling identity as 750, frequency hopping as 'interslot', and number of antenna ports as 2. Enable transform precoding and PT-RS configuration.

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 52;
pusch.NID = 750;
pusch.TransmissionScheme = 'codebook';
pusch.FrequencyHopping = 'interslot';
pusch.NumAntennaPorts = 2;
pusch.TransformPrecoding = 1;
pusch.EnablePTRS = 1;
disp(pusch)
```

```
  nrPUSCHConfig with properties:

              NSizeBWP: 52
             NStartBWP: []
            Modulation: 'QPSK'
             NumLayers: 1
           MappingType: 'A'
      SymbolAllocation: [0 14]
                PRBSet: [1x52 double]
    TransformPrecoding: 1
    TransmissionScheme: 'codebook'
       NumAntennaPorts: 2
                  TPMI: 0
      FrequencyHopping: 'interSlot'
     SecondHopStartPRB: 1
         BetaOffsetACK: 20
        BetaOffsetCSI1: 6.2500
        BetaOffsetCSI2: 6.2500
            UCIScaling: 1
                   NID: 750
                  RNTI: 1
                  DMRS: [1x1 nrPUSCHDMRSConfig]
```

```
          EnablePTRS: 1
                PTRS: [1x1 nrPUSCHPTRSConfig]
```

Generate PUSCH Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission. Set the number of antenna ports to 4, modulation scheme to pi/2-BPSK, transmitted precoding matrix indicator to 10, and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.Modulation = 'pi/2-BPSK';
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 10;
```

Generate PUSCH indices in subscript form.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 32448x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
    5    1    1
    6    1    1
    7    1    1
    8    1    1
    9    1    1
   10    1    1
      ⋮
```

```
info = struct with fields:
                G: 8112
               Gd: 8112
        NREPerPRB: 156
    DMRSSymbolSet: 2
    PTRSSymbolSet: [1x0 double]
```

Generate PUSCH Symbols and Indices

Create a carrier configuration object with default properties. This object corresponds to 30 kHz of subcarrier spacing and 20 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 51;
```

Create a PUSCH configuration object with specified properties. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NStartBWP = 10;
pusch.NSizeBWP = 41;
pusch.Modulation = '16QAM';
pusch.NID = []; % Set NID equal to the NCellID property of carrier.
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
```

Generate PUSCH indices, setting the index orientation with respect to the carrier grid.

```
[ind,info] = nrPUSCHIndices(carrier,pusch,'IndexOrientation','carrier')
```

```
ind = 864x1 uint32 column vector

   121
   122
   123
   124
   125
   126
   127
   128
   129
   130
      ⋮


info = struct with fields:
                 G: 3456
                Gd: 864
        NREPerPRB: 144
    DMRSSymbolSet: [2 7]
    PTRSSymbolSet: [1x0 double]
```

Generate PUSCH symbols of data type single.

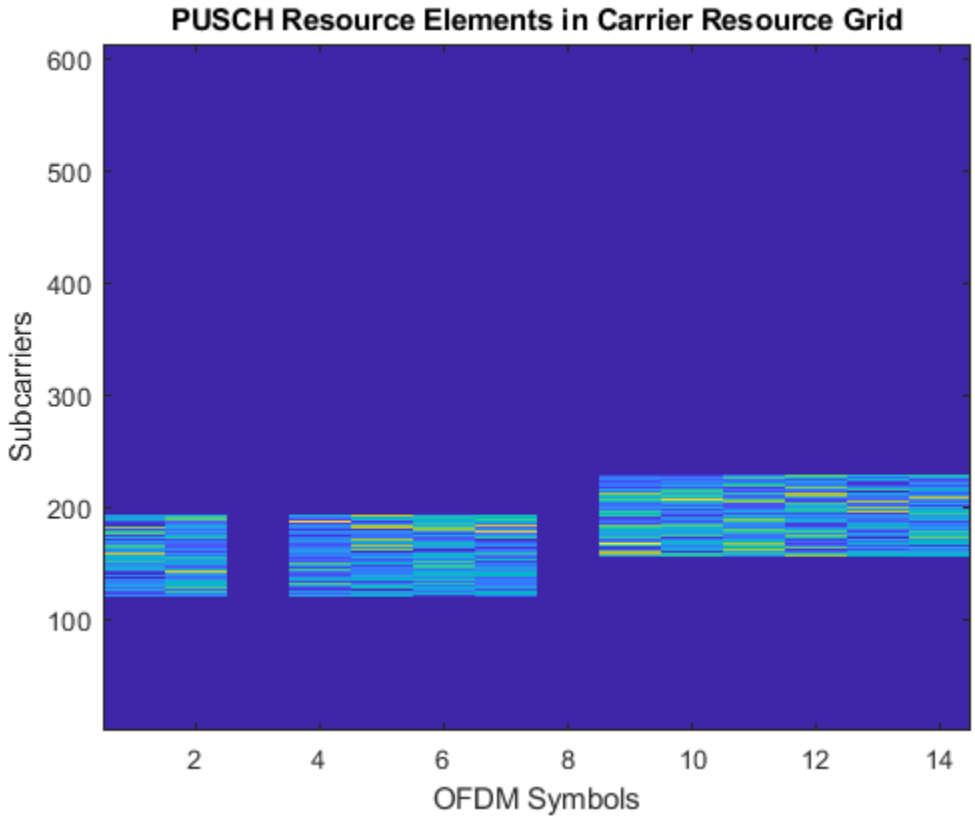
```
numDataBits = info.G;
cws = randi([0 1],numDataBits,1);
sym = nrPUSCH(carrier,pusch,cws,'OutputDataType','single')
```

```
sym = 864x1 single column vector

  -0.7454 + 0.2981i
   0.3406 - 0.2312i
  -0.1153 + 0.2756i
   1.1921 - 0.3658i
  -0.3968 - 0.0277i
  -0.8788 - 0.6493i
```

```
-0.8737 + 0.8318i
-0.5764 + 0.0269i
-1.6638 + 0.0482i
-1.0270 - 0.1347i
    ⋮
```

Plot the generated symbols and indices on the carrier resource grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers]));
grid(ind) = sym;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH Resource Elements in Carrier Resource Grid');
```



References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

nrPUSCH | nrPUSCHDMRS | nrPUSCHDMRSIndices | nrPUSCHDecode | nrPUSCHIndices | nrPUSCHPTRS | nrPUSCHPTRSIndices | nrULSCHDemultiplex | nrULSCHInfo | nrULSCHMultiplex

Objects

nrCarrierConfig | nrPUSCHDMRSConfig | nrPUSCHPTRSConfig

Introduced in R2020a

nrPUSCHDMRSConfig

PUSCH DM-RS configuration parameters

Description

The `nrPUSCHDMRSConfig` object sets demodulation reference signal (DM-RS) configuration parameters for a physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.1 [1].

The object defines the properties of PUSCH DM-RS symbols and indices generation and the resource elements pattern not used for data in DM-RS symbol locations. The read-only properties of this object provide the DM-RS subcarrier locations within a resource block (RB), code division multiplexing (CDM) groups, and time and frequency weights for DM-RS symbols. By default, the object specifies a single symbol DM-RS at symbol index 2 (0-based) with configuration type 1 and antenna port 0. Use this object when setting the `DMRS` property of the `nrPUSCHConfig` object.

Creation

Syntax

```
dmrs = nrPUSCHDMRSConfig
dmrs = nrPUSCHDMRSConfig(Name,Value)
```

Description

`dmrs = nrPUSCHDMRSConfig` creates a DM-RS configuration object for a PUSCH with default properties.

`dmrs = nrPUSCHDMRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'DMRSConfigurationType',1,'DMRSLength',2` specifies a double-symbol DM-RS with configuration type as 1. Unspecified properties take their default values.

Properties

`DMRSConfigurationType` — DM-RS configuration type
1 (default) | 2

DM-RS configuration type, specified as `1` or `2`. This property is the higher-layer parameter *dmrs-Type*.

This property value must be `1` when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` with `TransformPrecoding` property set to `1`.

Data Types: `double`

`DMRSTypeAPosition` — Position of first DM-RS OFDM symbol
2 (default) | 3

Position of first DM-RS OFDM symbol, provided by higher layer parameter *dmrs-TypeA-Position*, specified as `2` or `3`.

This property is applicable when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object with `MappingType` property value set to `'A'`.

Data Types: `double`

`DMRSAdditionalPosition` — Maximum number of DM-RS additional positions
`0` (default) | `1` | `2` | `3`

Maximum number of DM-RS additional positions, specified as `0`, `1`, `2`, or `3`. This property is the higher layer parameter *dmrs-AdditionalPosition*.

This property value must be `0` or `1` when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object with `FrequencyHopping` property set to `'intraSlot'`.

Data Types: `double`

`DMRSLength` — Number of consecutive front-loaded DM-RS OFDM symbols
`1` (default) | `2`

Number of consecutive front-loaded DM-RS OFDM symbols, specified as `1` (single-symbol DM-RS) or `2` (double-symbol DM-RS).

This property value must be `1` when nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object with `FrequencyHopping` property set to `'intraSlot'`.

Data Types: `double`

`CustomSymbolSet` — DM-RS OFDM symbol locations
`[]` (default) | integer from 0 to 13 | vector of nonnegative integers

DM-RS OFDM symbol locations that are 0-based, specified as one of these options.

- Integer from 0 to 13 — For one DM-RS symbol
- Vector of nonnegative integers from 0 to 13 — For multiple DM-RS symbols

Each input symbol location is assumed to be a single-symbol DM-RS within the physical shared channel symbol allocation.

The default value, `[]`, corresponds to the DM-RS symbols locations as per TS 38.211 Table 6.4.1.1.3-3, 6.4.1.1.3-4, or 6.4.1.1.3-6 [1]. Setting this property overrides the corresponding DM-RS symbol locations in these standard lookup tables.

Data Types: `double`

`DMRSPortSet` — DM-RS antenna ports
`[]` (default) | integer scalar | vector of nonnegative integers

DM-RS antenna ports, specified as one of these options.

- Integer from 0 to 11 — For a single antenna port
- Vector of nonnegative integers from 0 to 11 — For multiple antenna ports

Nominal antenna ports supported depend on `DMRSLength` and `DMRSConfigurationType` property values, as shown in this table.

DMRSLength Value	DMRSConfigurationType Value	Nominal Range of Antenna Ports Supported
1	1	[0, 3]
	2	[0, 5]
2	1	[0, 7]
	2	[0, 11]

The default value of `[]` implies that DM-RS antenna port is equal to `0`.

When nrPUSCHDMRSConfig is a property of `nrPUSCHConfig` object, `[]` implies that `DMRSPortSet` is in the range from 0 to `NumLayers`–1.

Data Types: `double`

`NIDNSCID` — DM-RS scrambling identity for CP-OFDM

`[]` (default) | integer from 0 to 65,535

DM-RS scrambling identity for CP-OFDM, specified as one of these options.

- Integer from 0 to 65,535 — If `NIDNSCID` is higher-layer parameter *scramblingID0/scramblingID1*
- `[]` — If `NIDNSCID` is not a higher-layer parameter, then the value is equal to the `NCellID` property of the `nrCarrierConfig` object. Use `[]` to set this property to the `NCellID` property value.

Dependencies

This property applies when the `TransformPrecoding` property of the `nrPUSCHConfig` object is set to `0`.

Data Types: `double`

`NRSID` — DM-RS scrambling identity for DFT-s-OFDM

`[]` (default) | integer from 0 to 1007

DM-RS scrambling identity for DFT-s-OFDM, specified as one of these options.

- Integer from 0 to 1007 — If `NRSID` is the higher-layer parameter *nPUSCH-Identity*.
- `[]` — Use this option to set the value of this property to the `NCellID` property value of the `nrCarrierConfig` object when the higher-layer parameter *nPUSCH-Identity* is undefined.

Dependencies

This property applies when the `TransformPrecoding` property of the `nrPUSCHConfig` object is set to `1`.

Data Types: `double`

`NSCID` — DM-RS scrambling initialization for CP-OFDM

`0` (default) | `1`

DM-RS scrambling initialization for CP-OFDM, specified as `0` or `1`.

Dependencies

This property applies when the `TransformPrecoding` property of the `nrPUSCHConfig` object is set to `0`.

Data Types: `double`

`GroupHopping` — Group hopping configuration

`0` (default) | `1`

Group hopping configuration, specified as one of these options.

- `0` — Group hopping is disabled.
- `1` — Group hopping is enabled.

Dependencies

This property applies when the `TransformPrecoding` property of the `nrPUSCHConfig` object is set to `1` and `SequenceHopping` is set to `0`.

Data Types: `logical` | `double`

`SequenceHopping` — Sequence hopping configuration

`0` (default) | `1`

Sequence hopping configuration, specified as one of these options.

- `0` — `SequenceHopping` is disabled.
- `1` — `SequenceHopping` is enabled.

Dependencies

This property applies when the `TransformPrecoding` property of the `nrPUSCHConfig` object is set to `1` and `GroupHopping` is set to `0`.

Data Types: `logical` | `double`

`NumCDMGroupsWithoutData` — Number of CDM groups without data

`2` (default) | `1` | `3`

Number of DM-RS CDM groups without data, specified as `1`, `2`, or `3`.

Each value indicates a different set of CDM group numbers, according to TS 38.214 Section 6.2.2 [2].

- `1` — CDM group number 0
- `2` — CDM group numbers 0 and 1
- `3` — CDM group numbers 0, 1, and 2

When `TransformPrecoding` property of the `nrPUSCHConfig` object is set to `1`, this property value must be `2`.

Data Types: `double`

`CDMGroups` — CDM group numbers corresponding to each port

`0` (default) | integer from 0 to 2 | row vector of integers

This property is read-only.

CDM group numbers corresponding to each DM-RS port, specified as one of these options.

- Integer from 0 to 2 — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from 0 to 2 — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to a CDM group number for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`DeltaShifts` — Delta shifts corresponding to each CDM group
`0` (default) | integer from the set {0, 1, 2, 4} | row vector of integers

This property is read-only.

Delta shifts corresponding to each CDM group, specified as one of these options.

- Integer from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Row vector of integers from the set {0, 1, 2, 4} — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each element corresponds to the delta shift to be applied for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`FrequencyWeights` — Frequency weights
`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Frequency weights for the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`TimeWeights` — Time weights
`[1; 1]` (default) | column vector of integers | matrix of integers

This property is read-only.

Time weights for to the DM-RS symbols, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.

- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the weights for that port.

Value of this property depends on the `DMRSConfigurationType` according to TS 38.211 Table 6.4.1.1.3-1 or 6.4.1.1.3-2 [1].

