5G Toolbox™

Reference

MATLAB®

R2019b

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)

Contents

1

Functions — Alphabetical List

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 UT velocity of 15 km/h:

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

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

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

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

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

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

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

Transmit the input waveform through the channel.

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

Obtain the path filters used in channel filtering.

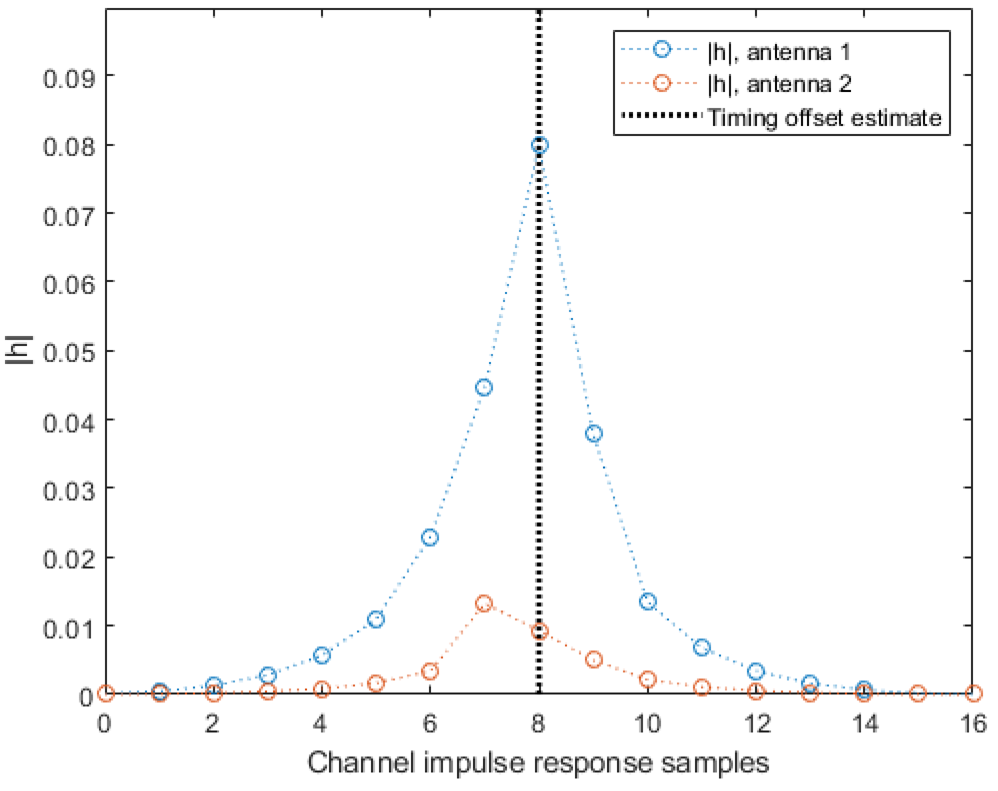
```
pathFilters = getPathFilters(cdl);
```

Perform timing offset estimation using `nrPerfectTimingEstmate`.

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

Plot the magnitude of the channel impulse response.

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



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

System 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 TR 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 TR 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`

System 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'.

Parameter Field	Value	Description
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.

Parameter Field	Value	Description
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).
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

System 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 23.
- 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`

Topics
“NR Synchronization Procedures”

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.

```
cdblk = 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
isequal(trblk,rxtrblk)
isequal(bi2de(rxSFN4lsb','left-msb'),mod(sfn,16))
[isequal(hrf,rxHRF) isequal(de2bi(floor(kssb/16),1),rxKssb)]
```

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`

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`

Topics

"NR Synchronization Procedures"

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( ___ ,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( ___ ,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-dependant 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.

```
nTxAnts = 1;
txGrid = complex(zeros([240 14 nTxAnts]));
txGrid(dmrsInd) = dmrsSym;
```

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

```
nFFT = 512;
cpLengths = ones(1,14) * 36;
cpLengths([1 8]) = 40;
nulls = [1:136 377:512].';
txWaveform = ofdmmod(txGrid,nFFT,cpLengths,nulls);
```

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

```
SR = 7.68e6;
channel = nrTDLChannel;
channel.NumReceiveAntennas = 1;
channel.SampleRate = SR;
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*SR)) + 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 spans 20 resource blocks at 15 kHz subcarrier spacing and uses initial slot number 0.

```
nrb = 20;
scs = 15;
initialSlot = 0;
offset = nrTimingEstimate(rxWaveform,nrb,scs,initialSlot,dmrsInd,dmrsSym);
```

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.

```
rxLength = sum(cpLengths) + nFFT*numel(cpLengths);
cpFraction = 0.55;
symOffsets = fix(cpLengths * cpFraction);
rxGrid = ofdmdemod(rxWaveform(1:rxLength,:),nFFT,cpLengths,symOffsets,nulls);
```

Obtain the practical channel estimate.

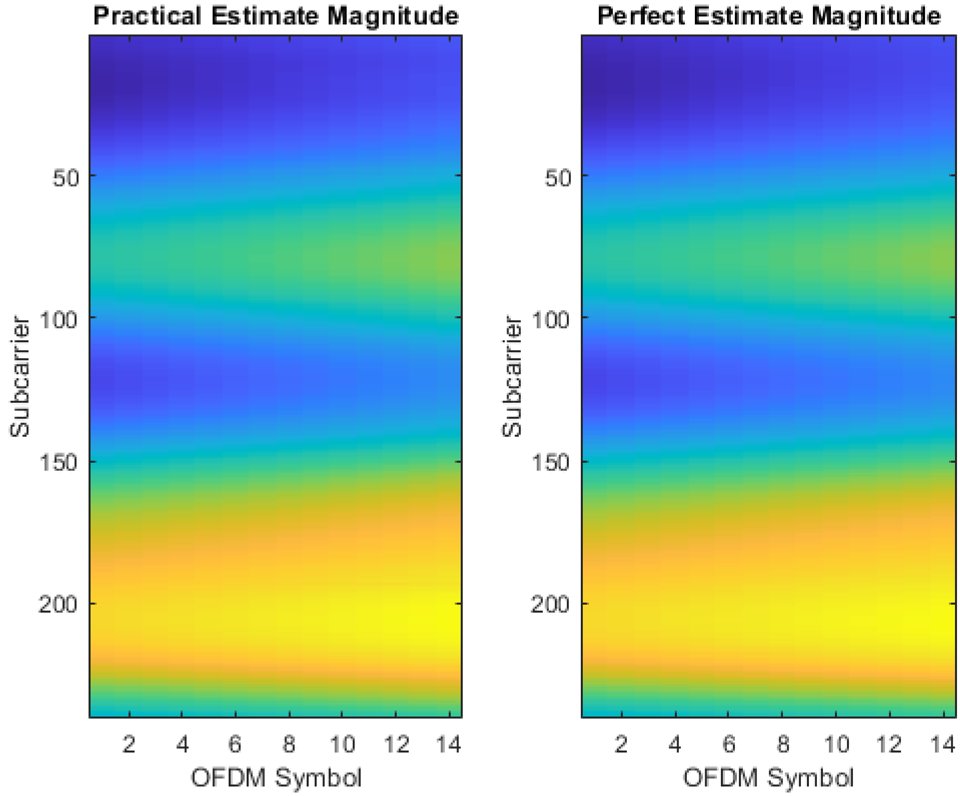
```
H = nrChannelEstimate(rxGrid,dmrsInd,dmrsSym);
```

Obtain the perfect channel estimate.

```
pathFilters = getPathFilters(channel);
H_ideal = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,initialSlot,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

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 values:

- `'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`

`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`

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. 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. Concatenate the code block segments using `nrCodeBlockDesegmentLDPC`. The concatenated result is of the same size as the original input with CRC and filler bits removed. Check whether the CRC decoding was successful by checking the error vector.

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

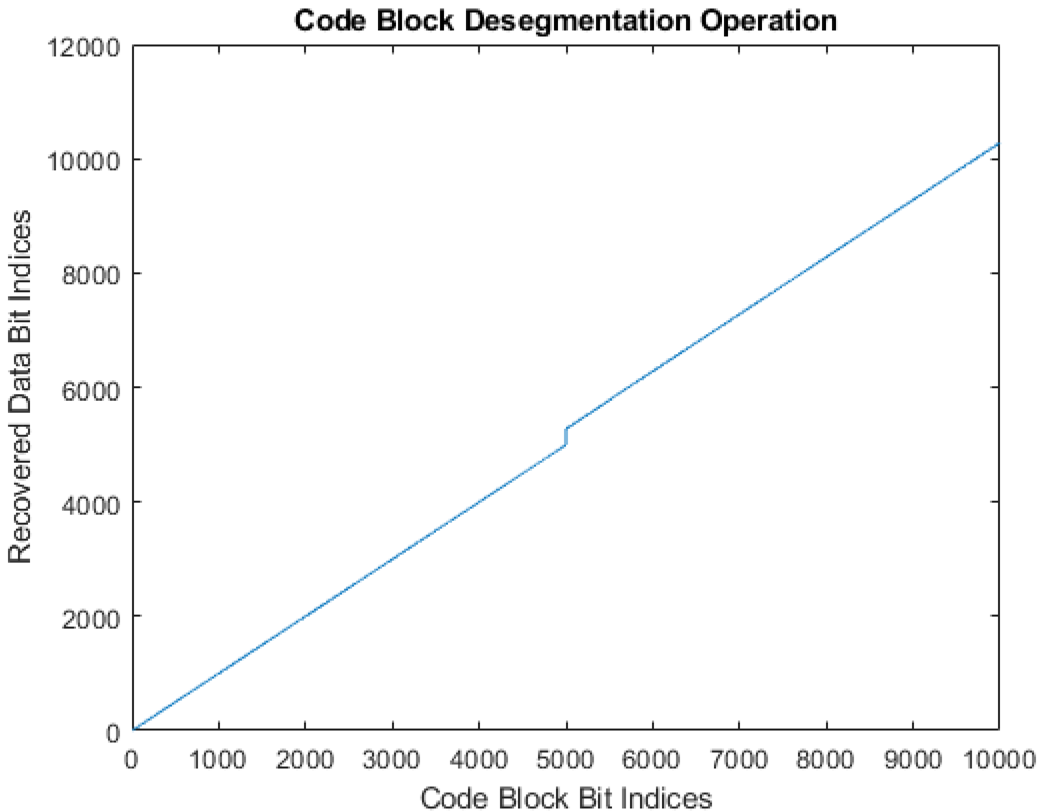
```
size(cbs)
[blk,err] = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen);
blkSize = size(blk)
err
```

```
ans =

    5280           2

blkSize =

    10000           1


err =

    1×2 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. Concatenate the code block segments using `nrCodeBlockDesegmentLDPC` with the specified base graph number and output block length. To see how 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.

