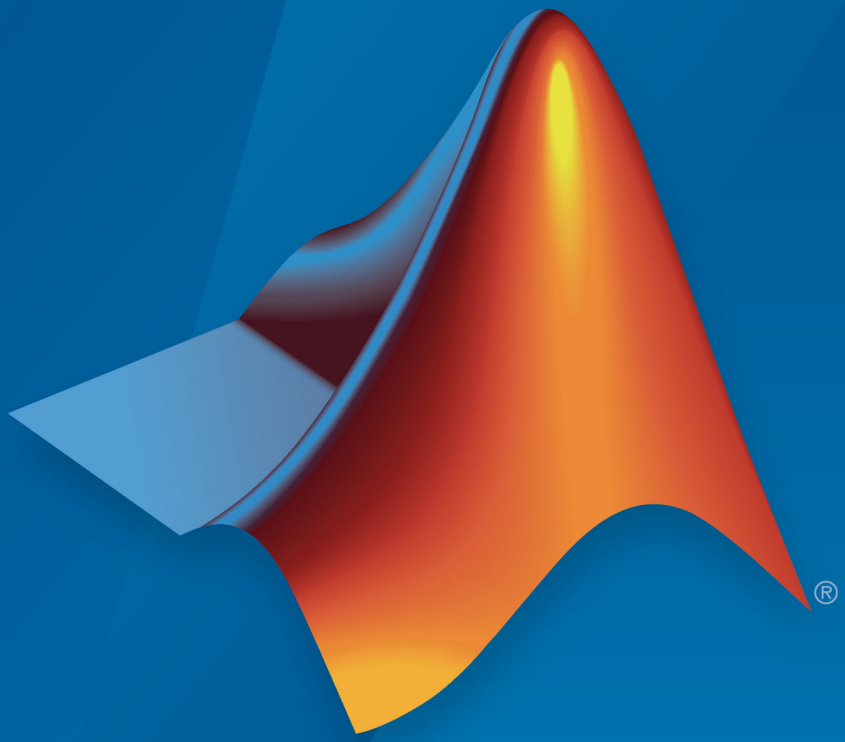
5G Toolbox™

Reference



MATLAB®

R2018b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

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

Phone: 508-647-7000

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

5G Toolbox™ Reference

Revision History

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

Contents

Functions — Alphabetical List

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`, `iil`, and `CRClen` are 10, `false`, and 11, respectively.

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. `E` must be larger than `K`, and $E \leq 8192$.

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, $N \le 2^{nmax}$, see TS 38.212 Section 5.3.1.2.

Data Types: `double`

`iil` — Output deinterleaving
`true` (default) | `false`

Output deinterleaving, specified as `true` or `false`.

- For DL configuration, specify `true`.
- For UL configuration, specify `false`.

Data Types: `logical`

`CRClen` — Number of appended CRC bits
`24` (default) | `11`

Number of appended CRC bits, specified as `24` or `11`.

- For DL configuration, specify `24`.
- For UL configuration, specify `11`.

The numbers 24 and 11 correspond to the polynomials gCRC24C and gCRC11, 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™.

See Also

Functions

`nrCRCDecode` | `nrDCIDecode` | `nrPolarEncode` | `nrRateRecoverPolar`

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` must be larger than the length of the input message `in`, and $E \leq 8192$.

Data Types: `double`

`nmax` — Base-2 logarithm of the encoded message's maximum length
`9` (default) | `10`

Base-2 logarithm of the encoded message's maximum length, specified as `9` or `10`.

- For DL configuration, specify `9`.
- For UL configuration, specify `10`.

If N is the length of the polar-encoded message in bits, then $N \le 2^{nmax}$. See TS 38.212 Section 5.3.1.2 [1].

Data Types: `double`

`iil` — Input interleaving
`true` (default) | `false`

Input interleaving, specified as `true` or `false`.

- For DL configuration, specify `true`.
- For UL configuration, specify `false`.

Data Types: `logical`

Output Arguments

`enc` — Polar-encoded message
column vector of binary values

Polar-encoded message, returned as a column vector of binary values. `enc` inherits its data type from the input message `in`.

The length of the polar-encoded message, N, is a power of two. For more information, see TS 38.212 Section 5.3.1.

- For DL configuration, $N \le 512$.
- For UL configuration, $N \le 1024$.

Data Types: `double` | `int8`

Limitations

The `nrPolarEncode` function assumes $nPC = 0$ and does not support non-zero parity-check bits, see TS 38.212 Section 5.3.1.2 [1]. The parity-check bits apply only for UL and input message of length K, where $18 \le K \le 25$.

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`

Topics

"5G New Radio Polar Coding"

Introduced in R2018b

nrRateMatchPolar

Polar rate matching

Syntax

```
rm = nrRateMatchPolar(enc,K,E)
rm = nrRateMatchPolar(enc,K,E,ibil)
```

Description

`rm = nrRateMatchPolar(enc,K,E)` returns the rate-matched output of length `E` for the polar-encoded input `enc` and information block length `K`, as specified in TS 38.212 Section 5.4.1 [1]. In this syntax, coded-bit interleaving is disabled. Use this syntax for downlink (DL) configuration.

`rm = nrRateMatchPolar(enc,K,E,ibil)` controls coded-bit interleaving. To enable coded-bit interleaving, set `ibil` to `true`. Use this syntax for uplink (UL) configuration with coded-bit interleaving enabled.

Examples

Perform Polar Rate Matching

Create a polar encoded random block of 512 bits and perform polar rate matching. Specify an information block of 56 bits and a rate-matched output of 864 bits.

```
N = 2^9;
K = 56;
E = 864;
in = randi([0 1],N,1);
out = nrRateMatchPolar(in,K,E)
```

```
out = 864×1
```

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

Input Arguments

enc — Polar-encoded message

column vector of binary values

Polar-encoded message, specified as a column vector of binary values.