Data Types: `double`

`DMRSSubcarrierLocations` — Subcarrier locations in RB for each port
`[0; 2; 4; 6; 8; 10]` (default) | column vector of integers | matrix of integers

This property is read-only.

Subcarrier locations in an RB for each port, specified as one of these options.

- Column vector of integers — When `DMRSPortSet` property is a scalar, specifying single DM-RS antenna port.
- Matrix of integers — When `DMRSPortSet` property is a vector, specifying multiple DM-RS antenna ports. Each column corresponds to the subcarrier locations for that port.

Data Types: `double`

`CDMLengths` — CDM arrangement for reference signals
`[1 1]` (default) | two-element row vector

This property is read-only.

CDM arrangement for reference signals, specified as the comma-separated pair consisting of `'CDMLengths'` and a two-element row vector of nonnegative integers [*FD TD*]. Array elements *FD* and *TD* specify the length of CDM despreading in the frequency domain (FD-CDM) and time domain (TD-CDM), respectively. A value of `1` for an element specifies no CDM.

Data Types: `double`

Examples

Create PUSCH DM-RS Object

Create a physical uplink shared channel (PUSCH) specific demodulation reference signal (DM-RS) object, `dmrs`.

Specify a single-symbol DMRS with number of DM-RS additional positions as 3, sequence hopping as 1, and having antenna ports as 0 and 4.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSLength = 1;
dmrs.DMRSAdditionalPosition = 3;
dmrs.SequenceHopping = 1;
dmrs.DMRSPortSet = [0 4];
```

View the corresponding `dmrs` properties.

```
disp(dmrs)
```

```
  nrPUSCHDMRSConfig with properties:
```

```
      DMRSConfigurationType: 1
          DMRSTypeAPosition: 2
     DMRSAdditionalPosition: 3
                 DMRSLength: 1
            CustomSymbolSet: []
                DMRSPortSet: [0 4]
                   NIDNSCID: []
                      NSCID: 0
                GroupHopping: 0
            SequenceHopping: 1
                       NRSID: []
    NumCDMGroupsWithoutData: 2

   Read-only properties:
                  CDMGroups: [0 0]
                DeltaShifts: [0 0]
           FrequencyWeights: [2x2 double]
                TimeWeights: [2x2 double]
    DMRSSubcarrierLocations: [6x2 double]
                CDMLengths: [1 1]
```

Generate PUSCH DM-RS Symbols for CP-OFDM

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 1;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 2;
pusch.DMRS.NIDNSCID = 10;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS symbols associated with PUSCH of `single` data type.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 624x4 single matrix

  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
  -0.3536 + 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 + 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.3536 - 0.3536i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
      ⋮
```

Generate PUSCH DM-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

Configure PUSCH demodulation reference signal (DM-RS) object with specified parameters.

```
pusch.DMRS.DMRSAdditionalPosition = 2;
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSPortSet = 3;
pusch.DMRS.NIDNSCID = 15;
pusch.DMRS.NSCID = 1;
```

Generate DM-RS indices associated to PUSCH of subscript indexing form.

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 3744x3 uint32 matrix

    2    3    1
    4    3    1
    6    3    1
    8    3    1
   10    3    1
   12    3    1
   14    3    1
   16    3    1
   18    3    1
   20    3    1
      ⋮
```

Generate PUSCH DM-RS Symbols and Indices

Create a carrier configuration with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a physical uplink shared channel (PUSCH) configuration object with specified properties. When transform precoding is set to 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.NSizeBWP = 9;
pusch.NStartBWP = 1;
pusch.PRBSet = 0:3;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 5;
```

Create a PUSCH demodulation reference signal (DM-RS) object with specified properties.

```
dmrs = nrPUSCHDMRSConfig;
dmrs.DMRSAdditionalPosition = 1;
dmrs.DMRSTypeAPosition = 2;
dmrs.DMRSPortSet = 3;
dmrs.GroupHopping = 1;
dmrs.SequenceHopping = 0;
dmrs.NRSID = 10;
```

Assign the PUSCH DM-RS configuration object to DMRS property of PUSCH configuration object.

```
pusch.DMRS = dmrs;
```

Generate PUSCH DM-RS symbols and indices for the specified carrier, PUSCH configuration, and output formatting `name-value` pair argument.

```
sym = nrPUSCHDMRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 96x1 single column vector

  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
       ⋮
```

```
ind = nrPUSCHDMRSIndices(carrier,pusch,'IndexBase','0based','IndexOrientation','bwp')
```

```
ind = 96x1 uint32 column vector

   217
```

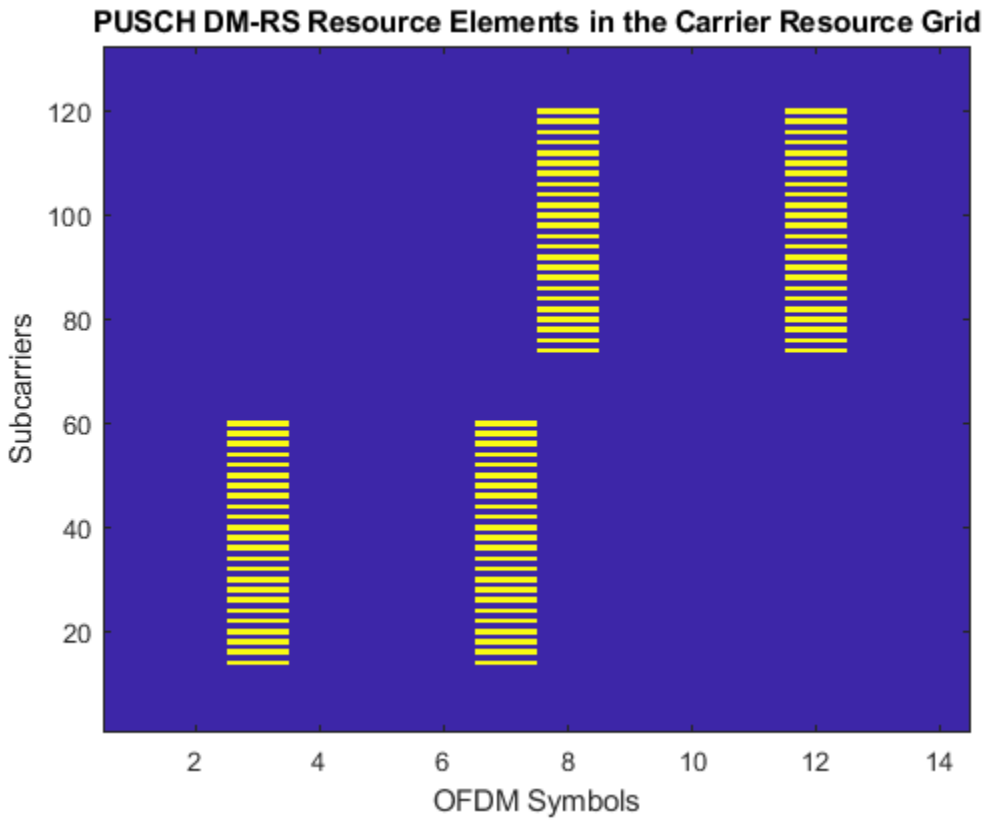
```
219
221
223
225
227
229
231
233
235
```

Create a bandwidth part (BWP) grid, and then map the DM-RS symbols on the grid.

```
bwp = complex(zeros([pusch.NSizeBWP*12 carrier.SymbolsPerSlot pusch.NumLayers]));
bwp(ind+1) = sym; % Map the DM-RS symbols
```

Map the BWP to the carrier resource grid, and then display the carrier grid.

```
grid = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot pusch.NumLayers])); % Create ca
offset = pusch.NStartBWP-carrier.NStartGrid; % BWP start location in the carrier grid
grid(offset*12+1:(offset+pusch.NSizeBWP)*12,:,:) = bwp;
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('PUSCH DM-RS Resource Elements in the Carrier Resource Grid');
```



References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPUSCH` | `nrPUSCHDMRS` | `nrPUSCHDMRSIndices`

Objects

`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHPTRSConfig`

Introduced in R2020a

nrPUSCHPTRSConfig

PUSCH PT-RS configuration parameters

Description

The `nrPUSCHPTRSConfig` object sets phase tracking reference signal (PT-RS) configuration parameters for the physical uplink shared channel (PUSCH), as defined in TS 38.211 Section 6.4.1.2 [1]. This object bundles all the properties involved in PUSCH PT-RS symbols and indices generation. By default, the object defines the PT-RS with a frequency density of 2 and time density of 1. Use this object when setting the `PTRS` property of the `nrPUSCHConfig` object.

Creation

Syntax

```
ptrs = nrPUSCHPTRSConfig
ptrs = nrPUSCHPTRSConfig(Name,Value)
```

Description

`ptrs = nrPUSCHPTRSConfig` creates a PUSCH-specific PT-RS configuration object with default properties.

`ptrs = nrPUSCHPTRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'TimeDensity',2,'FrequencyDensity',4` sets the time density to 2 and frequency density to 4. Unspecified properties take their default values.

Properties

`TimeDensity` — PT-RS time density

1 (default) | 2 | 4

PT-RS time density, specified as 1, 2, or 4. This property is the higher layer parameter *timeDensity*.

Data Types: `double`

`FrequencyDensity` — PT-RS frequency density

2 (default) | 4

PT-RS frequency density, specified as 2 or 4. This property is the higher layer parameter *frequencyDensity*.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `0`.

Data Types: `double`

`NumPTRSSamples` — Number of PT-RS samples

2 (default) | 4

Number of PT-RS samples per PT-RS group, specified as 2 or 4. This property is the higher layer parameter *sampleDensity*.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `1`.

Data Types: `double`

`NumPTRSGroups` — Number of PT-RS groups

2 (default) | 4 | 8

Number of PT-RS groups, specified as 2, 4, or 8. This property is the higher layer parameter *sampleDensity*.

When this property is set to 8, the number of PT-RS samples set by the `NumPTRSSamples` property must be set to 4.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `1`.

Data Types: `double`

`REOffset` — Resource element offset

`'00'` (default) | `'01'` | `'10'` | `'11'`

Resource element offset, specified as `'00'`, `'01'`,`'10'`, or `'11'`. This property is the higher layer parameter *resourceElementOffset*.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `0`.

Data Types: `char` | `string`

`PTRSPortSet` — PT-RS antenna port set

`[]` (default) | nonnegative integer | two-element vector of nonnegative integers

PT-RS antenna port set, specified as a two-element vector of nonnegative integers. Specify `[]` to set this property to the lowest value in the `DMRSPortSet` property of `nrPUSCHDMRSConfig` object. This usage of `[]` value is applicable only when `nrPUSCHDMRSConfig` object is used as a property of `nrPUSCHConfig` object.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `0`.

Data Types: `double`

`NID` — PT-RS scrambling identity

`[]` (default) | integer from 0 to 1007

PT-RS scrambling identity, specified as an integer from 0 to 1007. Specify `[]` to set this property equal to the `NRSID` property of `nrPUSCHDMRSConfig` object.

Dependencies

This property applies only when `nrPUSCHPTRSConfig` is a property of `nrPUSCHConfig` with `TransformPrecoding` set to `1`.

Data Types: `double`

Examples

Create PUSCH PT-RS Object

Create a default PUSCH configuration object. Enable the PT-RS configuration and transform precoding for a DFT-s-OFDM waveform.

```
pusch = nrPUSCHConfig;
pusch.EnablePTRS = 1;
pusch.TransformPrecoding = 1;
```

Create a default PT-RS configuration object for the PUSCH. Set number of PT-RS samples to 4, number of PT-RS groups to 8, and PT-RS scrambling identity to 750.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to the PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Display the properties for PUSCH PT-RS configuration object and PUSCH configuration object, respectively.

```
disp(pusch)
```

```
  nrPUSCHConfig with properties:

            NSizeBWP: []
           NStartBWP: []
          Modulation: 'QPSK'
           NumLayers: 1
         MappingType: 'A'
    SymbolAllocation: [0 14]
               PRBSet: [1x52 double]
   TransformPrecoding: 1
   TransmissionScheme: 'nonCodebook'
      NumAntennaPorts: 1
                 TPMI: 0
     FrequencyHopping: 'neither'
    SecondHopStartPRB: 1
        BetaOffsetACK: 20
       BetaOffsetCSI1: 6.2500
       BetaOffsetCSI2: 6.2500
           UCIScaling: 1
```

```
                  NID: []
                 RNTI: 1
                 DMRS: [1x1 nrPUSCHDMRSConfig]
           EnablePTRS: 1
                 PTRS: [1x1 nrPUSCHPTRSConfig]
```

```
disp(pusch.PTRS)
```

```
  nrPUSCHPTRSConfig with properties:

       TimeDensity: 1
    NumPTRSSamples: 4
     NumPTRSGroups: 8
               NID: 750
```

Generate PUSCH PT-RS Indices for Codebook-Based Transmission

Create a carrier configuration object with default properties. This object corresponds to a 10 MHz carrier with 15 kHz subcarrier spacing.

```
carrier = nrCarrierConfig;
```

Create a PUSCH configuration object with codebook-based transmission and enable the PT-RS configuration. Set the number of antenna ports to 4 and transform precoding to 0. When transform precoding is 0, the waveform type is cyclic-prefix orthogonal frequency division multiplexing (CP-OFDM).

```
pusch = nrPUSCHConfig;
pusch.TransformPrecoding = 0;
pusch.TransmissionScheme = 'codebook';
pusch.NumAntennaPorts = 4;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.FrequencyDensity = 4;
ptrs.REOffset = '11';
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS indices in subscript form

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 312x3 uint32 matrix

    21     1     1
    69     1     1
   117     1     1
   165     1     1
   213     1     1
```

```
   261     1     1
   309     1     1
   357     1     1
   405     1     1
   453     1     1
     ⋮
```

Generate PUSCH PT-RS Symbols and Indices

Create a carrier configuration object with 30 kHz subcarrier spacing and 5 MHz transmission bandwidth.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 30;
carrier.NSizeGrid = 11;
```

Create a PUSCH configuration object with intraslot frequency hopping and enable the PT-RS configuration. Set the transform precoding to 1, starting physical resource blocks (PRB) index of the second hop to 3 and PRB set to 0:5. When transform precoding is 1, the waveform type is discrete fourier transform spread orthogonal frequency division multiplexing (DFT-s-OFDM).

```
pusch = nrPUSCHConfig;
pusch.PRBSet = 0:5;
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'intraSlot';
pusch.SecondHopStartPRB = 3;
pusch.EnablePTRS = 1;
```

Create a PUSCH phase tracking reference signal (PT-RS) configuration object with specified properties.

```
ptrs = nrPUSCHPTRSConfig;
ptrs.TimeDensity = 2;
ptrs.NumPTRSSamples = 4;
ptrs.NumPTRSGroups = 8;
ptrs.NID = 750;
```

Assign the PUSCH PT-RS configuration object to PTRS property of PUSCH configuration object.