```
cbs = reshape([1:10560]',[],2);
bgn = 1;
blklen = 10000;
blk = nrCodeBlockDesegmentLDPC(cbs,bgn,blklen);
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. Perform code block segmentation. 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.

```
in = randi([0,1],4000,1);
cbs1 = nrCodeBlockSegmentLDPC(in,1);
cbs2 = nrCodeBlockSegmentLDPC(in,2);
size(cbs1)
size(cbs2)
```

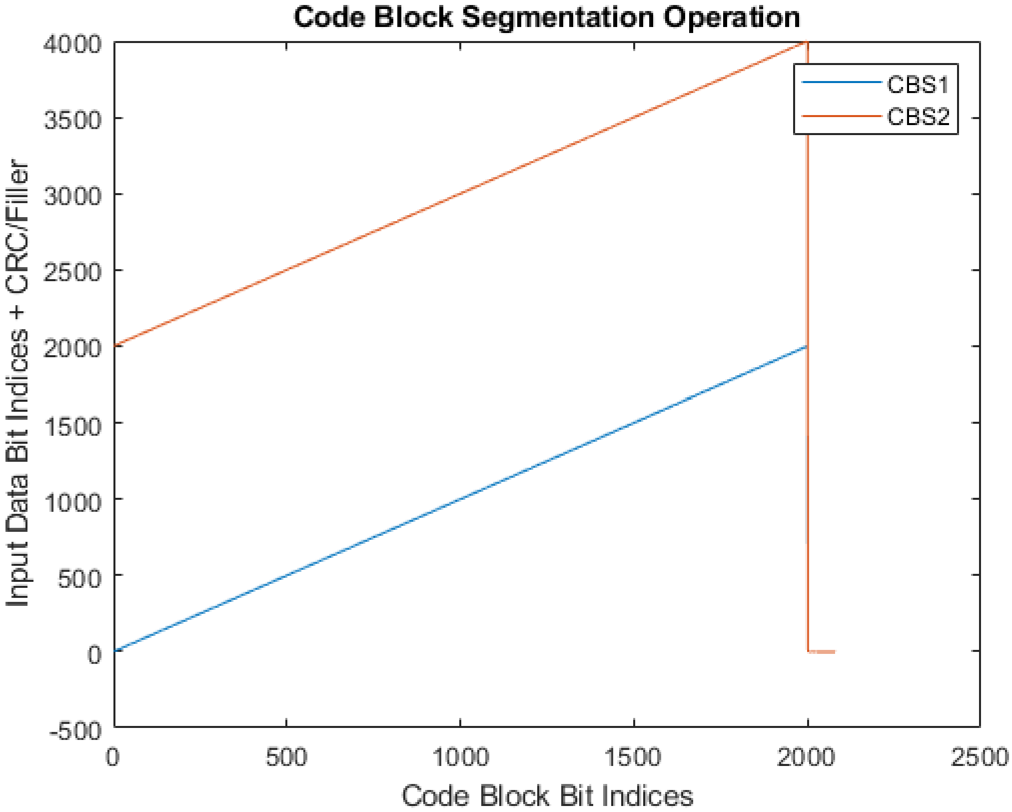
```
ans =
```

```
    4224           1


ans =

    2080           2
```

Display Index Mapping of LDPC Code Block Segmentation

Create a ramp data input and perform code block segmentation. 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.

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

     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
    {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
    {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.

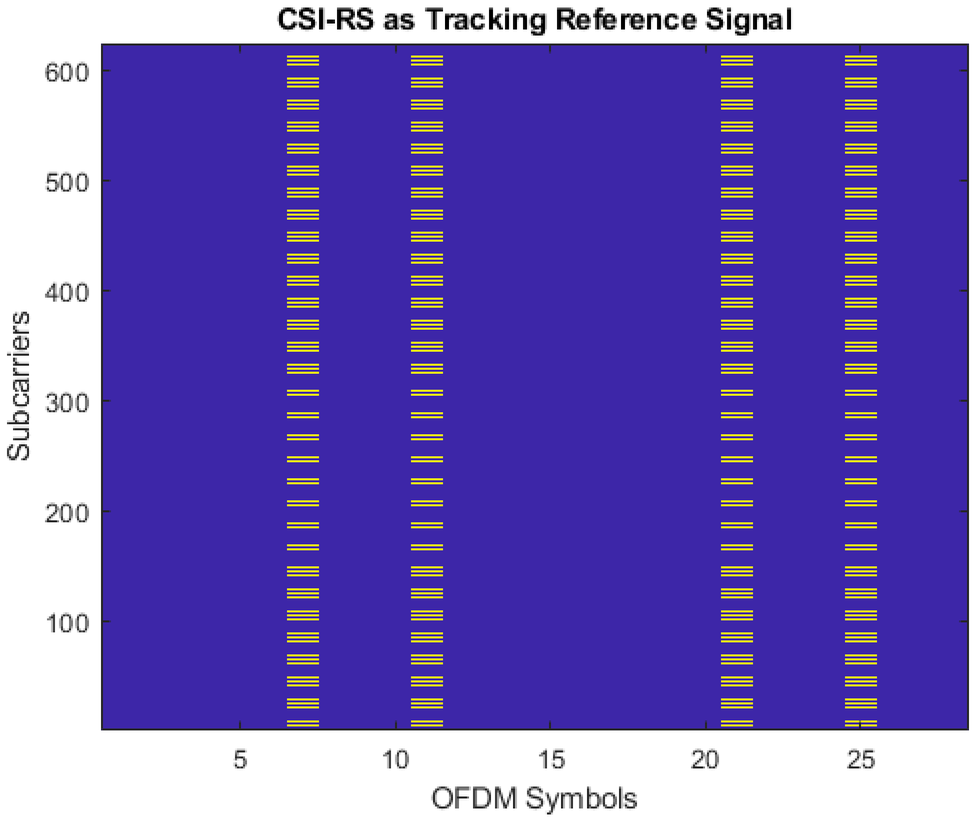
```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

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™.

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
    {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
    {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.

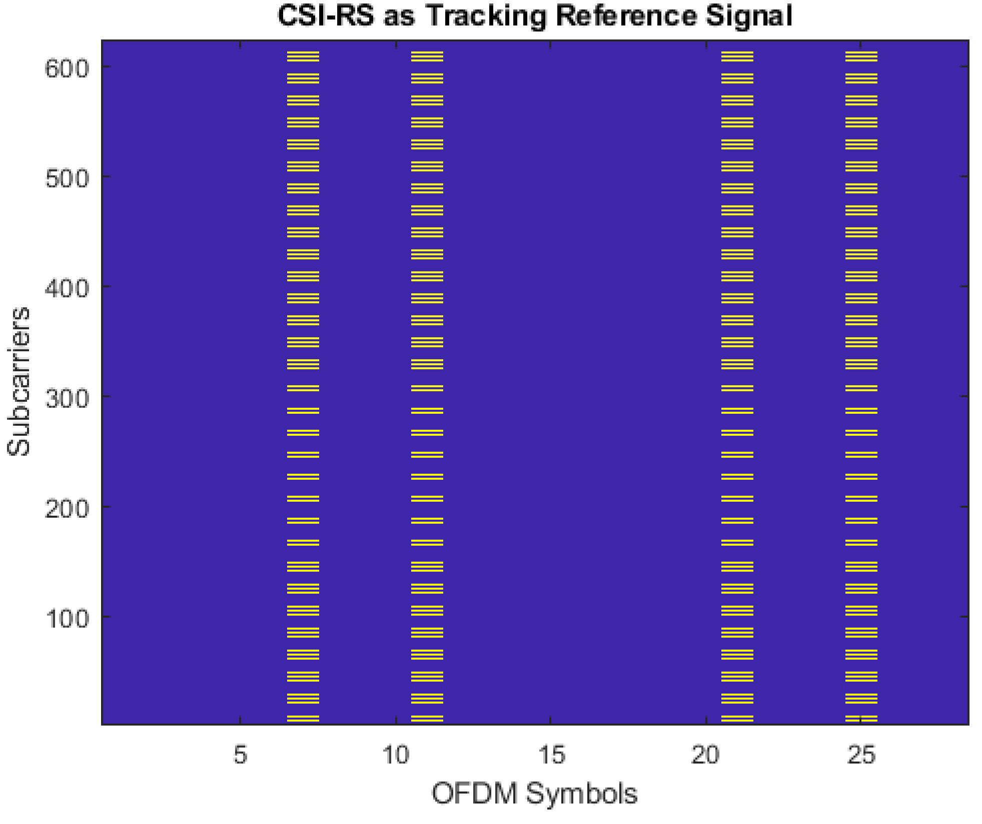
```
slotgrid1 = complex(zeros(gridSize));
slotgrid1(ind1) = sym1;
```

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™.

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)
```

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.

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`. Set the length of the polar decoding list to 8.

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

```
recBits = 32x1 int8 column vector

   1
   1
   0
   1
   1
   0
   0
   1
   1
   1
      ⋮

mask = uint32
    100
```

Verify that the transmitted and received message bits are identical.

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

```
ans = logical
   1
```

The recovered mask value is the RNTI value used for CRC masking.

```
mask
```

```
mask = uint32
    100
```

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`

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 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`

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 $K + 24 < \mathrm{E} \leq 8192$, 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

Fields	Values	Description
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

System 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.

```
P = 1;
txGrid = zeros([240 4 P]);
txGrid(pbchInd) = pbchTxSym;
```

Perform OFDM modulation.

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

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.

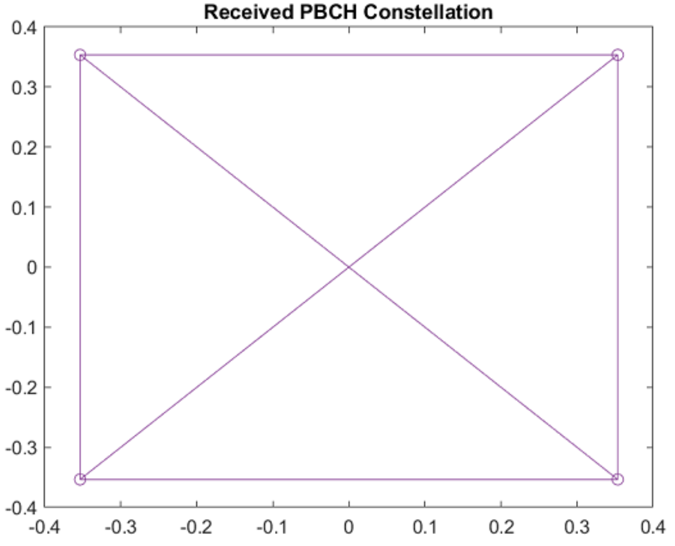
```
hEstGrid = repmat(permute(H.',[3 4 1 2]),[240 4]);
nEst = 0.1;
```

Perform OFDM demodulation.

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

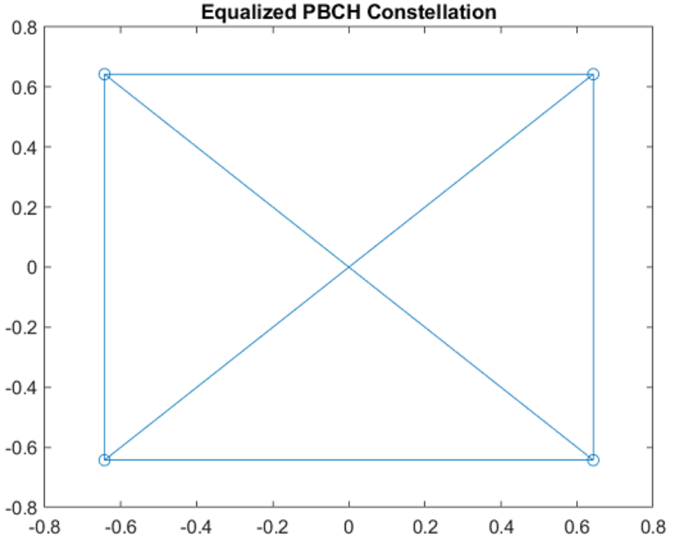
To prepare for PBCH decoding, use `nrExtractResources` to extract symbols from received and channel estimate grids. Plot the received PBCH constellation.

```
[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

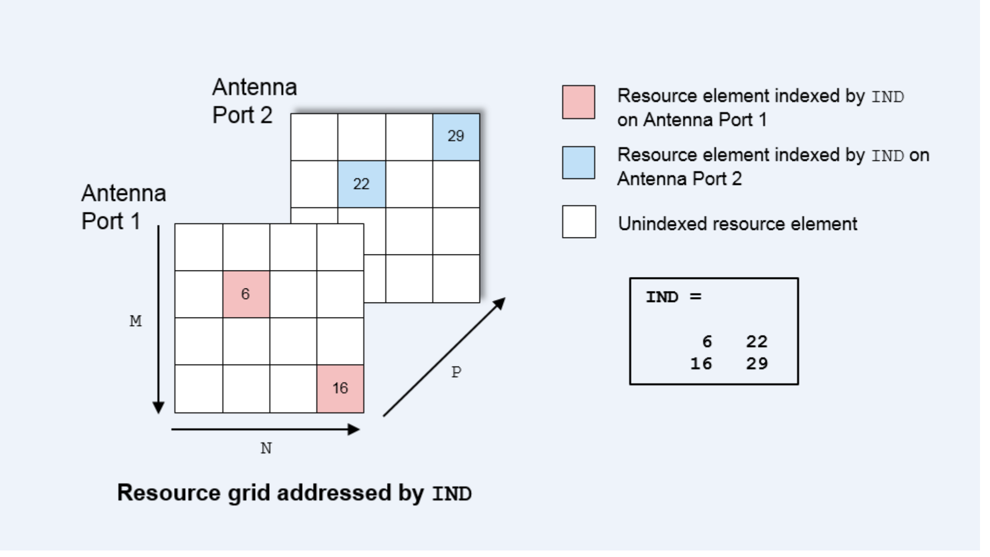
```
re = nrExtractResources(ind,grid)
[re,reind] = nrExtractResources(ind,grid)
[re1,...,reN,reind1,...,reindN] = nrExtractResources(
ind,grid1,grid2,....,gridN)
[ ___ ] = nrExtractResources( ___ ,Name,Value)
```

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.

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

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

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

PBCH Transmission and Decoding

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

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

Create channel estimates including beamforming.

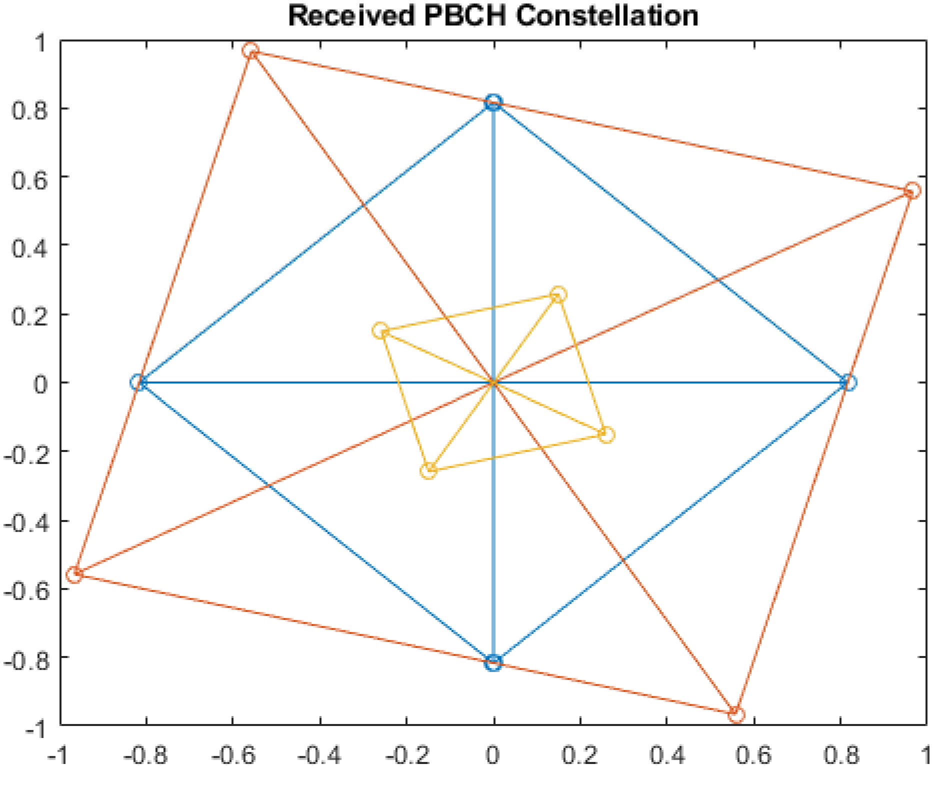
```
hEstGrid = repmat(permute(H.'*F.',[3 4 1 2]),[240 4]);
nEst = 0;
```

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

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

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

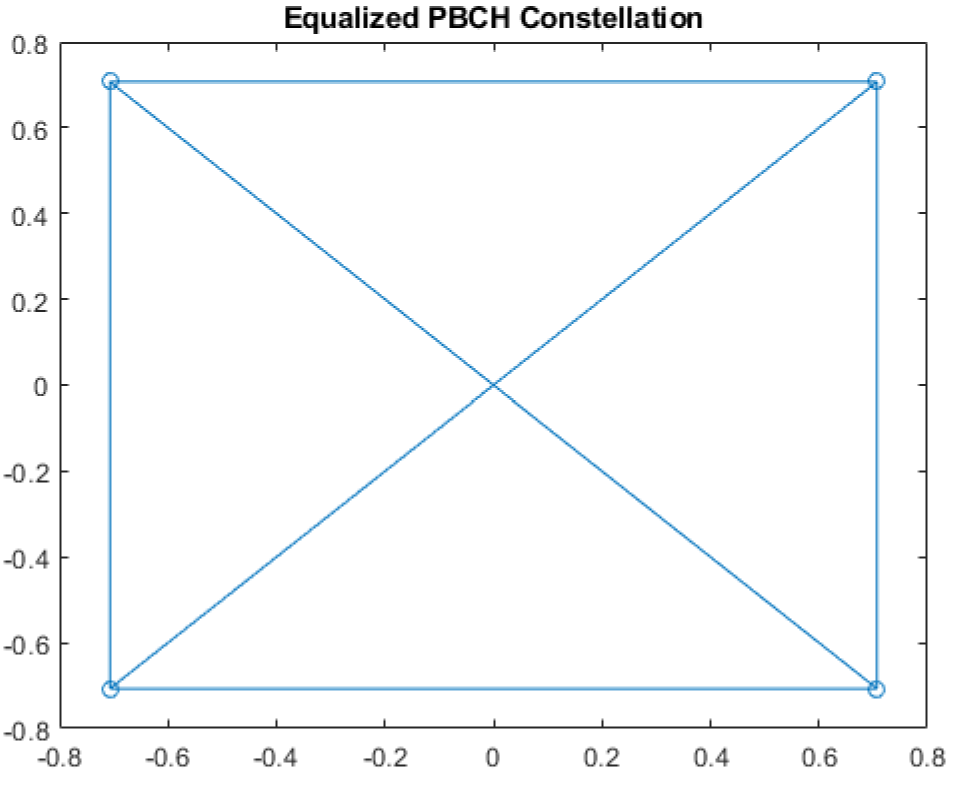
```
[pbchRxSym,pbchHestSym] = nrExtractResources(pbchInd,rxGrid,hEstGrid);
figure;
plot(pbchRxSym,'o:');
title('Received PBCH Constellation');
```



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

```
pbchEqSym = nrEqualizeMMSE(pbchRxSym,pbchHestSym,nEst);
figure;
```

```
plot(pbchEqSym,'o:');
title('Equalized PBCH Constellation');
```



Retrieve 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 1-92.

- `'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 1-94.

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 1-92.

reind — Indices of extracted resource elements

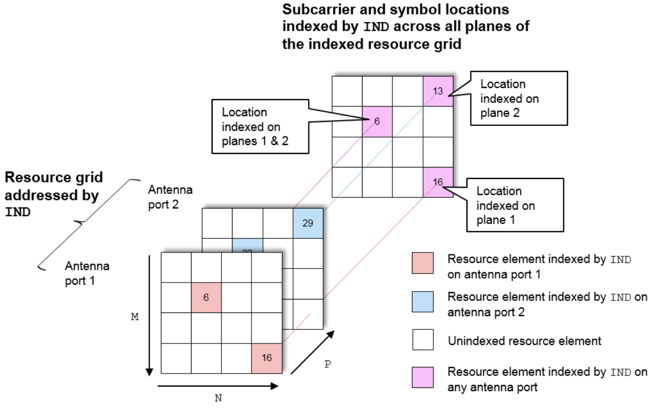
numeric array

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

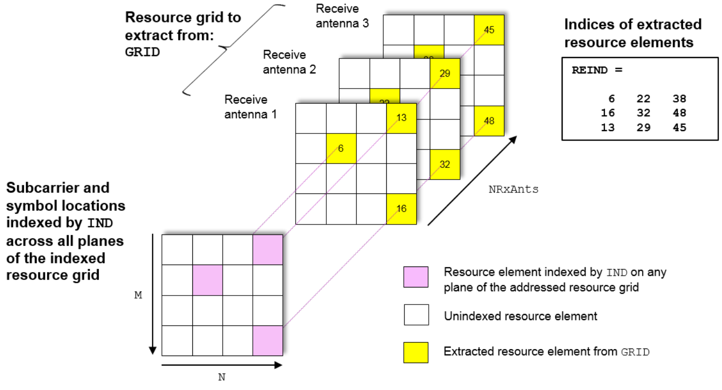
All-Planes Extraction Method (Default)

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 $P = 2$.



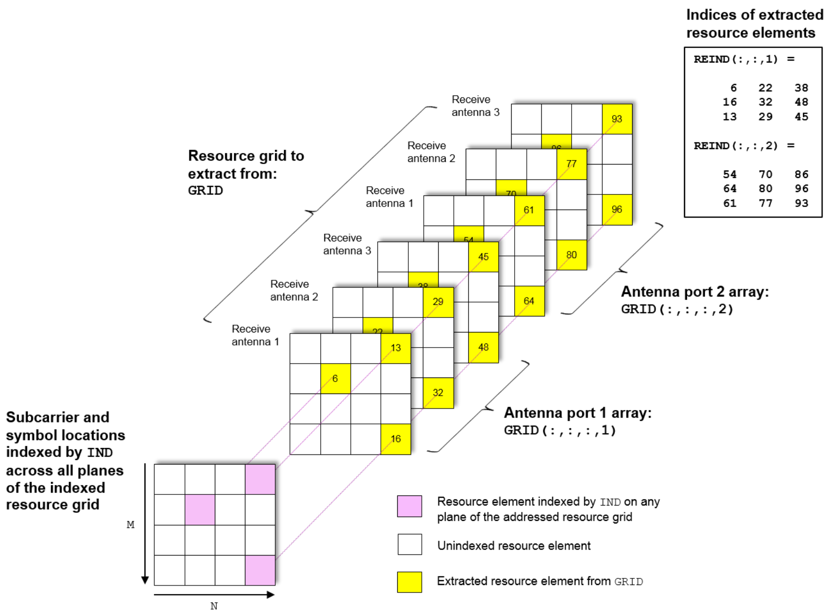
Extraction Process for a 3-D Received Grid

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.



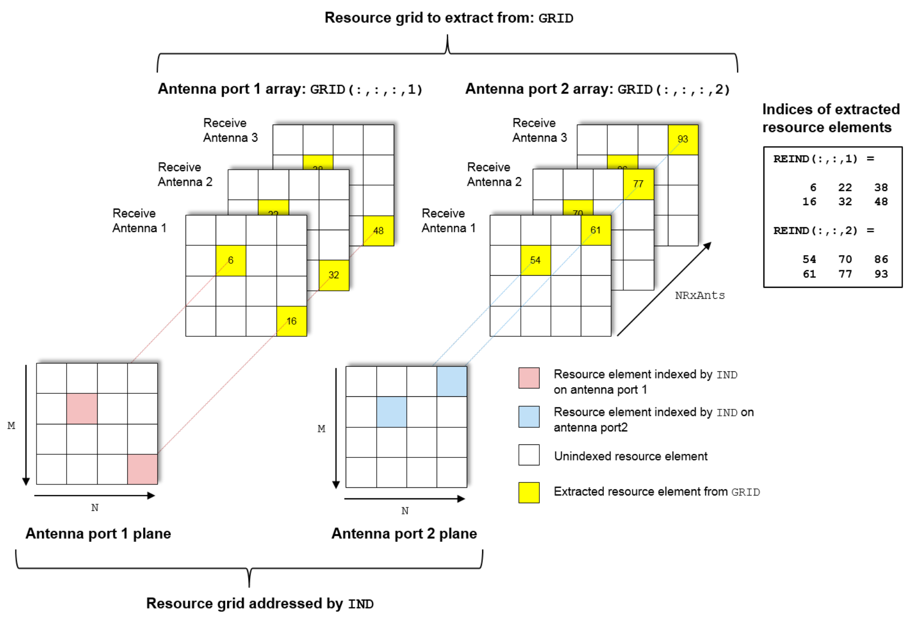
Direct Extraction Method

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 `codegen`. For more information, see `coder.Constant`.

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 onto four layers. Recover the original codeword using layer demapping. Check for errors.

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

Layer Mapping of One Codeword to Four Layers

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

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