```
pusch.PTRS = ptrs;
```

Generate PUSCH PT-RS symbols of data type single.

```
sym = nrPUSCHPTRS(carrier,pusch,'OutputDataType','single')
```

```
sym = 192x1 single column vector

   0.7071 + 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
   0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
```

```
   0.7071 + 0.7071i
   0.7071 + 0.7071i
  -0.7071 + 0.7071i
      ⋮
```

Generate PUSCH PT-RS indices in subscript form.

```
ind = nrPUSCHPTRSIndices(carrier,pusch,'IndexStyle','subscript')
```

```
ind = 192x3 uint32 matrix

    1    1    1
    2    1    1
    3    1    1
    4    1    1
   12    1    1
   13    1    1
   14    1    1
   15    1    1
   21    1    1
   22    1    1
      ⋮
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPUSCH` | `nrPUSCHDecode` | `nrPUSCHIndices` | `nrPUSCHPTRS` | `nrPUSCHPTRSIndices`

Objects

`nrCarrierConfig` | `nrPUSCHConfig` | `nrPUSCHDMRSConfig`

Introduced in R2020a

nrSCSCarrierConfig

SCS carrier configuration parameters for 5G waveform generation

Description

The `nrSCSCarrierConfig` object sets subcarrier spacing (SCS) carrier configuration parameters for a specific OFDM numerology. Use this object to set the `SCSCarriers` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

This object defines the carrier SCS, bandwidth, and offset parameters from point A, which is the center of subcarrier 0 in the common resource block 0 (CRB 0). By default, this object specifies a 10 MHz carrier corresponding to 52 resource blocks (RBs) and 15 kHz SCS.

Creation

Syntax

```
carrier = nrSCSCarrierConfig
carrier = nrSCSCarrierConfig(Name,Value)
```

Description

`carrier = nrSCSCarrierConfig` creates a default SCS carrier configuration object for 5G waveform generation.

`carrier = nrSCSCarrierConfig(Name,Value)` sets properties on page 3-103 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'SubcarrierSpacing',30,'NSizeGrid',273` specifies a 100 MHz carrier corresponding to 273 resource blocks (RBs) and 30 kHz SCS.

Properties

`SubcarrierSpacing` — Subcarrier spacing in kHz

`15` (default) | `30` | `60` | `120` | `240`

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as `15`, `30`, `60`, `120`, or `240`.

Data Types: `double`

`NSizeGrid` — Number of RBs in carrier resource grid

`52` (default) | integer from 1 to 275

Number of RBs in the carrier resource grid, specified as an integer from 1 to 275. The default value of `52` corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz SCS.

Data Types: `double`

`NStartGrid` — Start of carrier resource grid relative to CRB 0
`0` (default) | integer from 0 to 2199

Start of carrier resource grid relative to CRB 0, specified as an integer from 0 to 2199. This property is the higher-layer parameter *offsetToCarrier*.

Data Types: `double`

Examples

Configure SCS Carriers with Mixed Numerologies

Create a default SCS carrier configuration object, which configures a 10 MHz carrier with 15 kHz SCS.

```
scs1 = nrSCSCarrierConfig;
```

Create an SCS carrier configuration object, which configures a 100 MHz carrier with 30 kHz SCS.

```
scs2 = nrSCSCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',273);
```

Create a downlink carrier configuration object, specifying the two SCS carrier configurations.

```
cfgDL = nrDLCarrierConfig('SCSCarriers',{scs1,scs2});
```

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig` | `nrWavegenBWPConfig`

Introduced in R2020b

nrSearchSpaceConfig

Search space set configuration parameters

Description

The `nrSearchSpaceConfig` object sets search space set configuration parameters for the physical downlink control channel (PDCCH), as defined in TS 38.213 Section 10 [1]. Use this object when setting the `SearchSpace` property of the `nrPDCCHConfig` or `nrDLCarrierConfig` objects.

Creation

Syntax

```
cfgSS = nrSearchSpaceConfig
cfgSS = nrSearchSpaceConfig(Name,Value)
```

Description

`cfgSS = nrSearchSpaceConfig` creates a search space set configuration object with default properties.

`cfgSS = nrSearchSpaceConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'Duration',3,'NumCandidates',[5 5 3 2 1]` configures the search space set over three consecutive slots with the specified number of candidates at each aggregation level. Unspecified properties take their default values.

Properties

`SearchSpaceID` — Search space set ID

1 (default) | nonnegative integer

Search space set ID, specified as a nonnegative integer.

Data Types: `double`

`Label` — Name of search space set configuration

`'SearchSpace1'` (default) | character array | string scalar

Name of the search space set configuration, specified as a character array or string scalar. Use this property to set a mnemonic description to the search space set configuration.

Data Types: `char` | `string`

`CORESETID` — Associated CORESET ID for search space

1 (default) | integer from 0 to 11

Associated CORESET ID for the search space, specified as an integer from 0 to 11. When this object and `nrCORESETConfig` object specify the `SearchSpace` and `CORESET` properties, respectively, of the same `nrPDCCHConfig` object, the `CORESETID` properties of these objects must match.

Data Types: `double`

`SearchSpaceType` — Search space type
`'ue'` (default) | `'common'`

Search space type, specified as `'ue'` or `'common'`.

Data Types: `char` | `string`

`StartSymbolWithinSlot` — First symbol in monitored slot
`0` (default) | integer from 0 to 13

First symbol in monitored slot, specified as an integer from 0 to 13. Values from 0 to 11 apply only to extended cyclic prefix. When setting this property, the CORESET must fit within a single slot in terms of the associated CORESET duration.

Data Types: `double`

`SlotPeriodAndOffset` — Slot period and offset for PDCCH monitoring
`[1 0]` (default) | 1-by-2 integer vector

Slot period and offset for PDCCH monitoring, specified as a 1-by-2 integer vector. The first vector element specifies the period. The period must be a positive integer greater than or equal to the search space duration specified by the `Duration` property. The second vector element specifies the offset with respect to the period. The offset must be a nonnegative integer less than the period (the first vector element).

Data Types: `double`

`Duration` — Search space duration in slots
`1` (default) | integer from 0 to 2559

Search space duration in slots, specified as an integer from 0 to 2559. This property specifies the number of consecutive slots that the search space lasts within each period. The value of this property must be less than or equal to the slot period specified by the `SlotPeriodAndOffset` property.

Data Types: `double`

`NumCandidates` — Number of candidates per aggregation level
`[8 8 4 2 1]` (default) | 1-by-5 integer vector

Number of candidates per aggregation level, specified as a 1-by-5 integer vector. For each aggregation level, you can specify 0, 1, 2, 3, 4, 5, 6, or 8 candidates. The vector element values correspond to the number of candidates for aggregation levels *AL1*, *Al2*, *AL4*, *Al8*, and *AL16*, respectively.

Data Types: `double`

Examples

Generate PDCCH DM-RS Symbols for All Candidates and Aggregation Levels

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure the CORESET with 6 frequency resources, a duration of 3 OFDM symbols, and a REG bundle size of 3.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.REGBundleSize = 3;
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 6;
pdcch.NSizeBWP = 36;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS symbols for all candidates and aggregation levels.

```
[~,allDMRS] = nrPDCCHSpace(carrier,pdcch)
```

```
allDMRS=5×1 cell array
    { 18x5 double}
    { 36x5 double}
    { 72x3 double}
    {144x2 double}
    {288x1 double}
```

Verify that the number of generated candidates for the PDCCH DM-RS symbols at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRS{1},2) ...
    size(allDMRS{2},2) ...
    size(allDMRS{3},2) ...
    size(allDMRS{4},2) ...
    size(allDMRS{5},2)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

Generate PDCCH DM-RS Indices for All Candidates and Aggregation Levels

Configure a carrier grid of 60 resource blocks (RBs), where the starting RB index relative to the common resource block 0 (CRB 0) is 3.

```
carrier = nrCarrierConfig;
carrier.NStartGrid = 3;
carrier.NSizeGrid = 60;
```

Configure noninterleaved CORESET with 6 frequency resources and a duration of 3 OFDM symbols.

```
crst = nrCORESETConfig;
crst.FrequencyResources = ones(1,6);
crst.Duration = 3;
crst.CCEREGMapping = 'noninterleaved';
```

Configure the search space set for the PDCCH with the specified number of candidates at each aggregation level.

```
cfgSS = nrSearchSpaceConfig;
cfgSS.NumCandidates = [5 5 3 2 1];
```

Configure the PDCCH with the specified bandwidth part, CORESET, and search space set.

```
pdcch = nrPDCCHConfig;
pdcch.NStartBWP = 5;
pdcch.NSizeBWP = 48;
pdcch.CORESET = crst;
pdcch.SearchSpace = cfgSS;
```

Generate PDCCH DM-RS resource element indices for all candidates and aggregation levels using 1-based, subscript indexing form relative to the BWP grid.

```
[~,~,allDMRSInd] = nrPDCCHSpace(carrier,pdcch, ...
    'IndexOrientation','bwp','IndexStyle','subscript')
```

```
allDMRSInd=5×1 cell array
    { 18x3x5 uint32}
    { 36x3x5 uint32}
    { 72x3x3 uint32}
    {144x3x2 uint32}
    {288x3   uint32}
```

Verify that the number of generated candidates for PDCCH DM-RS indices at each aggregation level matches the number of candidates specified by the search space set.

```
numCandidates = [...
    size(allDMRSInd{1},3) ...
    size(allDMRSInd{2},3) ...
    size(allDMRSInd{3},3) ...
    size(allDMRSInd{4},3) ...
    size(allDMRSInd{5},3)];
isequaln(cfgSS.NumCandidates,numCandidates)
```

```
ans = logical
   1
```

References

[1] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`nrPDCCH` | `nrPDCCHResources` | `nrPDCCHSpace`

Objects

`nrCORESETConfig` | `nrCarrierConfig` | `nrDLCarrierConfig` | `nrPDCCHConfig`

Introduced in R2020a

nrSRSConfig

SRS configuration parameters

Description

The `nrSRSConfig` object sets sounding reference signal (SRS) configuration parameters, as defined in TS 38.211 Section 6.4.1.4 [1].

Creation

Syntax

```
srs = nrSRSConfig
srs = nrSRSConfig(Name,Value)
```

Description

`srs = nrSRSConfig` creates an SRS configuration object with default properties.

`srs = nrSRSConfig(Name,Value)` specifies properties using one or more name-value pair arguments. Enclose each property in quotes. For example, `'NumSRSPorts',2,'NumSRSSymbols',4` specifies a two-port SRS transmission of 4 OFDM symbols. Unspecified properties take their default values.

Properties

Configurable SRS Properties

`NumSRSPorts` — Number of SRS antenna ports
1 (default) | 2 | 4

Number of SRS antenna ports, specified as 1, 2, or 4.

Data Types: `double`

`NumSRSSymbols` — Number of OFDM symbols
1 (default) | 2 | 4

Number of OFDM symbols allocated to the SRS in a slot, specified as 1, 2, or 4.

Data Types: `double`

`SymbolStart` — 0-based index of first OFDM symbol
13 (default) | integer from 6 to 13

0-based index of first OFDM symbol in the SRS within a slot, specified as one of these options:

- Integer from 8 to 13 — Use this option for normal cyclic prefix.

- Integer from 6 to 11 — Use this option for extended cyclic prefix.

The SRS must be allocated within the last 6 OFDM symbols of a slot. For the SRS symbols and index generation, set the cyclic prefix of the carrier by using the `CyclicPrefix` property of the `nrCarrierConfig` object.

Data Types: `double`

`KTC` — Transmission comb number
2 (default) | 4

Transmission comb number, in subcarriers, specified as 2 or 4. The object allocates the SRS sequence every `KTC` number of subcarriers.

Data Types: `double`

`KBarTC` — Transmission comb offset
0 (default) | integer from 0 to (`KTC` – 1)

Transmission comb offset, in subcarriers, specified as an integer from 0 to (`KTC` – 1). This property specifies a frequency shift within the comb.

Data Types: `double`

`CyclicShift` — Cyclic shift offset
0 (default) | integer from 0 to 11

Cyclic shift offset, specified as an integer from 0 to 11. This property determines the cyclic shift applied to the SRS sequence for each antenna port. This property corresponds to parameter n_{SRS}^{cs} in TS 38.211 Section 6.4.1.4.2.

Set the cyclic shift offset in relation to the transmission comb property, `KTC`:

- If `KTC` is set to 2, set `CyclicShift` to an integer from 0 to 7.
- If `KTC` is set to 4, set `CyclicShift` to an integer from 0 to 11.

For multiport SRS transmissions, the `nrSRS` function applies consecutive cyclic shift numbers for each port, modulo 8 or modulo 12, depending on the `KTC` property.

Data Types: `double`

`FrequencyStart` — Frequency-domain offset
0 (default) | integer from 0 to 271

Frequency-domain offset of the SRS, in terms of a physical resource block (PRB) index relative to the carrier, specified as an integer from 0 to 271. `FrequencyStart` is analogous to parameter n_{shift} from TS 38.211 Section 6.4.1.4.3.

This property, the additional circular frequency-domain offset property `NRRC`, and the bandwidth configuration parameters in TS 38.211 Table 6.4.1.4.3-1 determine the actual frequency-domain location of the SRS. For more information, see “NR SRS Configuration”.

Data Types: `double`

`NRRC` — Additional circular frequency-domain offset
0 (default) | integer from 0 to 67

Additional circular frequency-domain offset of the SRS, in multiple of 4 PRBs, specified as an integer from 0 to 67.

This property, the frequency domain offset property `FrequencyStart`, and the bandwidth configuration parameters in TS 38.211 Table 6.4.1.4.3-1 determine the actual frequency-domain location of the SRS. For more information, see “NR SRS Configuration”.

Data Types: `double`

`CSRS` — Row index of bandwidth configuration table
`0` (default) | integer from 0 to 63

Row index of bandwidth configuration table from TS 38.211 Table 6.4.1.4.3-1, specified as an integer from 0 to 63. Use this property with the `BSRS` property to control the bandwidth allocated to the SRS and the frequency hopping pattern. Larger `CSRS` values result in larger SRS bandwidths. The default value of `0` results in a bandwidth of 4 PRBs.

Data Types: `double`

`BSRS` — Column index of bandwidth configuration table
`0` (default) | integer from 0 to 3

Column index of bandwidth configuration table from TS 38.211 Table 6.4.1.4.3-1, specified as an integer from 0 to 3. Use this property with the `CSRS` property to control the bandwidth allocated to the SRS and the frequency hopping pattern. Larger `BSRS` values result is smaller SRS bandwidths.