```
sizeOut =

    10     4
```

Layer Mapping of Two Codewords to Five 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 =

    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
matrix

Rate recovered soft bits for input code block segments, specified as a 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
scalar

Maximum number of decoding iterations, specified as a 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 1-108.
- '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 1-110.
- '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 1-110.
- '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 1-111.

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

matrix

Decoded LDPC codeword or information bits, returned as a 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 $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid \text{channel output for } c_i)}{\Pr(c_i = 1 \mid \text{channel output for } c_i)}\right)$.

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

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

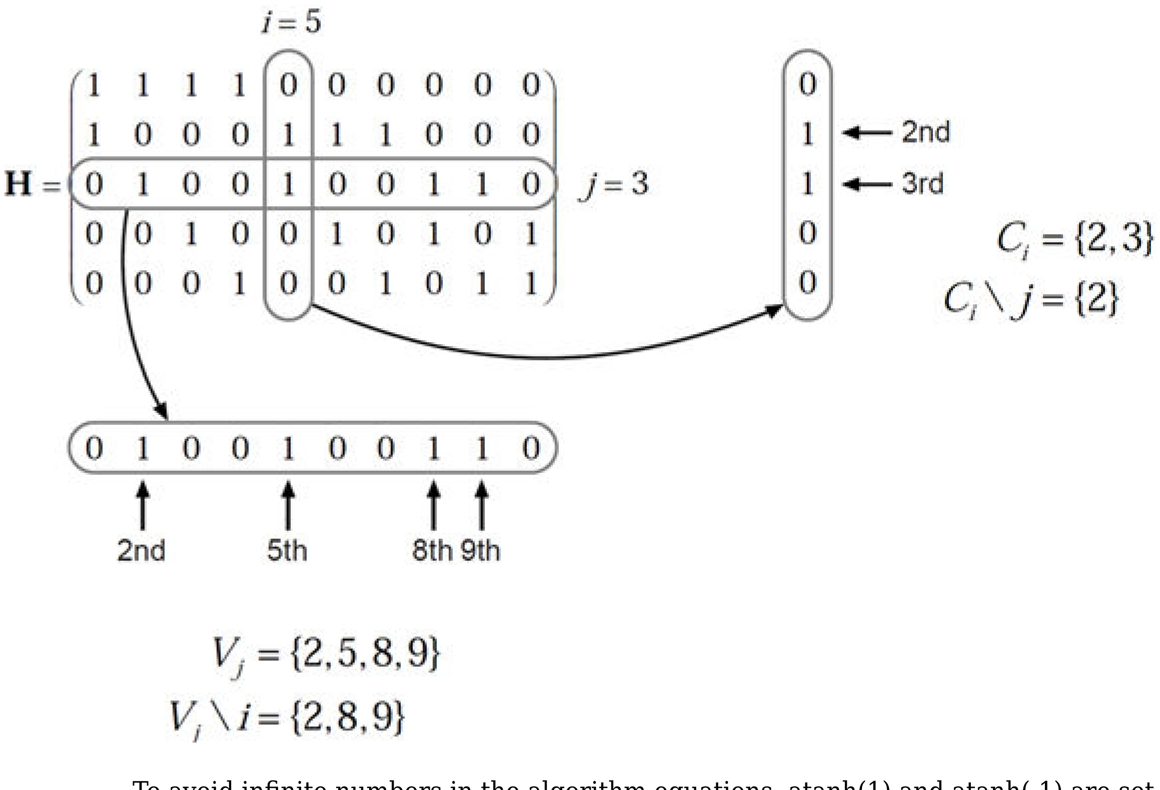
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

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

(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

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

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 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 codegen. For more information, see coder.Constant.

See Also

Functions
nrPRBS | nrPUCCH0 | nrPUCCH1

Introduced in R2019a

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

Generate the sequence of 432 PBCH quadrature phase shift keying (QPSK) modulation symbols. Consider the first Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst. Assume that the number of SS/PBCH blocks per half-frame is 4. To represent the encoded BCH bits, generate a random sequence of binary values. The length of the random sequence corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5.

```
ncellid = 17;
ssbindex = 0;
v = mod(ssbindex,4);
E = 864;
cw = randi([0 1],E,1);
```

```
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 `codegen`. For more information, see `coder.Constant`.

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

Generate the sequence of 432 PBCH quadrature phase shift keying (QPSK) modulation symbols. Consider the first Synchronization Signal / Physical Broadcast Channel (SS/PBCH) block in a burst. Assume that the number of SS/PBCH blocks per half-frame is 4. To represent the encoded BCH bits, generate a random sequence of binary values. The length of the random sequence corresponds to the PBCH bit capacity as specified in TS 38.212 Section 7.1.5.

```
ncellid = 17;
ssbindex = 0;
v = mod(ssbindex,4);
E = 864;
cw = randi([0 1],E,1);
```

```
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)
```

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 = $i_{\mathrm{SSB}} + 4 \times n_{\mathrm{hf}}$, 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 `codegen`. For more information, see `coder.Constant`.

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 =
```

```
144×1 uint32 column vector

242
246
250
254
258
...
```

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 `codegen`. For more information, see `coder.Constant`.

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);
```

```
indices =
```

```
432×1 uint32 column vector

241
243
244
245
247
248
...
```

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 `codegen`. For more information, see `coder.Constant`.

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); % assuming 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 `codegen`. For more information, see `coder.Constant`.

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 65,535

UE identifier, specified as an integer from 0 to 65,535.

- 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

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrDCIDecode` | `nrDCIEncode` | `nrPDCCHDecode` | `nrPDCCHPRBS`

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 65,535

UE identifier, specified as an integer from 0 to 65,535.

- 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`

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

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 65,535

UE identifier, specified as an integer from 0 to 65,535.

- 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`

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrPDCCH` | `nrPDCCHDecode` | `nrPRBS`

Introduced in R2018b

nrPDSCH

Generate PDSCH modulation symbols

Syntax

```
sym = nrPDSCH(cws,mod,nlayers,nid,rnti)
sym = nrPDSCH( ___ ,'OutputDataType',datatype)
```

Description

`sym = nrPDSCH(cws,mod,nlayers,nid,rnti)` returns 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( ___ ,'OutputDataType',datatype)` specifies the PDSCH symbol data type, in addition to the input arguments in the previous syntax.

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 -
   0.1543 + 0.4629i  -1.0801 + 1.0801i  -0.7715 + 0.7715i  -0.1543 + 0.7715i  -0.2301 -
  -0.1543 + 0.1543i   0.7715 - 1.0801i  -0.4629 + 0.7715i   0.1543 + 1.0801i   0.0767 -
  -0.7715 - 0.4629i  -0.1543 + 0.7715i  -0.7715 - 0.7715i  -0.4629 - 0.1543i  -0.6903 -
   1.0801 - 1.0801i  -1.0801 + 0.7715i   0.1543 - 0.4629i   0.4629 - 0.4629i  -1.1504 -
   0.4629 + 0.4629i   0.1543 + 0.1543i  -0.1543 + 0.1543i   0.1543 - 0.4629i   0.6903 -
  -1.0801 + 0.7715i   0.4629 - 1.0801i   0.4629 + 1.0801i  -0.4629 + 0.4629i  -0.6903 -
```

```
-1.0801 + 0.7715i  -0.1543 - 0.1543i   0.7715 + 1.0801i  -0.4629 - 0.1543i   0.8437 -
-0.4629 - 1.0801i  -0.7715 - 0.1543i   0.1543 - 1.0801i  -0.1543 + 0.1543i   0.2301 -
 0.7715 + 1.0801i   1.0801 - 0.4629i   1.0801 + 1.0801i  -0.1543 - 1.0801i  -0.0767 -
     ⋮
```

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`

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions

`nrDLSCHInfo` | `nrLayerMap` | `nrPDSCHDecode` | `nrPDSCHPRBS` | `nrSymbolModulate`

Introduced in R2018b

nrPDSCHDecode

Decode PDSCH modulation symbols

Syntax

```
[cws,symbols] = nrPDSCHDecode(sym,mod,nid,rnti)
[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( ___ ,nVar)` specifies the noise variance scaling factor of the soft bits in the PDSCH demodulation, in addition to the input arguments in the previous syntax.

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);
```

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

Modulation Scheme	Number of Bits Per Symbol
`'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`

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 | nrLayerDemap | nrPDSCH | nrPDSCHPRBS | nrSymbolDemodulate

Introduced in R2018b

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrPDSCH` | `nrPDSCHDecode` | `nrPRBS`

Introduced in R2018b

nrPerfectChannelEstimate

Perfect channel estimation

Syntax

```
h = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,
initialSlot)
h = nrPerfectChannelEstimate( ___ ,toffset)
h = nrPerfectChannelEstimate( ___ ,toffset,sampleTimes)
h = nrPerfectChannelEstimate( ___ ,cpl)
h = nrPerfectChannelEstimate( ___ ,'CyclicPrefix',cpl)
```

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( ___ ,toffset)` also specifies the timing offset in addition to the input arguments in the previous syntax. 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 previous syntax.

`h = nrPerfectChannelEstimate( ___ ,cpl)` or `h = nrPerfectChannelEstimate( ___ ,'CyclicPrefix',cpl)` also specifies the cyclic prefix length in addition to the input arguments in the previous syntax.

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.

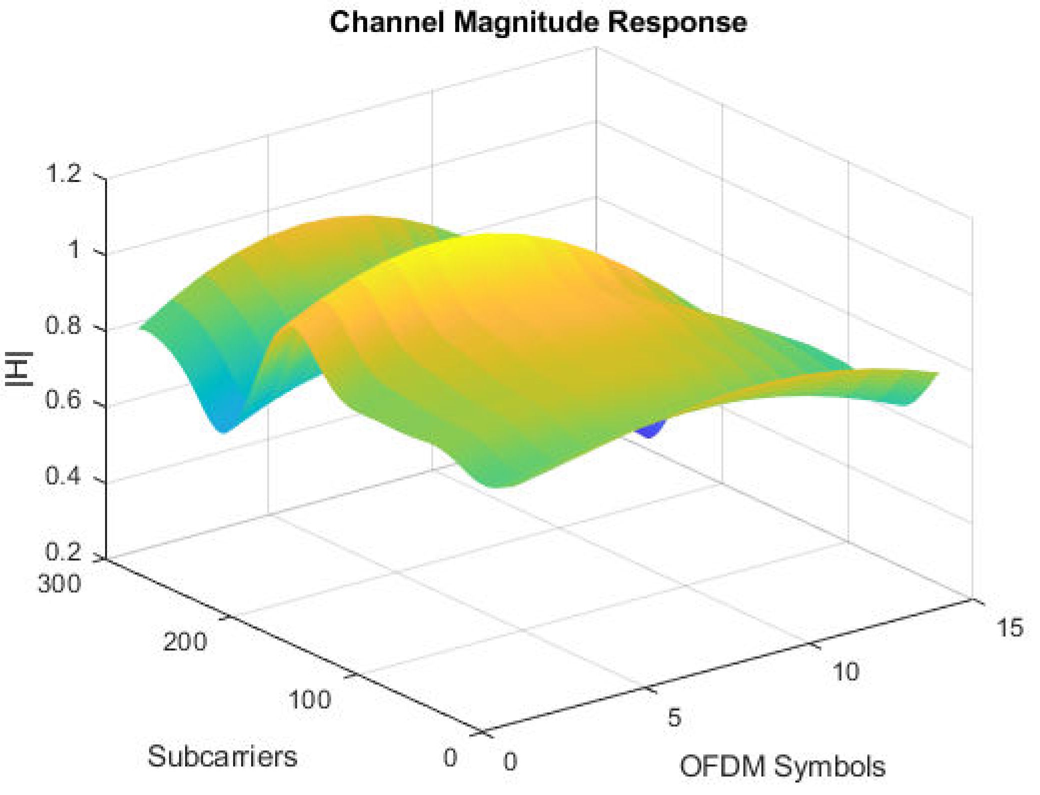
```
NRB = 25;
SCS = 15;
nSlot = 0;