Data Types: `double`

`BHop` — Frequency hopping index
`0` (default) | integer from 0 to 3

Frequency hopping index, specified as an integer from 0 to 3. Setting this property greater than or equal to the column index of the bandwidth configuration table property, `BSRS`, disables frequency hopping. Larger `BHop` values result in smaller hopping bandwidths.

Data Types: `double`

`Repetition` — Repetition factor of OFDM symbols
1 (default) | 2 | 4

Repetition factor of OFDM symbols, specified as `1`, `2`, or `4`.

- When frequency hopping is enabled, `Repetition` specifies the number of consecutive OFDM symbols in a slot occupied by the SRS in the same frequency resource. Set `Repetition` such that `Repetition` ≤ `NumSRSSymbols`.
- When frequency hopping is disabled, this property is ignored.

Data Types: `double`

`SRSPeriod` — Slot periodicity and offset
`'on'` (default) | `'off'` | [T_{SRS} T_{offset}]

Slot periodicity and offset, specified as one of these values:

- `'on'` — The SRS is present in all slots.
- `'off'` — The SRS is absent in all slots.

- [T_{SRS} T_{offset}] — The presence of the SRS in a given slot depends on the specified slot periodicity, T_{SRS}, and the offset, T_{offset}, based on TS 38.211 Section 6.4.1.4.4. Specify T_{SRS} as `1`, `2`, `4`, `5`, `8`, `10`, `16`, `20`, `32`, `40`, `64`, `80`, `160`, `320`, `640`, `1280`, or `2560`. Specify T_{offset} as a nonnegative integer such that $T_{offset} < T_{SRS}$.

Data Types: `char` | `string` | `double`

`ResourceType` — Time-domain behavior
`'periodic'` (default) | `'semi-persistent'` | `'aperiodic'`

Time-domain behavior of the SRS, specified as `'periodic'`, `'semi-persistent'`, or `'aperiodic'`. Downlink control information (DCI) triggers aperiodic SRS transmissions. When the resource type is aperiodic, the `SRSPeriod` property determines the periodicity and offset of the DCI-triggering signal. An aperiodic resource type also disables interslot frequency hopping.

Data Types: `char` | `string`

`GroupSeqHopping` — Type of SRS symbol hopping
`'neither'` (default) | `'groupHopping'` | `'sequenceHopping'`

Type of SRS symbol hopping, specified as `'neither'`, `'groupHopping'`, or `'sequenceHopping'`. When either group or sequence hopping is enabled, the group or sequence hopping numbers per OFDM symbol in the SRS transmission are based on a pseudo-random binary sequence (PRBS). Set the scrambling identity for the PRBS by using the `NSRSID` property.

Data Types: `char` | `string`

`NSRSID` — SRS scrambling identity
`0` (default) | integer from 0 to 1023

SRS scrambling identity, specified as an integer from 0 to 1023.

- When the `GroupSeqHopping` property is set to `'neither'`, this property determines the group number.
- When the `GroupSeqHopping` property is set to `'groupHopping'` or `'sequenceHopping'`, this property initializes the PRBS.

Data Types: `double`

Nonconfigurable SRS Properties

The object automatically sets these properties based on configurable SRS property values by using the configuration tables from TS 38.211 Section 6.4.1.4.

`NRB` — Number of RBs allocated for SRS transmission
4 (default) | positive integer

This property is read-only.

Number of RBs allocated for SRS transmission, specified as a positive integer. When frequency hopping is enabled, this property denotes the hopping bandwidth or number of RBs over which the SRS signal hops across multiple time slots.

Data Types: `double`

`NRBPerTransmission` — Number of RBs allocated per SRS OFDM symbol
4 (default) | positive integer

This property is read-only.

Number of RBs allocated per SRS OFDM symbol, specified as a positive integer. When frequency hopping is enabled, this property specifies the allocated bandwidth at each SRS OFDM symbol. When frequency hopping is disabled, this property is equal to the `NRB` property.

Data Types: `double`

SRS Lookup Table

`BandwidthConfigurationTable` — SRS bandwidth configuration table
constant 64-by-9 matrix (default)

This property is read-only.

SRS bandwidth configuration table corresponding to TS 38.211 Table 6.4.1.4.3-1, specified as a constant 64-by-9 matrix.

Examples

Generate and Map SRS Symbols to Carrier Grid

Configure the SRS and the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
srs = nrSRSConfig;
```

Generate SRS symbols and indices using the specified carrier and SRS configuration parameters.

```
srsSym = nrSRS(carrier,srs);
srsInd = nrSRSIndices(carrier,srs);
```

Create a carrier grid corresponding to the number of subcarriers, OFDM symbols, and number of antenna ports specified in the configuration objects.

```
K = carrier.NSizeGrid*12;       % Number of subcarriers
L = carrier.SymbolsPerSlot;     % Number of OFDM symbols per slot
P = srs.NumSRSPorts;  % Number of antenna ports
gridSize = [K L P];
```

Initialize the carrier grid for one slot with all zeros.

```
slotGrid = complex(zeros(gridSize));
```

Map the SRS symbols to the carrier grid using the indices.

```
slotGrid(srsInd) = srsSym;
```

Generate SRS Symbols for Two-Port Transmission

Configure the carrier with default configuration parameters.

```
carrier = nrCarrierConfig;
```

Configure a two-port SRS transmission of 4 OFDM symbols.

```
srs = nrSRSConfig;
srs.NumSRSPorts = 2;
srs.NumSRSSymbols = 4;
```

The SRS must be located in the last six symbols of the slot. Set the time-domain starting position of the SRS to 8 and the bandwidth configuration index to 5.

```
srs.SymbolStart = 8;
srs.CSRS = 5;
```

Generate SRS symbols for the specified carrier and SRS configuration parameters.

```
[sym,info] = nrSRS(carrier,srs);
```

Verify that the symbols vector contains two columns corresponding to the two-port transmission.

```
size(sym)
```

```
ans = 1×2

   480     2
```

Verify the number of SRS symbols per port.

```
isequal(info.SeqLength*srs.NumSRSSymbols,size(sym,1))
```

```
ans = logical
   1
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

nrSRS | nrSRSIndices

Objects

nrCarrierConfig

Topics

“NR SRS Configuration”

Introduced in R2020a

nrWavegenBWPConfig

BWP configuration parameters for 5G waveform generation

Description

The `nrWavegenBWPConfig` object sets bandwidth part (BWP) configuration parameters in a specific subcarrier spacing (SCS) carrier. Use this object to set the `BandwidthParts` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation. Specify the SCS carrier with the same numerology by using the `SCSCarriers` property of the same `nrDLCarrierConfig` object.

This object defines the SCS of the carrier containing the BWP, the size of the BWP, the offset from the common resource block 0 (CRB 0), and the cyclic prefix. For a SCS of 60 kHz, you can specify either normal or extended cyclic prefix.

Creation

Syntax

```
bwp = nrWavegenBWPConfig
bwp = nrWavegenBWPConfig(Name,Value)
```

Description

`bwp = nrWavegenBWPConfig` creates a default BWP configuration object for 5G waveform generation.

`bwp = nrWavegenBWPConfig(Name,Value)` specifies properties on page 3-117 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'SubcarrierSpacing',30` specifies a SCS of 30 kHz for the carrier containing the BWP.

Properties

`BandwidthPartID` — ID of BWP configuration

`1` (default) | nonnegative integer

ID of BWP configuration, specified as a nonnegative integer.

Data Types: `double`

`Label` — Name of BWP configuration

`'BWP1'` (default) | character array | string scalar

Name of BWP configuration, specified as a character array or string scalar. Use this property to set a mnemonic description to the BWP configuration.

Data Types: `char` | `string`

SubcarrierSpacing — Subcarrier spacing in kHz
15 (default) | 30 | 60 | 120

Subcarrier spacing in kHz, for all channels and reference signals of the carrier, specified as 15, 30, 60, or 120.

Data Types: double

CyclicPrefix — Cyclic prefix length
'normal' (default) | 'extended'

Cyclic prefix length, specified as one of these options.

- 'normal' — Use this value to specify normal cyclic prefix. This option corresponds to 14 OFDM symbols in a slot.
- 'extended' — Use this value to specify extended cyclic prefix. This option corresponds to 12 OFDM symbols in a slot. For the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length applies for only 60 kHz subcarrier spacing.

Data Types: char | string

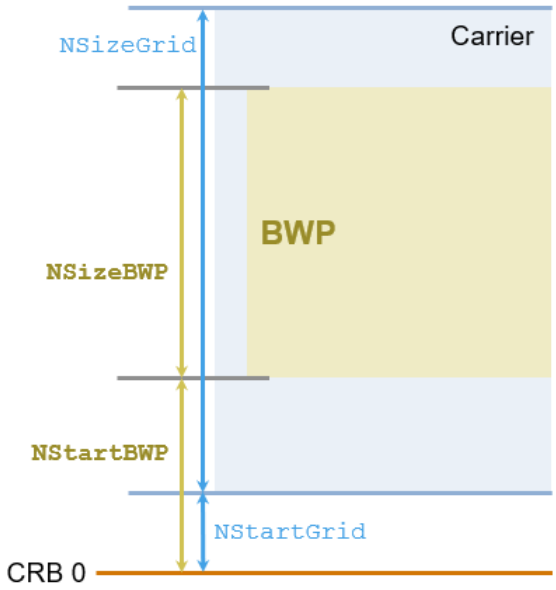
NSizeBWP — Number of RBs in BWP resource grid
52 (default) | integer from 1 to 275

Number of resource blocks (RBs) in the BWP resource grid, specified as an integer from 1 to 275. This property must be less than or equal to the size of the SCS carrier with the same SCS, specified by the SCSCarriers property of the nrDLCarrierConfig object.

Data Types: double

NStartBWP — Start of BWP resource grid relative to CRB 0
0 (default) | nonnegative integer

Start of the BWP resource grid relative to CRB 0, specified as a nonnegative integer. Set this property relative to the SCS carrier such that the property value is in this range: NStartGrid ≤ NStartBWP ≤ (NStartGrid + NSizeGrid – NSizeBWP). NStartGrid and NSizeGrid are properties of the SCS carrier with the same SCS, specified by the SCSCarriers property of the nrDLCarrierConfig object. This figure shows where in the carrier the BWP is located in terms of this property and the NSizeBWP property.



Data Types: `double`

Examples

Configure Bandwidth Part for Default SCS

Create a BWP configuration object for the default SCS carrier of 15 kHz. Specify the number of RBs in the BWP resource grid and the start of the BWP resource grid relative to the CRB 0.

```
bwp = nrWavegenBWPConfig;
bwp.NSizeBWP = 50;
bwp.NStartBWP = 12;
```

Create a downlink carrier configuration object, specifying the previously defined BWP configuration.

```
cfgDL = nrDLCarrierConfig('BandwidthParts',{bwp});
```

Configure Two Bandwidth Parts for Two SCS Carriers

Create a default SCS carrier configuration object, which configures a 10 MHz carrier with 15 kHz SCS.

```
scs1 = nrSCSCarrierConfig;
```

Create an SCS carrier configuration object, which configures a 100 MHz carrier with 30 kHz SCS.

```
scs2 = nrSCSCarrierConfig('SubcarrierSpacing',30,'NSizeGrid',273);
```

Create two BWP configurations, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig;
bwp2 = nrWavegenBWPConfig('SubcarrierSpacing',scs2.SubcarrierSpacing, ...
    'NSizeBWP',12,'NStartBWP',30);
```

Create a downlink carrier configuration object, specifying the previously defined BWP and corresponding SCS carrier configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'SCSCarriers',{scs1,scs2}, ...
    'BandwidthParts',{bwp1,bwp2});
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig` | `nrSCSCarrierConfig`

Introduced in R2020b

nrWavegenCSIRSConfig

CSI-RS configuration parameters for 5G waveform generation

Description

The `nrWavegenCSIRSConfig` object sets channel state information reference signal (CSI-RS) configuration parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resources, as defined in TS 38.211 Section 7.4.1.5 [1]. Use this object to set the `CSIRS` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

By default, the object defines an NZP-CSI-RS resource configured for two antenna ports with a CDM type of FD-CDM2 and density 1, corresponding to CSI-RS resource defined in row 3 of TS 38.211 Table 7.4.1.5.3-1.

Creation

Syntax

```
csirs = nrWavegenCSIRSConfig
csirs = nrWavegenCSIRSConfig(Name,Value)
```

Description

`csirs = nrWavegenCSIRSConfig` creates a default CSI-RS configuration object for 5G waveform generation.

`csirs = nrWavegenCSIRSConfig(Name,Value)` sets propertiesproperties on page 3-121 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'CSIRSType',{'zp','nzp','zp'}` specifies three CSI-RS resources.

Properties

`Enable` — Enable CSI-RS

`0` (default) | `1`

Enable CSI-RS in 5G waveform generation, specified as one of these values.

- `0` — Disable CSI-RS.
- `1` — Enable CSI-RS.

Data Types: `double` | `logical`

`Label` — Name of CSI-RS configuration

`'CSIRS1'` (default) | character array | string scalar

Name of CSI-RS configuration, specified as a character array or string scalar. Use this property to set a mnemonic description to the CSI-RS configuration.

Data Types: `char` | `string`

`Power` — Power scaling of CSI-RS in dB
`0` (default) | real scalar

Power scaling of CSI-RS in dB, specified as a real scalar. Use this property to scale the power of the CSI-RS in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of bandwidth part
`1` (default) | nonnegative integer

ID of bandwidth part (BWP), containing the configured CSI-RS, specified as a nonnegative integer. Use this property to associate this CSI-RS configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`CSIRSType` — Type of one or more CSI-RS resource configurations
`'nzp'` (default) | `'zp'` | cell array | string scalar | string array

Type of one or more CSI-RS resource configurations, specified as one of these options.

- `'nzp'` — Use this option to specify a single NZP-CSI-RS resource.
- `'zp'` — Use this option to specify a single ZP-CSI-RS resource.
- Cell array with elements `'nzp'` or `'zp'` — Use this option to specify multiple CSI-RS resources.

Alternatively, you can specify this property by using `"nzp"` and `"zp"` as string scalars or as elements of a string array.

The number of CSI-RS resource configurations is equal to the number of values provided for this property.

Data Types: `cell` | `string` | `char`

`CSIRSPeriod` — Slot periodicity and offset of CSI-RS resource
`'on'` (default) | `'off'` | vector of integers | cell array | string scalar | string array

Slot periodicity and offset of the CSI-RS resource, specified as one of these options.

For Single CSI-RS Resource

- `'on'` — Use this option to indicate that the resource is present in all slots.
- `'off'` — Use this option to indicate that the resource is absent in all slots.
- Vector of integers of the form [*Tcsi-rs Toffset*] — Use this option to specify slot periodicity *Tcsi-rs* and offset *Toffset* for scheduling the CSI-RS resource in specific slots.

  Tcsi-rs is 4, 5, 8, 10, 16, 20, 32, 40, 64, 80, 160, 320, or 640. For a particular value of *Tcsi-rs*, the value of *Toffset* is in the range from 0 to *Tcsi-rs*–1.

For Multiple CSI-RS Resources

- Cell array with elements `'on'`, `'off'`, or [*Tcsi-rs Toffset*] — The number of elements in the cell array must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the cell array contains only one element, the specified value applies to all CSI-RS resources.

Alternatively, you can specify this property by using `"on"` and `"off"` as string scalars or as elements of a string array.

This property is the higher-layer parameter *CSI-ResourcePeriodicityAndOffset* or *slotConfig* defined in the *CSI-RS-CellMobility* IE.

Data Types: `cell` | `string` | `char` | `double`

`RowNumber` — Row number of CSI-RS resource
`3` (default) | integer from 1 to 18 | vector of integers

Row number of CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 18

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 18 — The number of vector elements must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`Density` — Frequency density of CSI-RS resource
`'one'` (default) | `'three'` | `'dot5even'` | `'dot5odd'` | cell array | string scalar | string array

Frequency density of the CSI-RS resource, as defined in TS 38.211 Table 7.4.1.5.3-1, specified as one of these options.

For Single CSI-RS Resource

- `'one'` — This option corresponds to $\rho = 1$ from the specified table.
- `'three'` — This option corresponds to $\rho = 3$ from the specified table.
- `'dot5even'` — This option corresponds to $\rho = 0.5$ from the specified table with even resource block (RB) allocation regarding the common resource block 0 (CRB 0).
- `'dot5odd'` — This option corresponds to $\rho = 0.5$ from the specified table with odd RB allocation regarding CRB 0.

For Multiple CSI-RS Resources

- Cell array of the character vectors `'one'`, `'three'`, `'dot5even'`, or `'dot5odd'` — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Alternatively, you can specify this property by using `"one"`, `"three"`, `"dot5even"`, and `"dot5odd"` as string scalars or as elements of a string array.

The value of ρ is the higher-layer parameter *density* defined in the *CSI-RS-ResourceMapping* IE or the *CSI-RS-CellMobility* IE.

Data Types: `cell` | `string` | `char`

`SymbolLocations` — Time-domain locations of CSI-RS resource
`0` (default) | integer from 0 to 13 | vector of integers | cell array

Time-domain locations of the CSI-RS resource (l_0 and l_1 values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 13 — This option corresponds to the l_0 value in the specified table.
- Vector of integers of the form [l_0 l_1] or [l_0; l_1], where l_0 and l_1 are the corresponding l_0 and l_1 values in the specified table — The l_1 values are required only in table rows 13, 14, 16, and 17. l_0 is an integer from 0 to 13, and l_1 is an integer from 2 to 12.

For Multiple CSI-RS Resources

- Cell array of l_0 values or vectors of the form [l_0 l_1] or [l_0; l_1] — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

The values of l_0 and l_1 are the higher-layer parameters *firstOFDMSymbolInTimeDomain* and *firstOFDMSymbolInTimeDomain2*, respectively, in the *CSI-RS-ResourceMapping* IE or *CSI-RS-Resource-Mobility* IE.

Data Types: `double`

`SubcarrierLocations` — Frequency-domain locations of CSI-RS resource
`0` (default) | numeric vector | cell array

Frequency-domain locations of the CSI-RS resource (k_i values in the TS 38.211 Table 7.4.1.5.3-1), specified as one of these options.

For Single CSI-RS Resource

- Numeric vector with elements 1, 2, 3, 4, or 6 — The vector elements correspond to the possible lengths of subcarrier locations.

For Multiple CSI-RS Resources

- Cell array of numeric vectors with elements 1, 2, 3, 4, or 6 — The number of elements in the cell array must equal the number of CSI-RS resources specified by the `CSIRSType` property.

Data Types: `double`

`NumRB` — CSI-RS resource bandwidth
52 (default) | integer from 1 to 275 | vector of integers

CSI-RS resource bandwidth, in terms of the number of allocated RBs, specified as one of these options.

For Single CSI-RS Resource

- Integer from 1 to 275

For Multiple CSI-RS Resources

- Vector of integers in the range from 1 to 275 — The number of vector elements must equal to one or the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *nrOfRBs* in *FrequencyOccupation* IE or *nrOfPRBs* in *CSI-RS-ResourceConfigMobility* IE.

Data Types: `double`

`RBOffset` — Starting RB index of CSI-RS resource allocation

`0` (default) | integer from 0 to 274 | vector of integers

Starting RB index of the CSI-RS resource allocation, relative to the carrier resource grid, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 274

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 274 — The number of vector elements must be equal to one or the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

Note Specify this property relative to the carrier start, which contains the BWP specified by the `BandwidthPartID` property.

Data Types: `double`

`NID` — Scrambling identity

`0` (default) | integer from 0 to 1023 | vector of integers

Scrambling identity, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 1023

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 1023 — The number of vector elements must be one or equal the number of CSI-RS resources specified by the `CSIRSType` property. When the vector contains only one element, the specified value applies to all CSI-RS resources.

This property is the higher-layer parameter *scramblingID* in *NZP-CSI-RS-Resource* IE or *sequenceGenerationConfig* in *CSI-RS-ResourceConfigMobility* IE.

When the `CSIRSType` property defines only ZP resources, this property is hidden.

Data Types: `double`

Examples

Configure CSI-RS for 5G Downlink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies and a custom number of resource blocks.

```
carriers = {
    nrSCSCarrierConfig('SubcarrierSpacing',15,'NStartGrid',10,'NSizeGrid',100), ...
```

```
    nrSCSCarrierConfig('SubcarrierSpacing',30,'NStartGrid',0,'NSizeGrid',70)};
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp = {
    nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',15,'NStartBWP',10,'NSizeBWP',80),
    nrWavegenBWPConfig('BandwidthPartID',2,'SubcarrierSpacing',30,'NStartBWP',0,'NSizeBWP',60)};
```

Create two CSI-RS configuration objects, one for each of the BWP.

```
 csirs = {
     nrWavegenCSIRSConfig('BandwidthPartID',1,'RowNumber',2,'RBOffset',10), ...
     nrWavegenCSIRSConfig('BandwidthPartID',2,'Density','three','RowNumber',4)};
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'SCSCarriers',carriers, ...
    'BandwidthParts',bwp, ...
    'CSIRS',csirs);
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig` | `nrWavegenBWPConfig`

Introduced in R2020b

nrWavegenPDCCHConfig

PDCCH configuration parameters for 5G waveform generation

Description

The `nrWavegenPDCCHConfig` object sets physical downlink control channel (PDCCH) configuration parameters, as defined in TS 38.211 Section 7.3.2 [1] and TS 38.213 Section 10 [2]. Use this object to set the `PDCCH` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

Creation

Syntax

```
pdcch = nrWavegenPDCCHConfig
pdcch = nrWavegenPDCCHConfig(Name,Value)
```

Description

`pdcch = nrWavegenPDCCHConfig` creates a default PDCCH configuration object for 5G waveform generation.

`pdcch = nrWavegenPDCCHConfig(Name,Value)` specifies properties on page 3-127 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'AggregationLevel',2` configures the PDCCH with the specified aggregation level.

Properties

`Enable` — Enable PDCCH

`1` (default) | `0`

Enable PDCCH in 5G waveform generation, specified as one of these values.

- `1` — Enable PDCCH.
- `0` — Disable PDCCH.

Data Types: `double` | `logical`

`Label` — Name of PDCCH configuration

`'PDCCH1'` (default) | character array | string scalar

Name of the PDCCH configuration, specified as a character array or string scalar. Use this property to set a mnemonic description to the PDCCH configuration.

Data Types: `char` | `string`

`Power` — Power scaling of PDCCH in dB

`0` (default) | real scalar

Power scaling of the PDCCH in dB, specified as a real scalar. Use this property to scale the power of the PDCCH in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of bandwidth part
`1` (default) | nonnegative integer

ID of the bandwidth part (BWP), containing the configured PDCCH, specified as a nonnegative integer. Use this property to associate this PDCCH configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`SearchSpaceID` — ID of search space set
`1` (default) | nonnegative integer

ID of the search space set containing the configured PDCCH, specified as a nonnegative integer. Use this property to associate this PDCCH configuration with one of the search space set configurations specified by the `SearchSpaces` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`AggregationLevel` — PDCCH aggregation level
`8` (default) | `1` | `2` | `4` | `16`

PDCCH aggregation level, specified as 1, 2, 4, 8, or 16.

Data Types: `double`

`AllocatedCandidate` — Candidate used for PDCCH instance
`1` (default) | integer from 1 to 8

Candidate used for the PDCCH instance, specified as an integer from 1 to 8. The value of this property is an index from the set of candidates specified for the aggregation level by the `NumCandidates` property of the search space specified by the `SearchSpaceID` property.

Data Types: `double`

`SlotAllocation` — Slot allocation in PDCCH period
`0` (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PDCCH period, specified as a nonnegative integer or row vector of nonnegative integers. This property specifies the slot positions of the PDCCH by using 0-based indexing and values smaller than the value of the `Period` property. This slot allocation must be within the slot allocation of the search space specified by the `SearchSpaceID` property.

Data Types: `double`

`Period` — PDCCH allocation period in slots
`1` (default) | nonnegative integer

PDCCH allocation period in slots, specified as a nonnegative integer.

Data Types: `double`

`Coding` — Enable DCI encoding
`1` (default) | nonnegative integer

Enable downlink control information (DCI) encoding, specified as one of these values.

- `1` — Enable DCI encoding.
- `0` — Disable DCI encoding.

Data Types: `double` | `logical`

`DataBlockSize` — Length of DCI in bits

`20` (default) | integer from 0 to 140

Length of DCI in bits, specified as an integer from 0 to 140.

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`DataSource` — Source of DCI contents

`'PN9-ITU'` (default) | `'PN9'` | `'PN11''PN15'` | `'PN23'` | cell array | binary vector

Source of DCI contents, specified as one of these options:

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Data Types: `double` | `cell` | `string` | `char`

`RNTI` — Radio network temporary identifier

`1` (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535. When you set this property to a value greater than 65,519, the object infers this property value to be `0`.

Data Types: `double`

`DMRSScramblingID` — PDCCH DM-RS scrambling identity

`2` (default) | integer from 0 to 65,535 | `[]`

PDCCH demodulation reference signal (DM-RS) scrambling identity, specified as an integer from 0 to 65,535 if the higher layer parameter *pdcch-DMRS-ScramblingID* is configured or as `[]` if *pdcch-DMRS-ScramblingID* is not configured. When you specify this property as `[]`, the object sets the PDCCH DM-RS scrambling identity to the physical layer cell identity specified by the `NCellID` property of the carrier.

Data Types: `double`

`DMRSPower` — Power scaling of PDCCH DM-RS in dB

`0` (default) | real scalar

Power scaling of the PDCCH DM-RS in dB, specified as a real scalar. Use this property to scale the power of the PDCCH DM-RS in the generated 5G waveform. This scaling is additional to the PDCCH-wide power scaling specified by the `Power` property.

Data Types: `double`

Examples

Configure PDCCH for 5G Downlink Waveform Generation

Create a default CORESET configuration object.

```
coreset = nrCORESETConfig;
```

Create a search space set configuration object, associating the search space set with the previously defined CORESET configuration.

```
searchSpace = nrSearchSpaceConfig('CORESETID',coreset.CORESETID);
```

Create a PDCCH configuration object for 5G waveform generation with the specified property values.

```
pdcch = nrWavegenPDCCHConfig( ...
    'SearchSpaceID',searchSpace.SearchSpaceID, ...
    'AggregationLevel',4, ...
    'AllocatedCandidate',2, ...
    'SlotAllocation',[0 2], ...
    'Period',3);
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'CORESET',{coreset}, ...
    'SearchSpaces',{searchSpace}, ...
    'PDCCH',{pdcch});
```

Configure Multiple PDCCH for 5G Downlink Waveform Generation

Create two CORESET configuration objects with unique IDs.

```
coreset1 = nrCORESETConfig('CORESETID',1);
coreset2 = nrCORESETConfig('CORESETID',2);
```

Create three search space set configuration objects with unique IDs. Associate each search space set with one of the previously defined CORESET configurations.

```
searchSpace1 = nrSearchSpaceConfig('SearchSpaceID',1,'CORESETID',coreset1.CORESETID);
searchSpace2 = nrSearchSpaceConfig('SearchSpaceID',2,'CORESETID',coreset1.CORESETID);
searchSpace3 = nrSearchSpaceConfig('SearchSpaceID',3,'CORESETID',coreset2.CORESETID);
```

Create four PDCCH configuration objects for 5G waveform generation. Specify a unique UE and one of the search space set configurations for each PDCCH.

```
pdcch1 = nrWavegenPDCCHConfig('RNTI',1,'SearchSpaceID',searchSpace1.SearchSpaceID);
pdcch2 = nrWavegenPDCCHConfig('RNTI',2,'SearchSpaceID',searchSpace2.SearchSpaceID);
```

```
pdcch3 = nrWavegenPDCCHConfig('RNTI',3,'SearchSpaceID',searchSpace2.SearchSpaceID);
pdcch4 = nrWavegenPDCCHConfig('RNTI',4,'SearchSpaceID',searchSpace3.SearchSpaceID);
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfgDL = nrDLCarrierConfig( ...
    'CORESET',{coreset1,coreset2}, ...
    'SearchSpaces',{searchSpace1,searchSpace2,searchSpace3}, ...
    'PDCCH',{pdcch1,pdcch2,pdcch3,pdcch4});
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrCORESETConfig` | `nrDLCarrierConfig` | `nrPDCCHConfig` | `nrSearchSpaceConfig` | `nrWavegenBWPConfig`

Introduced in R2020b

nrWavegenPDSCHConfig

PDSCH configuration parameters for 5G waveform generation

Description

The `nrWavegenPDSCHConfig` object sets physical downlink shared channel (PDSCH) configuration parameters, as defined in TS 38.211 Sections 7.3.1, 7.4.1.1, and 7.4.1.2 [1]. Use this object to set the `PDSCH` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

This object defines several properties of the PDSCH, including the modulation scheme, layer mapping, target code rate, time-domain and frequency-domain allocation, and virtual resource blocks (VRB) to physical resource blocks (PRBs) interleaving. The object also contains properties of the associated physical reference signals, such as the demodulation reference signal (DM-RS) and the phase tracking reference signal (PT-RS).

By default, the object configures a physical downlink shared channel occupying a 10 MHz bandwidth at a subcarrier spacing (SCS) of 15 kHz (52 resource blocks) and spanning over 14 OFDM symbols in a slot.

Creation

Syntax