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');
```



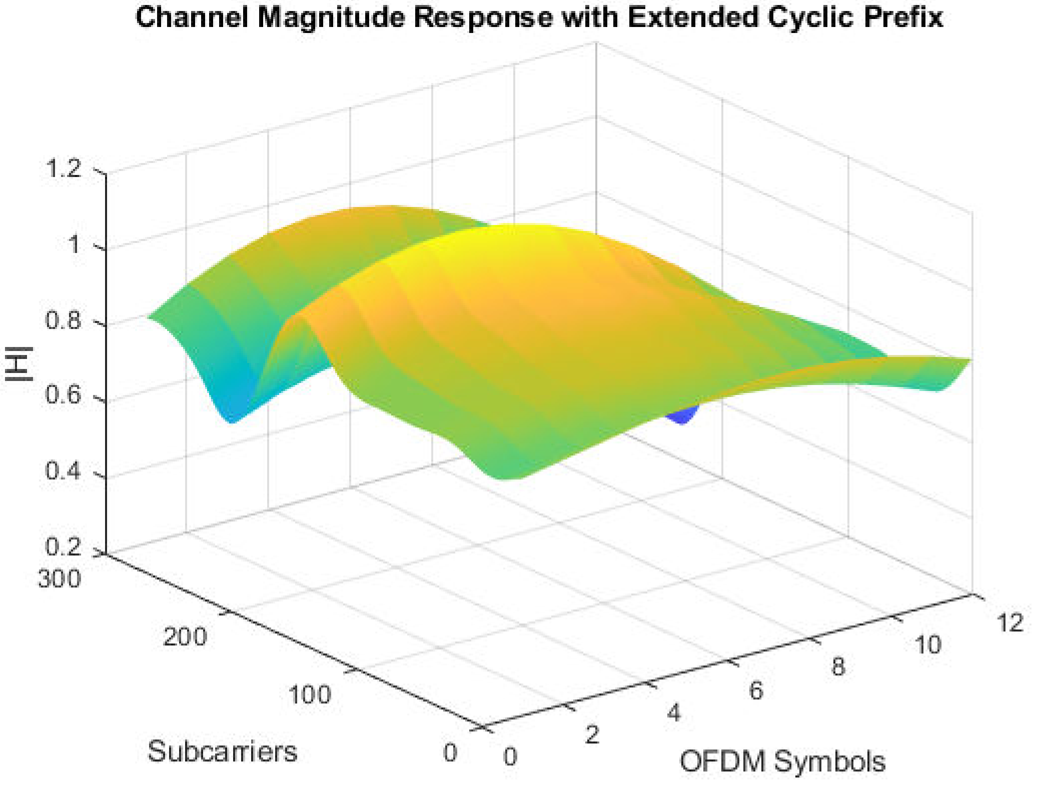
Repeat the channel estimate for extended cyclic prefix.

```
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);
```

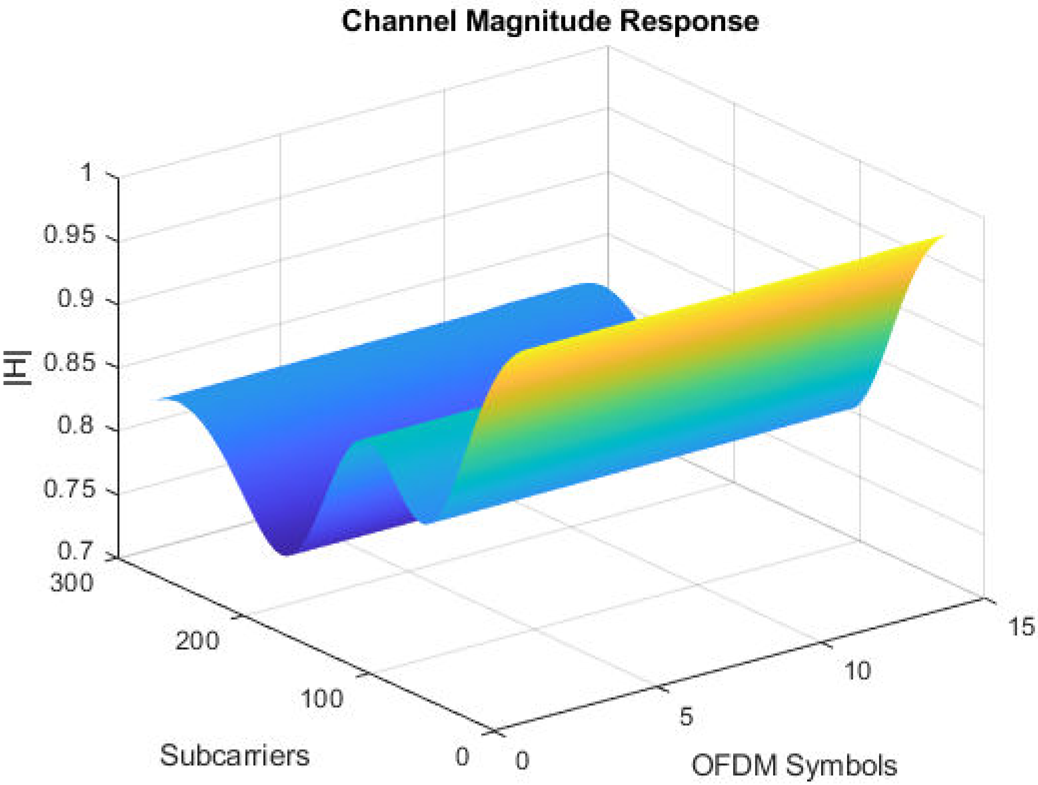
Perform perfect channel estimation. Use the specified number of blocks, subcarrier spacing, slot number, timing offset, and sample times.

```
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`

`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. Use `nrPerfectTimingEstimate` to derive `toffset`.

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. The number of channel snapshots, N_{CS}, is identical to the first dimension of `pathGains`.

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`

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™.

See Also

Functions
`nrChannelEstimate` | `nrPerfectTimingEstimate` | `nrTimingEstimate`

Objects
`nrCDLChannel` | `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 `toffset` 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.

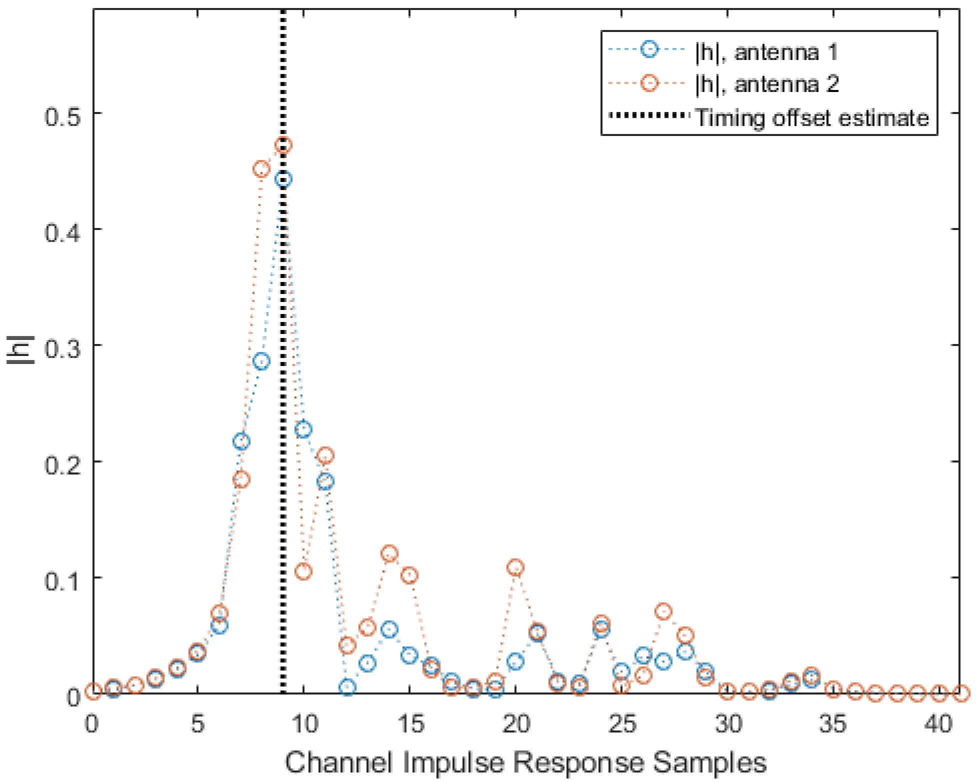
```
pathFilters = getPathFilters(tdl);
```

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,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,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.

- 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 ≤ `K` ≤ 25, `E` must be in the range `K` + 3 < `E` ≤ 8192.
- If `K` > 30, `E` must be in the range `K` < `E` ≤ 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`

`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, $N \leq 2^{nmax}$, 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`

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 ≤ 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 $N \leq 2^{nmax}$. 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, $N \leq 512$.
- For UL configuration, $N \leq 1024$.

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

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 1-201. 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 `codegen`. For more information, see `coder.Constant`.

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 =

    -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 `codegen`. For more information, see `coder.Constant`.

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 =

    127×1 uint32 column vector

    57
```

```
58
59
60
61
62
...
```

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 `codegen`. For more information, see `coder.Constant`.

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×*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 `codegen`. For more information, see `coder.Constant`.

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 `codegen`. For more information, see `coder.Constant`.

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 `codegen`. For more information, see `coder.Constant`.

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
`'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 `codegen`. For more information, see `coder.Constant`.

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 `codegen`. For more information, see `coder.Constant`.

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`.

Parameter Field	Values	Description
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 `codegen`. For more information, see `coder.Constant`.

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( ___ ,'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( ___ ,'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,t
```

```
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
      ⋮
```

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`

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrPUSCHCodebook` | `nrPUSCHDecode` | `nrPUSCHScramble` | `nrULSCHInfo`

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)
```

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`.

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,t
```

```
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,nl
```

```
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
```

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`

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`

Introduced in R2019a

nrPUSCHDescramble

Perform PUSCH descrambling

Syntax

```
cw = nrPUSCHDescramble(in,nid,rnti)
```

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). The function only performs data descrambling. Because uplink control information (UCI) placeholder bit locations are unknown, the function cannot correctly descramble UCIs if present in the input.

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
```

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`

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

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrPRBS` | `nrPUSCHDescramble` | `nrPUSCHScramble`

Introduced in R2019a

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.

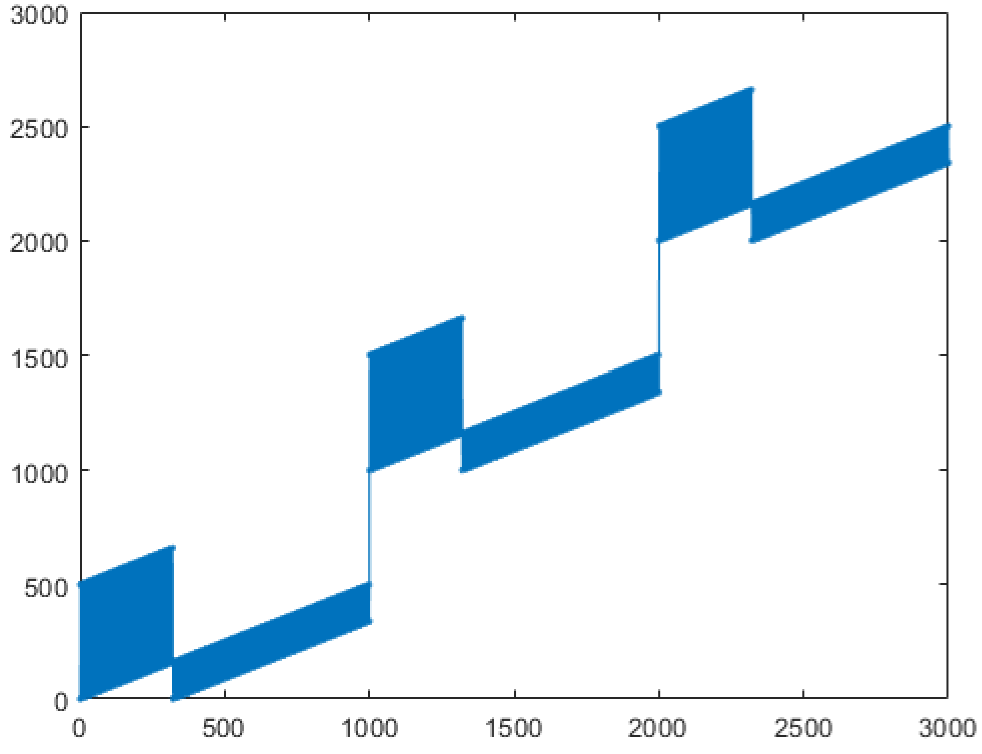
```
rv = 0;
mod = 'QPSK';
nLayers = 1;
outlen = 8000;
ratematched = nrRateMatchLDPC(encoded,outlen,rv,mod,nLayers);
size(ratematched)
```

```
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].

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'`.

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 with 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, $N \leq 512$.
- For UL configuration, $N \leq 1024$.

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 ≤ K ≤ 25, E must be in the range K + 3 < E ≤ 8192.
- If K > 30, E must be in the range K < E ≤ 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'.

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` ≤ 512 for DL configuration.
- `N` ≤ 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

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 =

    -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 `codegen`. For more information, see `coder.Constant`.

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 =

    127×1 uint32 column vector

    537
```

```
538
539
540
541
542
...
```

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 `codegen`. For more information, see `coder.Constant`.

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. Generate modulated symbols using QPSK modulation. Perform QPSK demodulation in soft decision mode for a noise variance of 0.1.

```
data = randi([0 1],40,1);
modsymb = nrSymbolModulate(data,'QPSK');
nVar = 0.1;
```

```
recsymb = awgn(modsymb,1/nVar,1,'linear');
out = nrSymbolDemodulate(recsymb,'QPSK',0.1);
```

16QAM Demodulation with Hard Decision Mode

Generate a random sequence of binary values of length 100. Generate modulated symbols using 16-QAM modulation. Add a noise to the modulated symbols corresponding to an SNR of 15 dB. Perform 16-QAM demodulation in hard decision mode. Check for bit errors.

```
data = randi([0 1],100,1,'int8');
modsymb = nrSymbolModulate(data,'16QAM');
recsymb = awgn(modsymb,15);
demodbits = nrSymbolDemodulate(recsymb,'16QAM','DecisionType','Hard');
numErr = biterr(data,demodbits)
```

```
numErr =

0
```

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 `codegen`. For more information, see `coder.Constant`.

See Also

Functions
`nrLayerMap` | `nrPBCH` | `nrPDCCH` | `nrPDSCH` | `nrPRBS` | `nrSymbolDemodulate`

Introduced in R2018b

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( ___ ,'CyclicPrefix',cpl)
```

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( ___ ,'CyclicPrefix',cpl)` also specifies the cyclic prefix length for the OFDM modulation in addition to the input arguments in any of 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.

```
txGrid = zeros([240 4]);
txGrid(pssInd) = pssSym;
```

OFDM modulate the resource grid.

```
txWaveform = ofdmmod(txGrid,512,[40 36 36 36],[1:136 377:512].');
```

Transmit the waveform through a TDL-C channel model by using a sample rate of 7.68 MHz.

```
SR = 7.68e6;
channel = nrTDLChannel;
channel.SampleRate = SR;
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 spans 20 resource blocks at 15-kHz subcarrier spacing and uses initial slot number 0.

```
nrb = 20;
scs = 15;
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 `cpl`.
- *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.
- *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`

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`

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`nrChannelEstimate` | `nrPerfectChannelEstimate` | `nrPerfectTimingEstimate`

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 1-329.

`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` ≥ 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 1-329.

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

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 1-334.

`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 ≥ *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

Modulation Scheme	Number of Bits Per Symbol
'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

A	Code Block Segmentation	CRC Bits	Encoding
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

nrULSCHInfo

Get uplink shared channel (UL-SCH) information

Syntax

`info = nrULSCHInfo(tBlkLen,targetCodeRate)`

Description

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

Examples

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
```

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 — 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

Fields	Values	Description
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

System Objects
`nrULSCH` | `nrULSCHDecoder`

Functions
`nrPUSCH` | `nrPUSCHDecode`

Introduced in R2019a

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

System 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 TR 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 TR 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`

System Objects
`nrDLSCH` | `nrULSCH`

Introduced in R2019a

2

System Objects — Alphabetical List

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? (MATLAB).

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 (MATLAB).

`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 — Desired scaled angle spreads in degrees

`[5.0 11.0 3.0 3.0]` (default) | four-element row vector

Desired scaled angle spreads in degrees, specified as a four-element row vector in one of these forms:

- [C_{ASD} C_{ASA} C_{ZSD} C_{ZSA}] row vector for scaling ray offset angles as described in TR 38.901 Section 7.7.1, Step1, where:

  - C_{ASD} is the cluster-wise azimuth spread of departure angles
  - C_{ASA} is the cluster-wise azimuth spread of arrival angles
  - C_{ZSD} is the cluster-wise zenith spread of departure angles
  - C_{ZSA} is the cluster-wise zenith spread of arrival angles

  To use this form, set `DelayProfile` to `'Custom'`.
- [ASD ASA ZSD ZSA] row vector for angle scaling, $AS_{desired}$, as described in TR 38.901 Section 7.7.5.1, where:

  - ASD is the azimuth spread of departure angles after scaling
  - ASA is the azimuth spread of arrival angles after scaling
  - ZSD is the zenith spread of departure angles after scaling
  - ZSA is the zenith spread of arrival angles after scaling

  To use this form, set `AngleScaling` to `true`.

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` — Desired mean angles in degrees

`[0.0 0.0 0.0 0.0]` (default) | four-element row vector

Desired mean angles in degrees, specified as a four-element row vector of the form [AoD AoA ZoD ZoA].

- AoD is the mean azimuth spread of departure angles after scaling
- AoA is the mean azimuth spread of arrival angles after scaling
- ZoD is the mean zenith spread of departure angles after scaling
- ZoA is the mean zenith spread of arrival angles after scaling

Use this vector for angle scaling as described in TR 38.901 Section 7.7.5.1

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 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 `'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 (UT) 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. • 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 antenna array elements are mapped panel-wise to the waveform channels (columns) 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, an antenna array of size `[4 8 2 2 2]` has the first $M = 4$ channels mapped to the first column of the first polarization angle of the first panel. The next $M = 4$ antennas are mapped to the next column, and so on. Following this pattern, the first $M \times N = 32$ channels are mapped to the first polarization angle of the complete first panel. Similarly, the remaining 32 channels are mapped to the second polarization angle of the first panel. Subsequent sets of $M \times N \times P = 64$ channels are mapped to consecutive panels, taking panel rows first, then panel columns.
`ElementSpacing`	`[0.5 0.5 1.0 1.0]` (default), row vector	Element spacing, in wavelengths, specified as a row vector of the form [λ_v λ_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 [θ ρ]. Polarization angles apply only when the number of polarizations is 2.

Parameter Field	Values	Description
Orientation	[0; 0; 0](default), column vector	Array orientation in degrees, specified as a column vector of the form [α; β; γ]. The vector elements specify the bearing, downtilt, and slant, respectively.
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. 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. • 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 antenna array elements are mapped panel-wise to the waveform channels (columns) 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, an antenna array of size `[4 8 2 2 2]` has the first $M = 4$ channels mapped to the first column of the first polarization angle of the first panel. The next $M = 4$ antennas are mapped to the next column, and so on. Following this pattern, the first $M \times N = 32$ channels are mapped to the first polarization angle of the complete first panel. Similarly, the remaining 32 channels are mapped to the second polarization angle of the first panel. Subsequent sets of $M \times N \times P = 64$ channels are mapped to consecutive panels, taking panel rows first, then panel columns.
`ElementSpacing`	`[0.5 0.5 0.5 0.5]` (default), row vector	Element spacing in wavelengths, specified as a row vector of the form [λ_v λ_h dg_v dg_h]. The vector values 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 [θ ρ]. Polarization angles apply only when the number of polarizations is 2.

Parameter Field	Values	Description
Orientation	[0; 0; 0](default), column vector	Array orientation in degrees, specified as a column vector of the form [α; β; γ]. The vector elements specify the bearing, downtilt, and slant, respectively.
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. See TR 38.901 Section 7.3.2.

Data Types: structure

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

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 UT velocity of 15 km/h:

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