```
pdsch = nrWavegenPDSCHConfig
pdsch = nrWavegenPDSCHConfig(Name,Value)
```

Description

`pdsch = nrWavegenPDSCHConfig` creates a default PDSCH configuration object for 5G waveform generation.

`pdsch = nrWavegenPDSCHConfig(Name,Value)` specifies properties on page 3-132 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'NumLayers',7` specifies seven transmission layers.

Properties

`Enable` — Enable PDSCH

`1` (default) | `0`

Enable PDSCH in 5G waveform generation, specified as one of these values.

- `1` — Enable PDSCH.
- `0` — Disable PDSCH.

Data Types: `double` | `logical`

`Label` — Name of PDSCH configuration
`'PDSCH1'` (default) | character array | string scalar

Name of the PDSCH configuration, specified as a character array or string scalar. Use this property to set a mnemonic description to the PDSCH configuration.

Data Types: `char` | `string`

`Power` — Power scaling of PDSCH in dB
`0` (default) | real scalar

Power scaling of the PDSCH in dB, specified as a real scalar. Use this property to scale the power of the PDSCH in the generated 5G waveform.

Data Types: `double`

`BandwidthPartID` — ID of bandwidth part
`1` (default) | nonnegative integer

ID of bandwidth part (BWP), containing the configured PDSCH, specified as a nonnegative integer. Use this property to associate this PDSCH configuration with one of the BWP configurations specified by the `BandwidthParts` property of the `nrDLCarrierConfig` object.

Data Types: `double`

`Modulation` — Modulation scheme
`'QPSK'` (default) | `'16QAM'` | `'64QAM'` | `'256QAM'` | string scalar | string array | cell array of character vectors

Modulation scheme, specified as `'QPSK'`, `'16QAM'`, `'64QAM'`, or `'256QAM'`, a string scalar, a string array, or a cell array of character vectors. This modulation scheme specifies the modulation type of the codewords and the number of bits used per modulation symbol. For one codeword, specify the modulation scheme as a character vector or string scalar. If two codewords are present (`NumLayers` > 4), the same modulation scheme applies to both codewords or you can specify different modulation schemes for each codeword by using a string array or a cell array of character vectors.

Modulation Scheme	Number of Bits Per Symbol
`'QPSK'`	2
`'16QAM'`	4
`'64QAM'`	6
`'256QAM'`	8

Example: `{'QPSK','16QAM'}` or `["QPSK","16QAM"]` specifies different modulation schemes for two codewords.

Data Types: `char` | `string` | `cell`

`NumLayers` — Number of transmission layers
`1` (default) | integer from 1 to 8

Number of transmission layers, specified as an integer from 1 to 8.

- For one codeword, specify an integer from 1 to 4.
- For two codewords, specify an integer from 5 to 8.

Data Types: `double`

`MappingType` — Mapping type
`'A'` (default) | `'B'`

Mapping type of the physical shared channel, specified as `'A'` or `'B'`.

Data Types: `char` | `string`

`ReservedPRB` — Reserved PRBs and OFDM symbols pattern in BWP
default `nrPDSCHReservedConfig` object (default) | cell array of `nrPDSCHReservedConfig` objects

Reserved PRBs and OFDM symbols pattern in the BWP, specified as a cell array of `nrPDSCHReservedConfig` objects.

Data Types: `cell`

`ReservedCORESET` — CORESET IDs for PDSCH rate matching
`[]` (default) | integer from 0 to 11 | vector of integers from 0 to 11

Control-resource set (CORESET) IDs for PDSCH rate matching, specified as `[]`, an integer from 0 to 11, or a vector of integers from 0 to 11. The integers must match the `CORESETID` property values of the `nrCORESETConfig` objects specified by the `nrDLCarrierConfig`.`CORESET` property. When you set this property to a value other than `[]`, this property specifies PDSCH rate matching around the denoted CORESET and associated search spaces.

Data Types: `double`

`SymbolAllocation` — OFDM symbol allocation
`[0 14]` (default) | two-element vector of nonnegative integers

OFDM symbol allocation of the physical shared channel, specified as a two-element vector of nonnegative integers. The first element of this property represents the start of symbol allocation (0-based). The second element represents the number of allocated OFDM symbols.

Data Types: `double`

`SlotAllocation` — Slot allocation in PDSCH period
`[0:9]` (default) | nonnegative integer | row vector of nonnegative integers

Slot allocation in a PDSCH period, specified as a nonnegative integer or row vector of nonnegative integers. This property specifies the slot positions of the PDSCH by using 0-based indexing and values smaller than the value of the `Period` property.

Data Types: `double`

`Period` — PDSCH allocation period in slots
`10` (default) | nonnegative integer

PDSCH allocation period in slots, specified as a nonnegative integer.

Data Types: `double`

`PRBSet` — PRB allocation
`[0:51]` (default) | vector of integers from 0 to 274

PRB allocation of the PDSCH in the BWP, specified as a vector of integers from 0 to 274.

Data Types: `double`

`VRBToPRBInterleaving` — Enable VRB-to-PRB interleaving

`0` (default) | `1`

Enable VRB-to-PRB interleaving, specified as one of these values.

- `0` — Disable VRB-to-PRB interleaving.
- `1` — Enable VRB-to-PRB interleaving.

Data Types: `double` | `logical`

`VRBBundleSize` — VRB bundle size

2 (default) | 4

VRB bundle size, in terms of the number of PRBs for VRB-to-PRB interleaving, specified as 2 or 4.

Dependencies

To enable this property, set the `VRBToPRBInterleaving` property to `1`.

Data Types: `double`

`NID` — PDSCH scrambling identity

`[]` (default) | integer from 0 to 1023

PDSCH scrambling identity, specified as `[]` or an integer from 0 to 1023.

- If the higher layer parameter *dataScramblingIdentityPDSCH* is configured, `NID` must be in the range from 0 to 1023.
- If the higher layer parameter *dataScramblingIdentityPDSCH* is not configured, `NID` must be in the range from 0 to 1007.

When you specify this property as `[]`, the object sets the PDSCH scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier.

Data Types: `double`

`RNTI` — Radio network temporary identifier

1 (default) | integer from 0 to 65,535

Radio network temporary identifier of the user equipment (UE), specified as an integer from 0 to 65,535.

Data Types: `double`

`Coding` — Enable DL-SCH encoding of transport blocks

1 (default) | nonnegative integer

Enable downlink shared channel (DL-SCH) encoding of transport blocks, specified as one of these values.

- `1` — Enable transport block encoding.
- `0` — Disable transport block encoding.

Data Types: `double` | `logical`

`TargetCodeRate` — Target code rate

`0.5137` (default) | numeric scalar | 1-by-2 numeric vector

Target code rate, specified as a numeric scalar or a 1-by-2 numeric vector of values in the interval (0, 1). The default value corresponds to 526/1024. If you specify this property as a scalar, the object applies scalar expansion when processing two transport blocks (`NumLayers` > 4). To specify different target code rates for each transport block, specify this property as a vector.

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`TBScaling` — Codeword scaling factor

`1` (default) | `0.25` | `0.5` | 1-by-2 integer vector

Codeword scaling factor, specified as one of these options.

- For one codeword, specify `0.25`, `0.5`, or `1`.
- For two codewords (`NumLayers` > 4), specify a 1-by-2 integer vector with values `0.25`, `0.5`, or `1`.

Dependencies

To enable this property, set the `Coding` property to `1`.

Data Types: `double`

`XOverhead` — Rate matching overhead

`0` (default) | `6` | `12` | `18`

Rate matching overhead, specified as `0`, `6`, `12`, or `18`.

Data Types: `double`

`RVSequence` — Redundancy version sequence

`[0 2 3 1]` (default) | nonnegative integer | vector of nonnegative integers | two-element cell array

Redundancy version sequence, specified as a nonnegative integer, a vector of nonnegative integers, or a two-element cell array containing unique nonnegative integers. When the sequence is a two-element cell array, the second value only applies to a second codeword (`NumLayers` > 4).

Data Types: `double`

`DataSource` — Source of contents for transport blocks

`'PN9-ITU'` (default) | `'PN9'` | `'PN11''PN15'` | `'PN23'` | cell array | binary vector

Source of contents for transport blocks, specified as one of these options.

- `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- Binary vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Data Types: `double` | `cell` | `string` | `char`

`DMRS` — PDSCH DM-RS configuration parameters
default `nrPDSCHDMRSConfig` object (default) | `nrPDSCHDMRSConfig` object

PDSCH DM-RS configuration parameters, specified as an `nrPDSCHDMRSConfig` object. The `nrWavegenPDSCHConfig` object uses only these `nrPDSCHDMRSConfig` properties:

`DMRSConfigurationType` — DM-RS configuration type
`1` (default) | `2`

DM-RS configuration type, specified as `1` or `2`. This property is the higher-layer parameter *dmrs-Type*.

Data Types: `double`

`DMRSReferencePoint` — Reference point for DM-RS sequence to subcarrier resource mapping
`CRB0` (default) | `PRB0`

Reference point for the DM-RS sequence to subcarrier resource mapping, specified as one of these options.

- `PRB0` — When the reference point is subcarrier 0 of the physical resource block 0 (PRB 0) of the bandwidth part (BWP). Use this option when PDSCH is signalled by control resource set 0 (CORESET 0). For this case, the BWP parameters must align with CORSET 0.
- `CRB0` — When the reference point is subcarrier 0 of the common resource block 0 (CRB 0)

Data Types: `char` | `string`

`DMRSTypeAPosition` — Position of first DM-RS OFDM symbol
`2` (default) | `3`

Position of first DM-RS OFDM symbol, provided by higher layer parameter *dmrs-TypeA-Position*, specified as `2` or `3`.

This property only applies when the `MappingType` property of the `nrPDSCHConfig` or `nrWavegenPDSCHConfig` objects is set to `'A'`.

Data Types: `double`

`DMRSAdditionalPosition` — Maximum number of DM-RS additional positions
`0` (default) | `1` | `2` | `3`

Maximum number of DM-RS additional positions, specified as `0`, `1`, `2`, or `3`. This property is the higher layer parameter *dmrs-AdditionalPosition*.

Data Types: `double`

`DMRSLength` — Number of consecutive front-loaded DM-RS OFDM symbols
`1` (default) | `2`

Number of consecutive front-loaded DM-RS OFDM symbols, specified as `1` (single-symbol DM-RS) or `2` (double-symbol DM-RS).

Data Types: `double`

`CustomSymbolSet` — DM-RS OFDM symbol locations
`[]` (default) | integer from 0 to 13 | vector of nonnegative integers

DM-RS OFDM symbol locations that are 0-based, specified as one of these options.

- Integer from 0 to 13 — For one DM-RS symbol
- Vector of nonnegative integers from 0 to 13 — For multiple DM-RS symbols

Each input symbol location is assumed to be a single-symbol DM-RS within the physical shared channel symbol allocation.

The default value, `[]`, corresponds to the DM-RS symbol locations, as defined in TS 38.211 Table 7.4.1.1.2-3 or 7.4.1.1.2-4. Setting this property overrides the corresponding DM-RS symbol locations in these standard lookup tables.

Data Types: `double`

`DMRSPortSet` — DM-RS antenna ports
`[]` (default) | integer scalar | vector of nonnegative integers

DM-RS antenna ports, specified as one of these options.

- Integer from 0 to 11 — For a single antenna port
- Vector of nonnegative integers from 0 to 11 — For multiple antenna ports

Nominal antenna ports supported depend on `DMRSLength` and `DMRSConfigurationType` property values.

`DMRSLength` Value	`DMRSConfigurationType` Value	Nominal Range of Antenna Ports Supported
1	1	[0, 3]
	2	[0, 5]
2	1	[0, 7]
	2	[0, 11]

The default value, `[]`, implies that `DMRSPortSet` is in the range from 0 to `NumLayers`– 1, where `NumLayers` is a property of `nrPDSCHConfig` or `nrWavegenPDSCHConfig`.

Data Types: `double`

`NIDNSCID` — DM-RS scrambling identity
`[]` (default) | integer from 0 to 65,535

DM-RS scrambling identity, specified as one of these options.

- Integer from 0 to 65,535 — Use this option when the higher layer parameter *scramblingID0/ scramblingID1* is configured.
- `[]` — Use this option when *scramblingID0/scramblingID1* is not configured. In this case, the object sets the DM-RS scrambling identity to the physical layer cell identity, specified by the `NCellID` property of the carrier.

Data Types: `double`

`NSCID` — DM-RS scrambling initialization
`0` (default) | `1`

DM-RS scrambling initialization, specified as `0` or `1`.

Data Types: `double`

`NumCDMGroupsWithoutData` — Number of DM-RS CDM groups without data
2 (default) | 1 | 3

Number of DM-RS CDM groups without data, specified as 1, 2, or 3.

Each value indicates a different set of CDM group numbers, according to TS 38.214 Section 5.1.6.2.

- 1 — CDM group number 0
- 2 — CDM group numbers 0 and 1
- 3 — CDM group numbers 0, 1, and 2

Data Types: `double`

`DMRSPower` — Power scaling of PSCCH DM-RS in dB
0 (default) | real scalar

Power scaling of the PDSCH DM-RS in dB, specified as a real scalar. Use this property to scale the power of the PDSCH DM-RS in the generated 5G waveform. This scaling is additional to the PDSCH-wide power scaling specified by the `Power` property.

Data Types: `double`

`EnablePTRS` — Enable PT-RS
0 (default) | 1

Enable PT-RS, specified as one of these values.

- 0 — Disable PT-RS configuration.
- 1 — Enable PT-RS configuration.

Data Types: `double` | `logical`

`PTRS` — PDSCH PT-RS configuration parameters
default `nrPDSCHPTRSConfig` object (default) | `nrPDSCHPTRSConfig` object

PDSCH PT-RS configuration, specified as an `nrPDSCHPTRSConfig` object. This property relates to the phase tracking reference signal configuration and contains all properties of the specified `nrPDSCHPTRSConfig` object.

`TimeDensity` — PT-RS time density
1 (default) | 2 | 4

PT-RS time density, specified as 1, 2 or 4. This property is the higher layer parameter *timeDensity*.

Data Types: `double`

`FrequencyDensity` — PT-RS frequency density
2 (default) | 4

PT-RS frequency density, specified as 2 or 4. This property is the higher layer parameter *frequencyDensity*.

Data Types: `double`

REOffset — Resource element offset

'00' (default) | '01' | '10' | '11'

Resource element offset with a specific subcarrier offset, specified as '00', '01','10', or '11'. This property is the higher layer parameter *resourceElementOffset*.

Data Types: char | string

PTRSPortSet — PT-RS antenna port set

[] (default) | nonnegative integer

PT-RS antenna port set, specified as a nonnegative integer. Specify [] to set this property to the lowest value in the DMRSPortSet property of nrPDSCHDMRSConfig object. This usage of [] value is applicable only when nrPDSCHPTRSConfig object is used as a property of nrPDSCHConfig or nrWavegenPDSCHConfig.

Data Types: double

Dependencies

To enable this property, set the EnablePTRS property to 1.

PTRSPower — Power scaling of PSCCH PT-RS in dB

0 (default) | real scalar

Power scaling of the PDSCH PT-RS in dB, specified as a real scalar. Use this property to scale the power of the PDSCH PT-RS in the generated 5G waveform. This scaling is additional to the PDSCH-wide power scaling specified by the Power property.

Dependencies

To enable this property, set the EnablePTRS property to 1.

Data Types: double

Examples

Configure PDSCH for 5G Downlink Waveform Generation

Create a PDSCH configuration object for 5G waveform generation with the specified property values.

```
pdsch = nrWavegenPDSCHConfig( ...
    'BandwidthPartID',0, ...
    'Modulation','16QAM', ...
    'TargetCodeRate',658/1024, ...
    'SymbolAllocation',[0 7], ...
    'SlotAllocation',[0 2], ...
    'Period',3, ...
    'PRBSet',[0:20], ...
    'EnablePTRS',true);
```

Create a downlink carrier configuration object, specifying the previously defined PDSCH configuration.

```
cfg = nrDLCarrierConfig('PDSCH',{pdsch});
```

Configure Multiple PDSCH for 5G Downlink Waveform Generation

Create two SCS carrier configuration objects with mixed numerologies.

```
carrier1 = nrSCSCarrierConfig('SubcarrierSpacing',15);
carrier2 = nrSCSCarrierConfig('SubcarrierSpacing',30);
```

Create two BWP configuration objects, one for each of the SCS carriers.

```
bwp1 = nrWavegenBWPConfig('BandwidthPartID',0,'SubcarrierSpacing',15);
bwp2 = nrWavegenBWPConfig('BandwidthPartID',1,'SubcarrierSpacing',30);
```

Create two PDSCH configuration objects for 5G waveform generation, specifying a unique UE and one of the BWP configurations for each PDSCH.

```
pdsch1 = nrWavegenPDSCHConfig('RNTI',1,'BandwidthPartID',0,'Modulation','QPSK');
pdsch2 = nrWavegenPDSCHConfig('RNTI',2,'BandwidthPartID',1,'Modulation','16QAM');
```

Create a downlink carrier configuration object, specifying the previously defined configurations.

```
cfg = nrDLCarrierConfig( ...
    'SCSCarriers',{carrier1,carrier2}, ...
    'BandwidthParts',{bwp1,bwp2}, ...;
    'PDSCH',{pdsch1,pdsch2});
```

References

[1] 3GPP TS 38.211. “NR; Physical channels and modulation.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig` | `nrPDSCHConfig` | `nrWavegenBWPConfig`

Introduced in R2020b

nrWavegenSSBurstConfig

SS burst configuration parameters for 5G waveform generation

Description

The `nrWavegenSSBurstConfig` object sets synchronization signal (SS) burst configuration parameters. Use this object to set the `SSBurst` property of the `nrDLCarrierConfig` object when configuring 5G downlink waveform generation.

This object defines the subcarrier spacing (SCS), time-domain and frequency-domain allocations, power, and payload of the SS burst.

Creation

Syntax

```
ssb = nrWavegenSSBurstConfig
ssb = nrWavegenSSBurstConfig(Name,Value)
```

Description

`ssb = nrWavegenSSBurstConfig` creates a default SS burst configuration object for 5G waveform generation.

`ssb = nrWavegenSSBurstConfig(Name,Value)` sets properties on page 3-142 using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'BlockPattern','Case B'` specifies block pattern Case B.

Properties

`Enable` — Enable SS burst

`1` (default) | `0`

Enable SS burst in 5G waveform generation, specified as one of these values.

- `1` — Enable SS burst.
- `0` — Disable SS burst.

Data Types: `double` | `logical`

`Power` — Power scaling of SS burst in dB

`0` (default) | real number

Power scaling of the SS burst in dB, specified as a real number. Use this property to scale the power of the SS burst in the generated 5G waveform.

Data Types: `double`

`BlockPattern` — Block pattern of SS burst

`'Case A'` (default) | `'Case B'` | `'Case C'` | `'Case D'` | `'Case E'`

Block pattern of the SS burst, specified as one of these values corresponding to the patterns from TS 38.213 Section 4.1 [1].

- `'Case A'` — Use this value for frequency range 1 (FR1) and 15 kHz SCS.
- `'Case B'` or `'Case C'` — Use either of these values for FR1 and 30 kHz SCS.
- `'Case D'` — Use this value for frequency range 2 (FR2) and 120 kHz SCS.
- `'Case E'` — Use this value for FR2 and 240 kHz SCS.

Data Types: `char` | `string`

`TransmittedBlocks` — Block transmission bitmap

`[1 1 1 1]` (default) | 4-bit, 8-bit, or 64-bit binary vector

Block transmission bitmap in a 5 ms half-frame burst, specified as a 4-bit or 8-bit binary vector for FR1 or a 64-bit binary vector for FR2. This vector specifies which SS blocks are active in the SS burst.

Data Types: `double`

`Period` — Period of SS burst in ms

`20` (default) | `5` | `10` | `40` | `80` | `160`

Period of the SS burst in ms, specified as `5`, `10`, `20`, `40`, `80`, or `160`.

Data Types: `double`

`KSSB` — Subcarrier offset

`0` (default) | integer from 0 to 23

Subcarrier offset, specified as one these options.

- For FR1, specify this property as an integer from 0 to 23. For block pattern Case B, `KSSB` must be even. Units are in terms of 15 kHz SCS.
- For FR2, specify this property as an integer from 0 to 11. For block pattern Case D, `KSSB` must be even. For block pattern Case E, `KSSB` must be a multiple of 4. Units are in terms of SCS equal to the `SubcarrierSpacingCommon` property value.

The object increases the frequency offset of the SS burst from point A by `KSSB` subcarriers.

Dependencies

To enable this property, set the `NCRBSSB` property to a value other than `[]`.

Data Types: `double`

`NCRBSSB` — Frequency offset from point A

`[]` (default) | integer from 0 to 2199

Frequency offset from point A, specified as `[]` or an integer from 0 to 2199. Point A is the center of subcarrier 0 in the common resource block 0 (CRB 0). This property specifies the frequency offset of the SS burst in resource blocks (RBs) relative to point A.

- For block pattern Case A, Case B and Case C, the unit of this property is expressed in terms of 15 kHz SCS.
- For block pattern Case D and Case E, the unit of this property is expressed in terms of 60 kHz SCS.

When `NCRBSSB` is an empty vector, `[]`, the SS burst is positioned in the center of the carrier with the SCS corresponding to the block pattern specified by the `BlockPattern` property.

Data Types: `double`

`DataSource` — Source of SS burst payload

`'MIB'` (default) | `'PN9-ITU'` | `'PN9'` | `'PN11''PN15'` | `'PN23'` | two-element cell array | 24-bit binary vector

Source of the SS burst payload, specified as one of these options.

- `'MIB'`, `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, or `'PN23'`
- Two-element cell array consisting of one of the character vectors from the previous list and a random numeric seed (for example, `{'PN9',7}`)
- 24-bit binary vector

If you do not specify a random seed, all shift registers are initialized with an active state.

Data Types: `double` | `cell` | `string` | `char`

`DMRSTypeAPosition` — Position of first DM-RS symbol

`2` (default) | `3`

Position of the first demodulation reference signal (DM-RS) symbol in the physical downlink shared channel (PDSCH) system information block type 1 (SIB1), specified as `2` or `3`.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double`

`CellBarred` — Cell barring

`0` (default) | `1`

Cell barring, specified as `0` or `1`. When `CellBarred` is set to `1`, the cell enables the user equipment (UE) to camp on the cell.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double` | `logical`

`IntraFreqReselection` — Enable intrafrequency reselection

`0` (default) | `1`

Enable intrafrequency reselection, specified as one of these values.

- `0` — Disable intrafrequency reselection.
- `1` — Enable intrafrequency reselection of the same frequency cells.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double` | `logical`

`PDCCHConfigSIB1` — Configuration type of PDCCH SIB1

`0` (default) | integer from 0 to 255

Configuration type of the physical downlink control channel (PDCCH) SIB1, specified as an integer from 0 to 255.

Dependencies

To enable this property, set the `DataSource` property to `'MIB'`.

Data Types: `double`

`SubcarrierSpacingCommon` — SIB1 SCS in kHz

`15` (default) | `30` | `60` | `120`

SIB1 SCS in kHz, specified as one of these values.

- `15` or `30` for FR1
- `60` or `120` for FR2

Dependencies

To enable this property, either set the `DataSource` property to `'MIB'` or set the `NCRBSSB` property to a value other than `[]` and the `BlockPattern` property to `'Case D'` or `'Case E'`.

Data Types: `double`

Examples

Configure SS Burst for 5G Downlink Waveform Generation

Create a downlink carrier configuration object with the specified property values.

```
cfgDL = nrDLCarrierConfig('FrequencyRange','FR2','ChannelBandwidth',100);
cfgDL.SCSCarriers{1} = nrSCSCarrierConfig('SubcarrierSpacing',120);
cfgDL.BandwidthParts{1} = nrWavegenBWPConfig('SubcarrierSpacing',120);
```

Create an SS burst configuration object for block pattern Case D, corresponding to 120 kHz SCS, and a block transmission bitmap for FR2.

```
ssb = nrWavegenSSBurstConfig('BlockPattern','Case D','TransmittedBlocks',ones(1,64));
```

Specify the SS burst frequency offset to be one third of the carrier size.

```
ssb.NCRBSSB = round(cfgDL.SCSCarriers{1}.NSizeGrid/3);
```

Specify the subcarrier offset of the SS burst, taking into account the specified block pattern. For block pattern Case D, the subcarrier offset value must be even.

```
ssb.KSSB = 2*4;
```

Specify the SS burst configuration for the downlink carrier configuration.

```
cfgDL.SSBurst = ssb;
```

References

[1] 3GPP TS 38.213. “NR; Physical layer procedures for control.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrDLCarrierConfig`

Introduced in R2020b

pcapWriter

PCAP file writer of protocol packets

Description

The `pcapWriter` object writes generated and recovered protocol packets to a packet capture (PCAP) file (`.pcap`).

You can write these packet types to a PCAP file:

- Generated and recovered 5G NR protocol packets
- Generated and recovered WLAN protocol packets (requires WLAN Toolbox)
- Generated and recovered Bluetooth® low energy (BLE) link layer (LL) packets (requires Communications Toolbox™ Library for the Bluetooth Protocol)

Creation

Syntax

```
pcapObj = pcapWriter
pcapObj = pcapWriter(Name,Value)
```

Description

`pcapObj = pcapWriter` creates a default PCAP file writer object.

`pcapObj = pcapWriter(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapWriter` object does not overwrite the existing PCAP file. During each call of this object, specify a unique PCAP file name.

`FileName` — Name of the PCAP file

`'capture'` (default) | character row vector | string scalar

Name of the PCAP file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order

`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeGlobalHeader	Write global header to PCAP file

Examples

Write 5G NR Packet to PCAP File

Create a PCAP file writer object, specifying the name of the PCAP file. 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
pcapObj = pcapWriter('FileName','sample');
linkType = 113;                                          % Link type of SLL packet
timestamp = 300;                                         % Timestamp
```

Write a global header to the PCAP file.

```
writeGlobalHeader(pcapObj,linkType);
```

The 5G New Radio (NR) packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                     % MAC subPDU (contains truncated buffer s
radioType = 1;                                           % Frequency division duplexing
linkDir = 0;                                             % Uplink packet
rntiType = 3;                                            % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];               % Tag to indicate the start of NR MAC sig
payloadTag = 1;                                          % Payload tag for NR packets
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a user datagram protocol (UDP) header.

```
udpPacketLength = 8 + length(macNRInfoPacket);           % Length of header (8 bytes) and payload
udpHeader = [163; 76;                                    % Source port number
    39; 15;                                              % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256); % Total length of UDP packet
    0; 0];                                               % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                   % Length of header (20 bytes) and payloa
ipHeader = [69;                                          % Version of IP protocol and priority or
    0;                                                   % Type of service
    fix(ipPacketLength/256); mod(ipPacketLength,256);    % Total length of the IPv4 packet
    0; 1;                                                % Identification
    0; 0;                                                % Flags and fragmentation offset
    64;                                                  % Time to live in seconds
    17;                                                  % UDP protocol number
    0; 0;                                                % Header checksum
```

```
    127; 0; 0; 1;                                   % Source IP address
    127; 0; 0; 1];                                  % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0; 0;                                  % Packet type
    3; 4;                                           % Address resolution protocol hardware (A
    0; 0;                                           % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                         % Link layer address
    8; 0];                                          % Protocol type
```

Construct 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet to the PCAP file.

```
write(pcapObj,packet,timestamp);
```

References

[1] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[2] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org/.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`pcapngWriter`

Introduced in R2020b

pcapngWriter

PCAPNG file writer of protocol packets

Description

The `pcapngWriter` object writes generated and recovered protocol packets to a packet capture next generation (PCAPNG) file (`.pcapng`).

You can write these packet types to a PCAPNG file:

- Generated and recovered 5G NR protocol packets
- Generated and recovered WLAN protocol packets (requires WLAN Toolbox)
- Generated and recovered Bluetooth low energy (BLE) link layer (LL) packets (requires Communications Toolbox Library for the Bluetooth Protocol)

Creation

Syntax

```
pcapngObj = pcapngWriter
pcapngObj = pcapngWriter(Name,Value)
```

Description

`pcapngObj = pcapngWriter` creates a default PCAPNG file writer object.

`pcapngObj = pcapngWriter(Name,Value)` sets properties using one or more name-value pair arguments. Enclose each property name in quotes. For example, `'ByteOrder','big-endian'` specifies the byte order as big-endian.

Properties

Note The `pcapngWriter` object does not overwrite the existing PCAPNG file. During each call of this object, specify a unique PCAPNG file name.

`FileName` — Name of the PCAPNG file

`'capture'` (default) | character row vector | string scalar

Name of the PCAPNG file, specified as a character row vector or a string scalar.

Data Types: `char` | `string`

`ByteOrder` — Byte order

`'little-endian'` (default) | `'big-endian'`

Byte order, specified as `'little-endian'` or `'big-endian'`.