```
fd = (v*1000/3600)/c*fc;     % UT max Doppler frequency in Hz

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

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

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

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

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

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

Transmit the input waveform through the channel.

```
rxWaveform = 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 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 = Inf`, a channel snapshot is taken for every input sample.
- When `SampleDensity = X`, a channel snapshot is taken at a rate of `Fcs = 2*X*MaximumDopplerShift`.

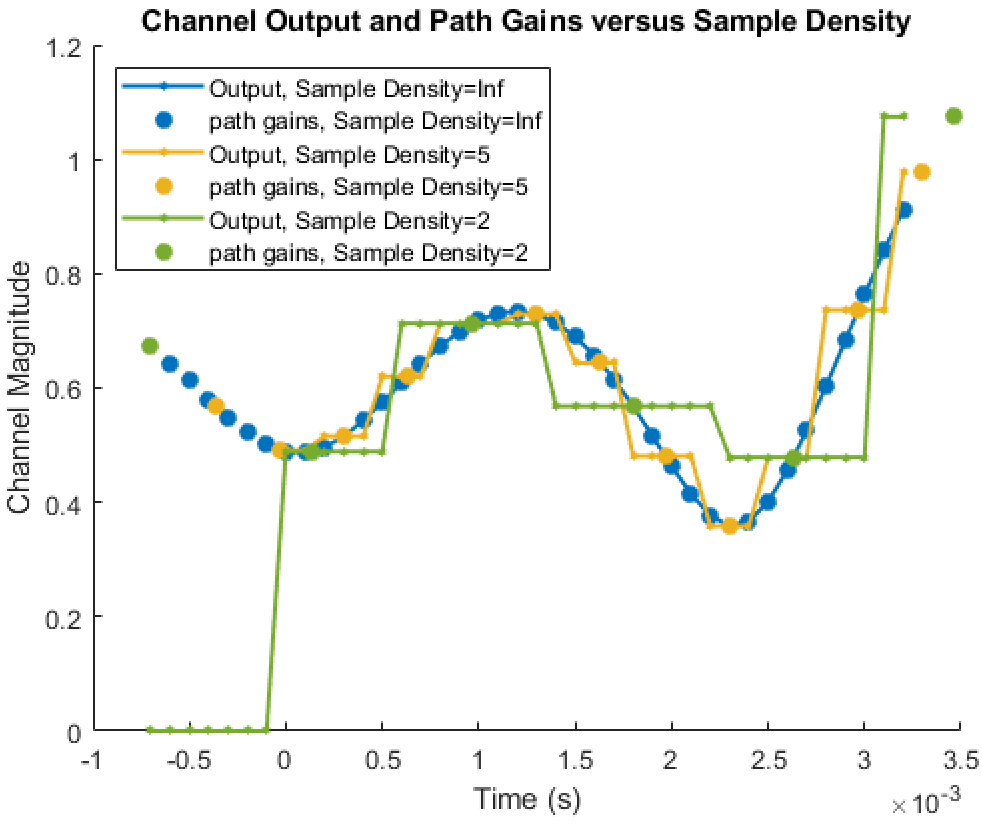
The `nrCDLChannel` object applies the channel snapshots to the input waveform by means of zero-order hold interpolation. The object takes an extra snapshot beyond the end of the input. Some of the final output samples use this extra value to minimize the interpolation error. The channel output contains 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 versus 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`

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 TR 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? (MATLAB).

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 (MATLAB).

`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

System 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 TR 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? (MATLAB).

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 (MATLAB).

`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 2-41.

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

Modulation Scheme	Number of Bits Per Symbol
'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 $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid \text{channel output for } c_i)}{\Pr(c_i = 1 \mid \text{channel output for } c_i)}\right)$.

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

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

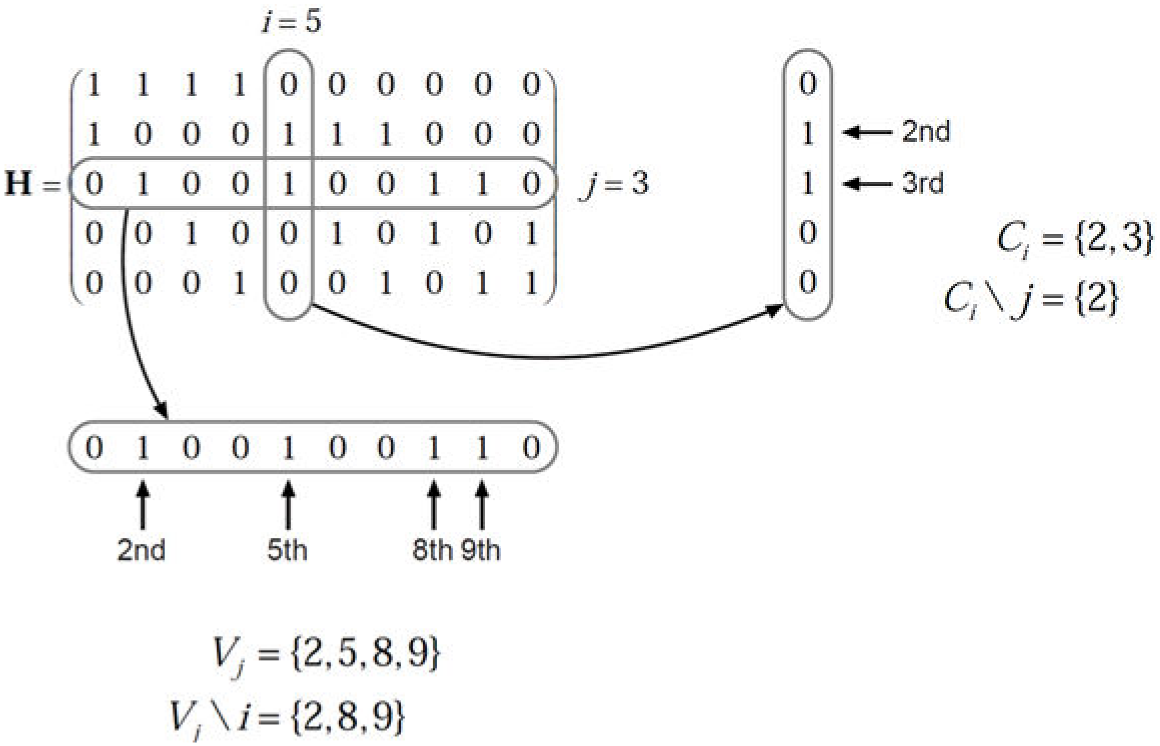
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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 ($\mathbf{Hc}^T = 0$) 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) $A_{mj} = \sum_{\substack{n \in N(m) \\ n \neq j}} \psi(L(q_{mn}))$,

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

(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

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

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 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*. 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 TR 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? (MATLAB).

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 (MATLAB).

`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

System 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 TR 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? (MATLAB).

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 (MATLAB).

`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 2-63.

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 $c = (c_0, c_1, \ldots, c_{n-1})$, the input to the LDPC decoder is the log-likelihood ratio (LLR) value $L(c_i) = \log\left(\frac{\Pr(c_i = 0 \mid \text{channel output for } c_i)}{\Pr(c_i = 1 \mid \text{channel output for } c_i)}\right)$.

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

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

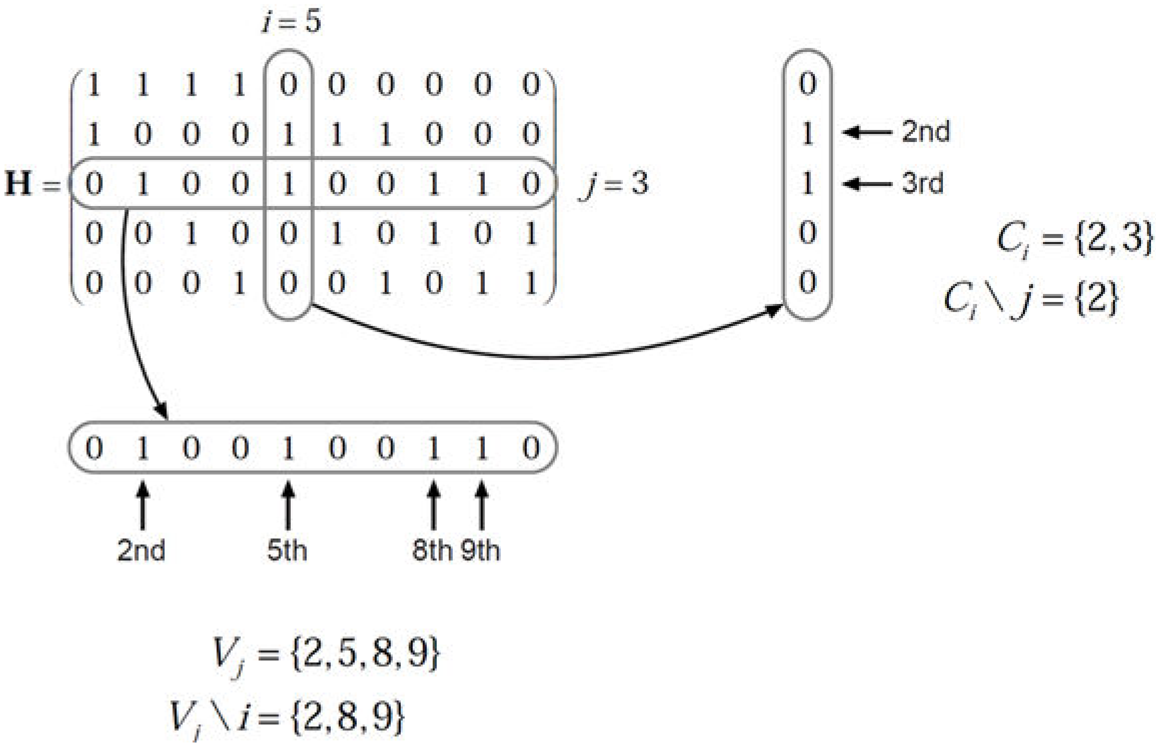
$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \backslash j} L(r_{j'i})$, initialized as $L(q_{ij}) = L(c_i)$ before the first iteration, and

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

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

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

(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

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

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

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 (MATLAB).

`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 ($N_T \times N_R$)-by-($N_T \times N_R$).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 ($N_T \times N_R$)-by-($N_T \times N_R$) with off-diagonal element properties as previously described. Each path has the same spatial correlation matrix.
  - 3-D array of size ($N_T \times N_R$)-by-($N_T \times N_R$)-by-N_P array — where each matrix of size ($N_T \times N_R$)-by-($N_T \times N_R$) 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 waveform spectrum received through a Tapped Delay Line (TDL) multi-input/multi-output (MIMO) channel model from TR 38.901 Section 7.7.2 using an `nrTDLChannel` System object.

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

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

tdl = nrTDLChannel;
tdl.DelayProfile = 'TDL-C';
```

```
tdl.DelaySpread = 300e-9;
tdl.MaximumDopplerShift = fd;
```