Data Types: `char` | `string`

`FileComment` — Comment for PCAPNG file
`''` (default) | character vector | string scalar

Comment for the PCAPNG file, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Object Functions

Specific to This Object

write	Write protocol packet data to PCAP or PCAPNG file
writeCustomBlock	Write custom block to PCAPNG file
writeInterfaceDescriptionBlock	Write interface description block to PCAPNG file

Examples

Write 5G NR Packet to PCAPNG File

Create a PCAPNG file writer object, specifying the name of the PCAPNG file.

```
pcapngObj = pcapngWriter('FileName','sample');
```

Write the interface block for 5G New Radio (NR). 5G NR packets do not have a valid link type. As per Tcpdump, if a valid link type is not present, specify the link type of SLL packet.

```
interface = '5GNR';                                          % Interface name
linkType = 113;                                              % Link type of SLL packet
timestamp = 300;                                             % Timestamp
interfaceID = writeInterfaceDescriptionBlock(pcapngObj,linkType,interface);
```

5G NR packets are not directly supported by Wireshark. To enable Wireshark to parse 5G NR packets, add encapsulation and metadata to the 5G NR packet.

```
payload = [59; 205];                                         % MAC subPDU (contains truncated buffe
radioType = 1;                                               % Frequency division duplexing
linkDir   = 0;                                               % Uplink packet
rntiType = 3;                                                % Cell-RNTI
startString = [109; 97; 99; 45; 110; 114];                   % Tag to indicate start of NR MAC signa
payloadTag = 1;                                              % Payload tag for NR MAC packets
signature = [startString; radioType; linkDir; rntiType];
macNRInfoPacket = [signature; payloadTag; payload];
```

Construct a user datagram protocol (UDP) header.

```
udpPacketLength = 8 + length(macNRInfoPacket);               % Length of header (8 bytes) and payloa
udpHeader = [163; 76;                                        % Source port number
    39; 15;                                                  % Destination port number
    fix(udpPacketLength/256); mod(udpPacketLength,256);      % Total length of UDP packet
    0; 0];                                                   % Checksum
```

Construct an IPv4 header.

```
ipPacketLength = 20 + udpPacketLength;                  % Length of header (20 bytes) and payl
ipHeader = [69;                                         % Version of IP protocol and priority
    0;                                                  % Type of service
    fix(ipPacketLength/256);mod(ipPacketLength,256);    % Total length of the IPv4 packet
    0; 1;                                               % Identification
    0; 0;                                               % Flags and fragmentation offset
    64;                                                 % Time to live in seconds
    17;                                                 % Protocol number
    0; 0;                                               % Header checksum
    127; 0; 0; 1;                                       % Source IP address
    127; 0; 0; 1];                                      % Destination IP address
```

Construct an SLL header.

```
sllHeader = [0; 0;                                      % Packet type
    3; 4;                                               % Address resolution protocol hardware
    0; 0;                                               % Link layer address length
    0; 0; 0; 0; 0; 0; 0; 0;                             % Link layer address
    8; 0];                                              % Protocol type
```

Construct an 5G NR packet by adding encapsulation and metadata.

```
packet = [sllHeader; ipHeader; udpHeader; macNRInfoPacket];
```

Write the 5G NR packet to the PCAPNG file.

```
packetComment = 'This is NR MAC packet';                % Packet comment
write(pcapngObj,packet,timestamp,interfaceID,'PacketComment',packetComment);
```

References

[1] Tuexen, M. “PCAP Next Generation (Pcapng) Capture File Format.” 2020. https://www.ietf.org/.

[2] Group, The Tcpdump. “Tcpdump/Libpcap Public Repository.” Accessed May 20, 2020. https://www.tcpdump.org.

[3] “Development/LibpcapFileFormat - The Wireshark Wiki.” Accessed May 20, 2020. https://www.wireshark.org/.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Objects

`pcapWriter`

Introduced in R2020b

Apps

5G Waveform Generator

Create, impair, visualize, and export 5G NR waveforms

Description

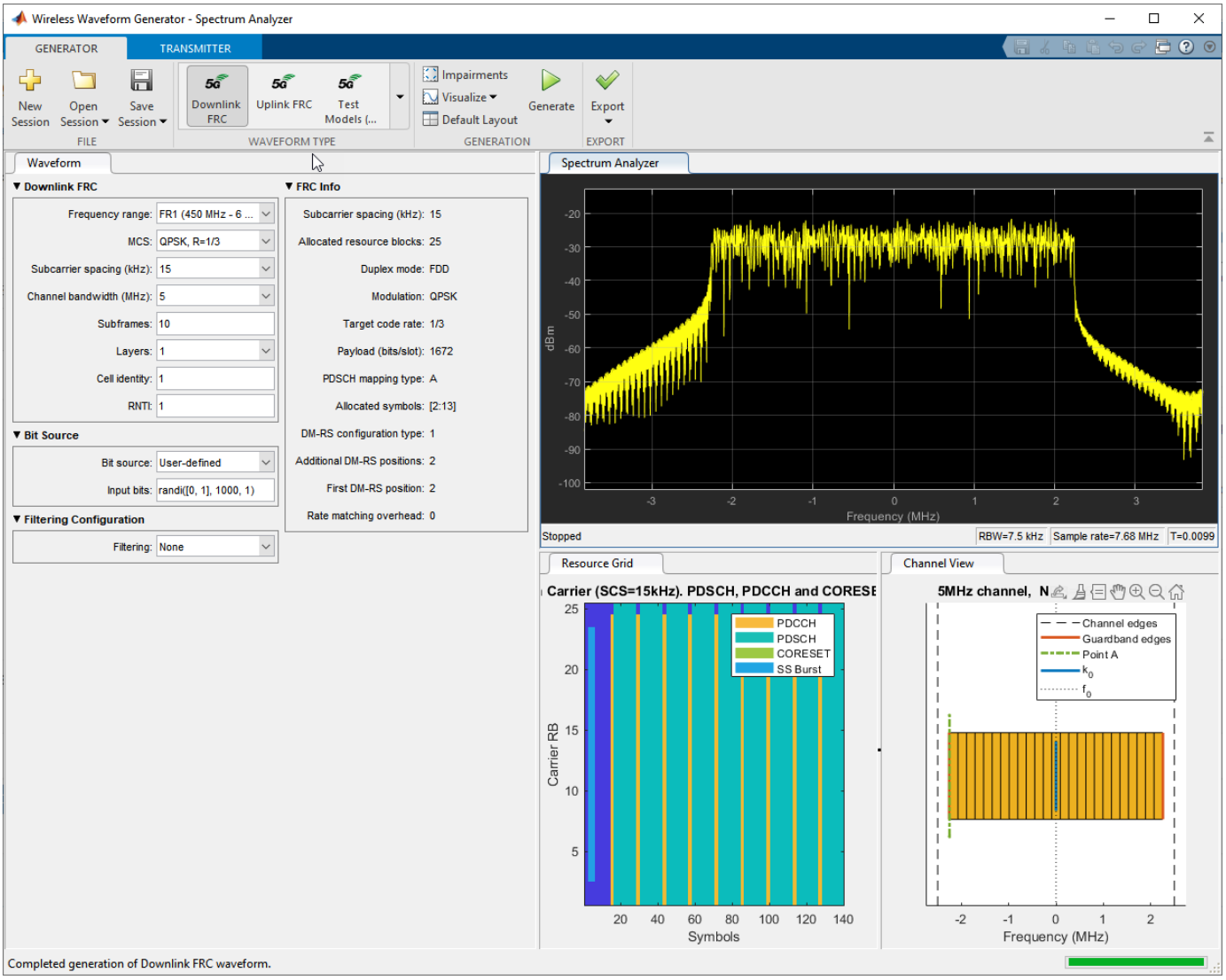
The **5G Waveform Generator** app enables you to create, impair, visualize, and export 5G NR waveforms.

The app provides these capabilities by using the **Wireless Waveform Generator** app configured for 5G NR waveform generation. Using the app, you can:

- Generate NR test models (NR-TM) for FR1, as defined in TS 38.141-1 Section 4.9.2 [1].
- Generate NR-TM for FR2, as defined in TS 38.141-2 Section 4.9.2 [2].
- Generate NR downlink fixed reference channel (FRC) waveforms, as defined in TS 38.101-1 Annex A.3 [3].
- Generate NR uplink FRC waveforms, as defined in TS 38.104 Annex A [4].
- Export the NR waveform to your workspace as a structure to a `.mat` or a `.bb` file.
- Distort the NR waveform by adding RF impairments, such as AWGN, phase offset, frequency offset, DC offset, IQ imbalance, and memoryless cubic nonlinearity.
- Visualize the NR waveform in spectrum analyzer and OFDM grid.
- Generate an NR waveform that you can transmit using a connected lab test instrument. The app can transmit a waveform by using instruments supported by the `rfsiggen` function. Use of the transmit feature in the app requires Instrument Control Toolbox™ software. For more information, see the documentation for “Instrument Control Toolbox”.

To create, impair, visualize, and export waveforms other than NR waveforms, you must reconfigure the app. For a full list of features, see the **Wireless Waveform Generator** app.

For more information, see “Using Wireless Waveform Generator App”.



Open the 5G Waveform Generator App

MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon. 

MATLAB Command Prompt: Enter `nrWaveformGenerator`. This command opens the **Wireless Waveform Generator** app configured for 5G waveform generation.

Examples

App-Based 5G NR-TM and FRC Waveform Generation

This example shows how to use the **5G Waveform Generator** app to generate NR test models (NR-TM) and NR uplink and downlink fixed reference channel (FRC) waveforms.

Open 5G Waveform Generator App

On the **Apps** tab of the MATLAB® toolstrip, under **Signal Processing and Communications**, click the **5G Waveform Generator** app icon. This app opens the **Wireless Waveform Generator** app configured for 5G waveform generation.

Select 5G NR Waveform

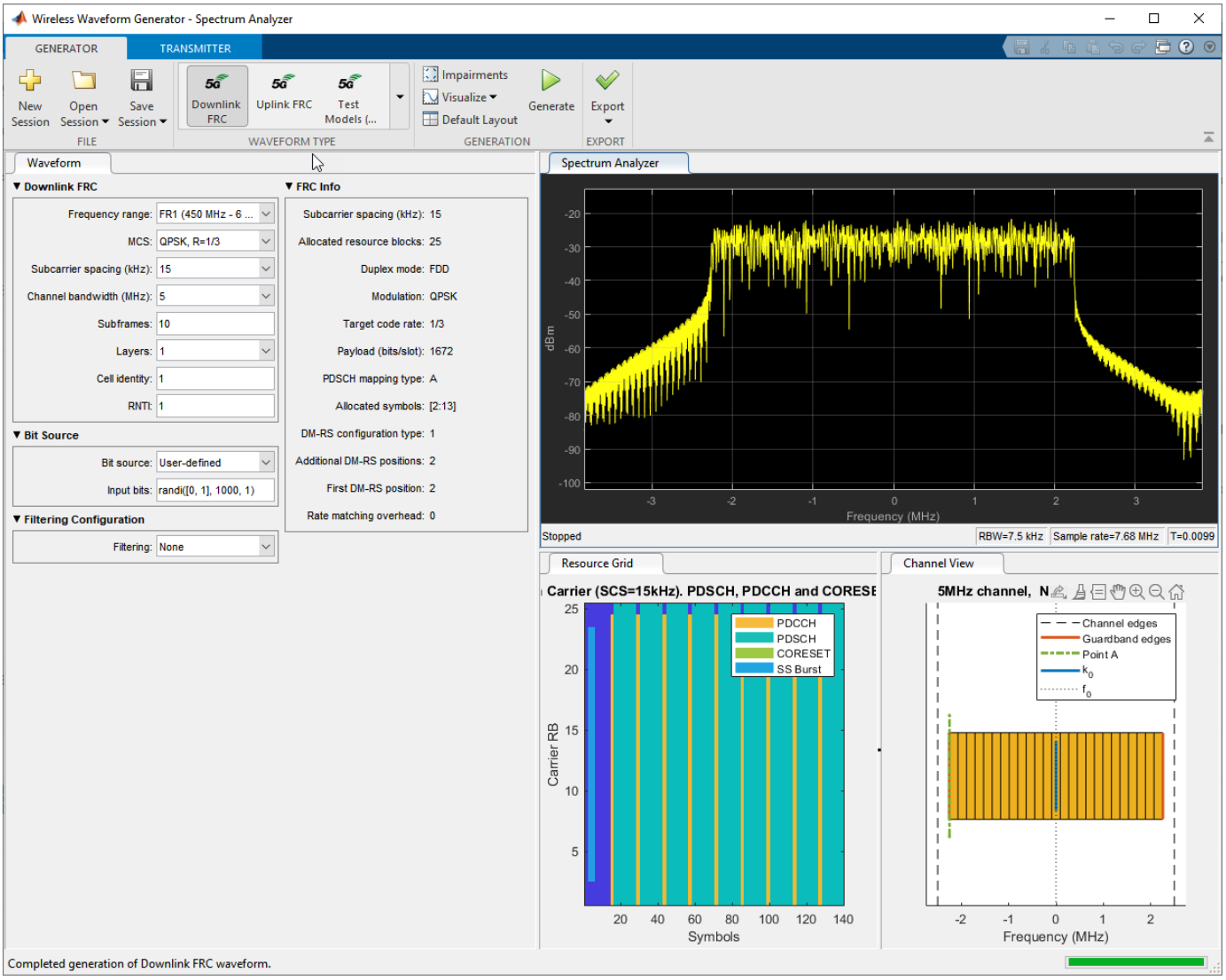
In the **Waveform Type** section on the app toolstrip, click the waveform you want to generate. You can select any of these waveforms.

- `5G Downlink FRC`
- `5G Uplink FRC`
- `5G Test Models`

Generate 5G NR Waveform

In the left pane of the app, on the **Waveform** tab, you can set the parameters of the selected waveform. On the app toolstrip, in the **Generation** section, you can add impairments and set visualization tools. To visualize the waveform, click **Generate**. You can export the generated waveform to the MATLAB workspace as a structure in a `.mat` or `.bb` file.

For example, this picture shows the visualization results of a downlink FRC waveform using default parameters.



Transmit 5G NR Waveform

This feature requires “Instrument Control Toolbox”. To transmit the generated waveform, on the app toolstrip, click on the **Transmitter** tab and set up the instruments. You can use all the instruments supported by the `rfsiggen` (Instrument Control Toolbox) function.

References

[1] 3GPP TS 38.141-1. “NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.141-2. “NR; Base Station (BS) conformance testing Part 2: Radiated conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.101-1. “NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Apps
Wireless Waveform Generator

Topics
“Using Wireless Waveform Generator App”

Introduced in R2020a