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

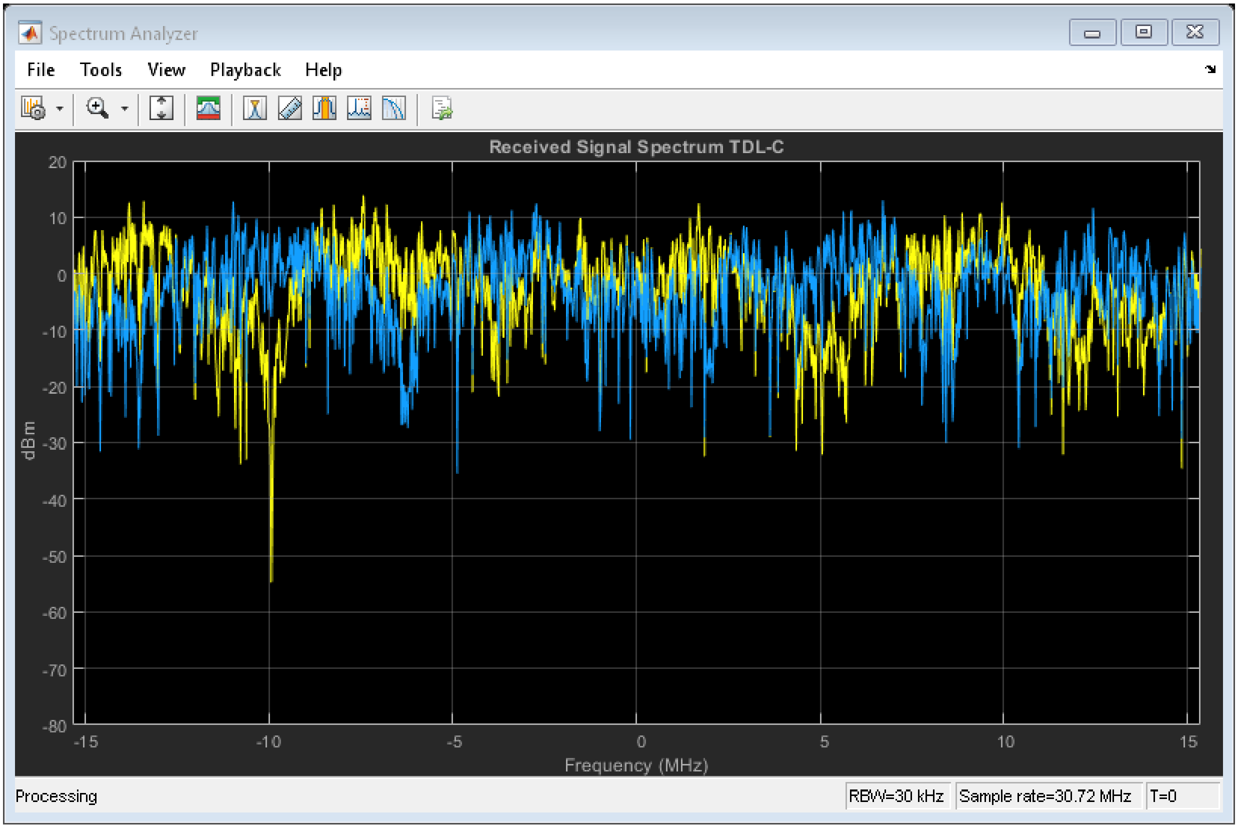
```
SR = 30.72e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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

Transmit the input waveform through the channel.

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

Plot the received waveform spectrum.

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



Plot Path Gains for TDL-E Delay Profile with SISO

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

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

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

Configure transmit and receive antenna arrays for SISO operation.

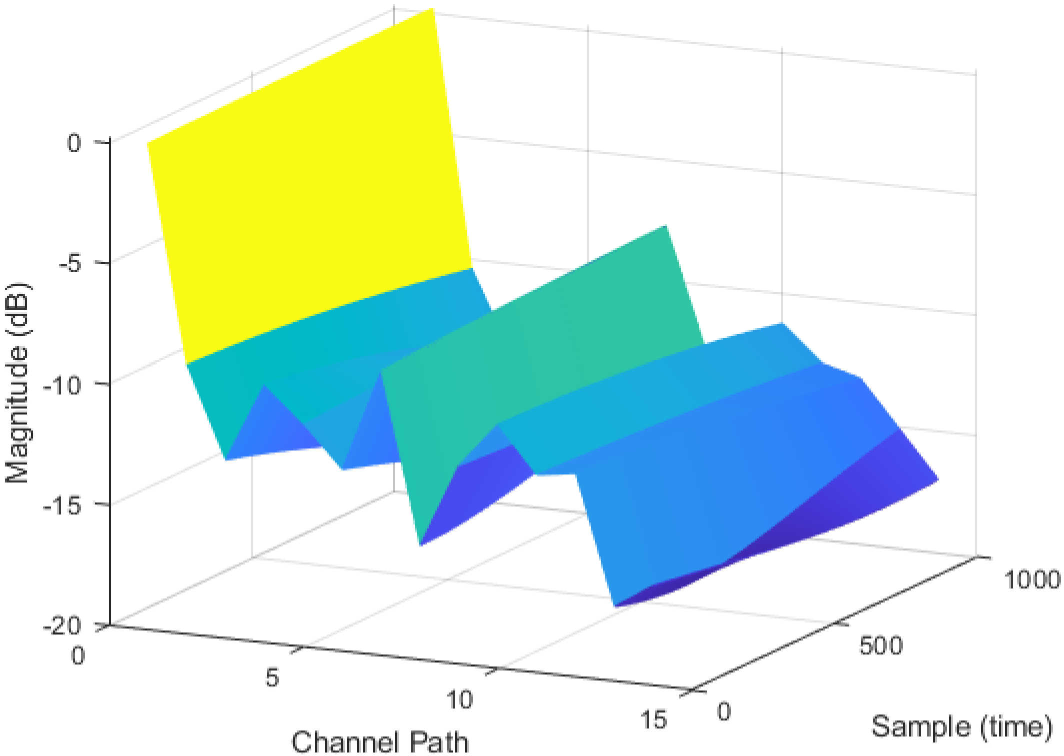
```
tdl.NumTransmitAntennas = 1;
tdl.NumReceiveAntennas = 1;
```

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

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

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

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



Transmission Over TDL-D Channel Model with Cross-Polar Antennas

Display waveform spectrum received through a Tapped Delay Line (TDL) channel model with delay profile TDL-D from TR 38.901 Section 7.7.2 and 4-by-2 high correlation cross-polar antennas as specified in TS 36.101 Annex B.2.3A.3.

Configure cross-polar antennas according to TS 36.101 Annex B.2.3A.3 4x2 high correlation.

```
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.

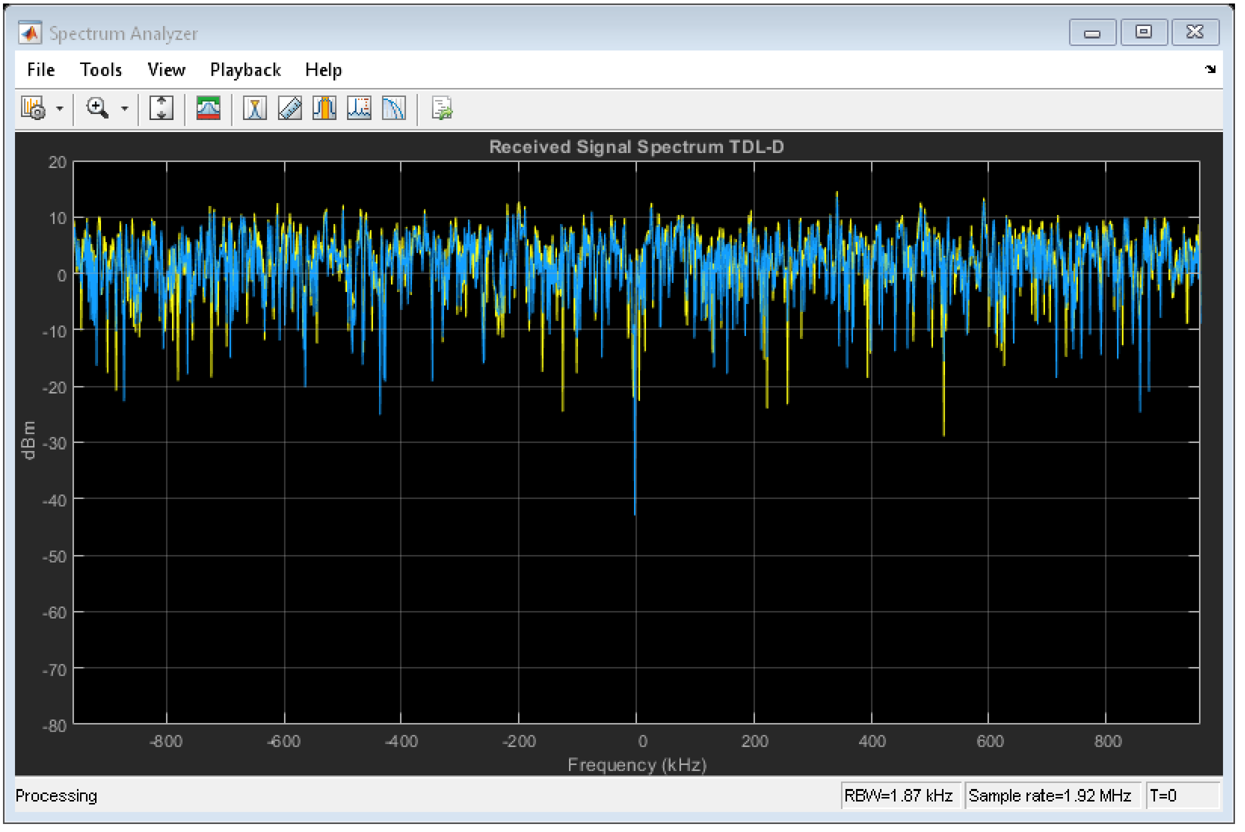
```
SR = 1.92e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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.

- Tap 1: Rician with average power 0 dB, K-factor 10 dB, and zero delay.
- Tap 2: 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 — Alphabetical List

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

`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 corresponds to the maximum number of RBs of a 10 MHz carrier with 15 kHz subcarrier spacing.

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
    {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
    {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

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 also specify one or more CSI-RS resources by using `"nzp"` and `"zp"` as string scalars or as elements of a string array, respectively.

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 also specify this property for one or more CSI-RS resources by using `"on"` and `"off"` as string scalars or as elements of a string array, respectively.

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 also specify the frequency density of one or more CSI-RS resources by using `"one"`, `"three"`, `"dot5even"`, and `"dot5odd"` as string scalars or as elements of a string array, respectively.

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 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 *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 2169 | vector of integers

Starting RB index of the CSI-RS resource allocation, relative to CRB 0, specified as one of these options.

For Single CSI-RS Resource

- Integer from 0 to 2169

For Multiple CSI-RS Resources

- Vector of integers in the range from 0 to 2169 — 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 *startingRB* in *FrequencyOccupation* IE or *startingPRB* in *CSI-RS-ResourceConfigMobility* IE.

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 only defines 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
    {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
    {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;
```

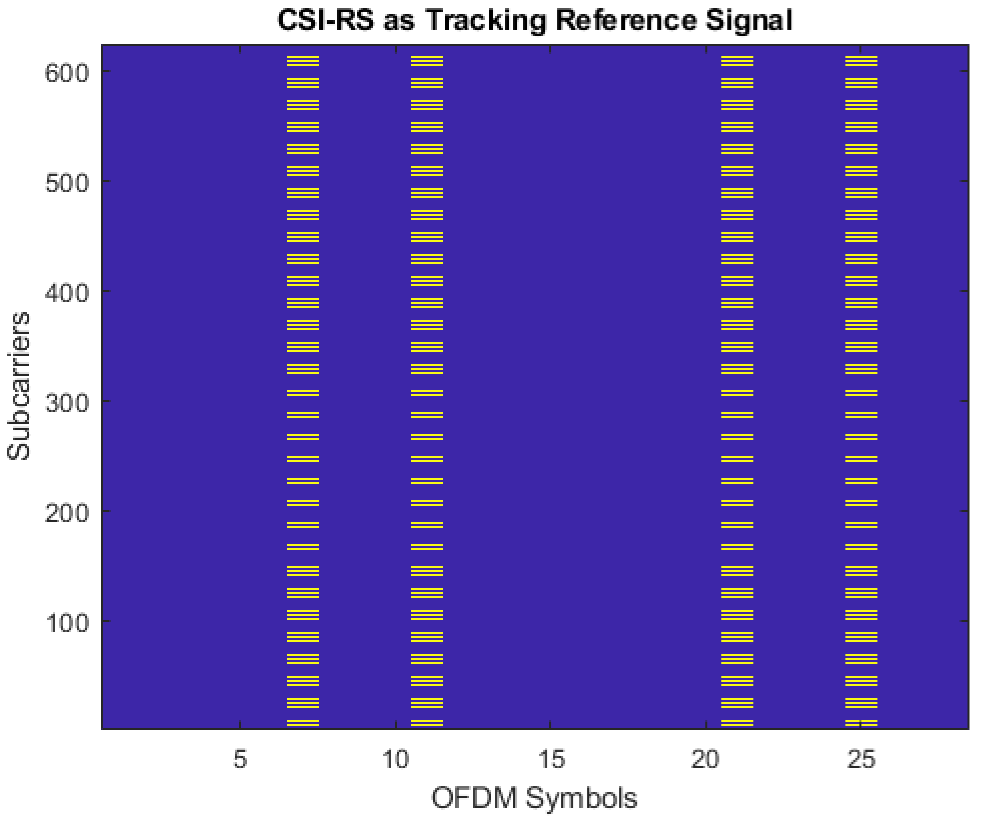
Concatenate the two slots to form the final grid.

```
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