The length of the polar-encoded message, N, is a power of two. For more information, see TS 38.212 Section 5.3.1.

- For DL configuration, $N \le 512$.
- For UL configuration, $N \le 1024$.

Data Types: `double` | `int8`

K — Length of information block in bits

positive integer

Length of information block in bits, specified as a positive integer. K includes the CRC bits if applicable

Data Types: `double`

E — Rate-matched output length in bits

positive integer

Rate-matched output length in bits, specified as a positive integer. E must be larger than K, and $E \le 8192$.

Data Types: `double`

`ibil` — Coded-bit interleaving
`false` (default) | `true`

Coded-bit interleaving, specified as `false` or `true`.

- For DL configuration, specify `false`.
- For UL configuration, specify `true`.

Data Types: `logical`

Output Arguments

`rm` — Rate-matched output data
column vector of binary values

Rate-matched output data, returned as an E-by-1 column vector of binary values. `rm` inherits its data type from the encoded message `enc`.

Data Types: `double` | `int8`

References

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

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`nrCRCEncode` | `nrDCIEncode` | `nrPolarEncode` | `nrRateRecoverPolar`

Topics
“5G New Radio Polar Coding”

Introduced in R2018b

nrRateRecoverPolar

Polar rate recovering

Syntax

```
rec = nrRateRecoverPolar(llr,K,N)
rec = nrRateRecoverPolar(llr,K,N,ibil)
```

Description

`rec = nrRateRecoverPolar(llr,K,N)` returns the rate-recovered output of length `N` for the soft input `llr` and information block length `K`, as specified in TS 38.212 Section 5.4.1 [1]. In this syntax, coded-bit deinterleaving is disabled. Use this syntax for downlink (DL) configuration.

`rec = nrRateRecoverPolar(llr,K,N,ibil)` controls coded-bit deinterleaving. To enable coded-bit deinterleaving, set `ibil` to `true`. Use this syntax for uplink (UL) configuration with coded-bit deinterleaving enabled.

Examples

Perform Polar Rate Recovery

Create a polar encoded random block of 512 bits and perform polar rate matching using `nrRateMatchPolar`. Perform polar rate recovery. Verify the results are identical to the original polar encoded input.

Specify an information block of 56 bits and an output of 864 bits for rate matching.

```
N = 512;
K = 56;
E = 864;
in = randi([0 1],N,1);
rateMatched = nrRateMatchPolar(in,K,E);
```

Perform rate recovery of the rate-matched data and information block of 56 bits. The length of the rate-recovered output, `N`, is the same as the length of the original polar encoded message.

```
rateRecovered = nrRateRecoverPolar(rateMatched,K,N);
```

Verify that the rate recovered output is identical to the original polar encoded input `in`.

```
isequal(rateRecovered,in)
```

```
ans = logical
   1
```

Input Arguments

`llr` — Log-likelihood ratio value input

column vector of real values

Log-likelihood ratio value input, specified as a column vector of real values. `llr` is the soft-demodulated input of length E, the same length as the rate-matched data vector before modulation.

Data Types: `single` | `double`

`K` — Length of information block in bits

positive integer

Length of information block in bits, specified as a positive integer. `K` includes the CRC bits if applicable

Data Types: `double`

`N` — Length of polar-encoded message in bits

power of two

Length of polar-encoded message in bits, specified as a power of two.

- $N \leq 512$ for DL configuration.
- $N \leq 1024$ for UL configuration.

For more details, see TS 38.212 Section 5.3.1 [1].

Data Types: `double`

`ibil` — Coded-bit deinterleaving
`false` for DL (default) | `true` for UL

Coded-bit deinterleaving, specified as `false` or `true`.

- For DL configuration, specify `false`.
- For UL configuration, specify `true`.

Data Types: `logical`

Output Arguments

`rec` — Rate-recovered output
column vector of real numbers

Rate-recovered output, returned as an N-by-1 column vector of real numbers.

Data Types: `single` | `double`

References

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

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`nrCRCDecode` | `nrDCIDecode` | `nrPolarDecode` | `nrRateMatchPolar`

Topics

“5G New Radio Polar Coding”

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

nrLDPCDecode

Low-density parity-check 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`.

`[out,actNumIter,finalParityChecks] = nrLDPCDecode( ___ ,Name,Value)` returns the LDPC-decoded output matrix with additional name-value pairs, in addition to the input arguments in previous syntax.

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].

Examples

Decode Low-Density Parity-Check Codeword

Encode two code block segments, each with length 2560. Include 36 filler bits at the end. Use a base graph number of 2.

```
bgn = 2;                    % Base graph nuumber
K = 2560;                   % Code block segment length
F = 36;                     % Number of filler bits per code block
C = 2;                      % Number of code blocks
txcbs = ones(K-F,C);
```

```
fillers = -1*ones(F,C);
txcbs = [txcbs;fillers];                % Add fillers
txcodedcbs = nrLDPCEncode(txcbs,bgn);   % Encode
rxcodedcbs = double(1-2*txcodedcbs);    % Convert to soft bits
FillerIndices = find(txcodedcbs(:,1) == -1);
rxcodedcbs(FillerIndices,:) = 0;        % Fillers have no LLR information
```

Decode the encoded codeword with a maximum of 25 iterations.

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

txcbs(end-F+1:end,:) = 0;           % Replace filler bits with 0

isequal(rxcbs,txcbs)
actualniters
```

```
ans =

  logical

   1


actualniters =

     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 a scalar of value `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 until after `maxNumIter` number of iterations are completed.

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 'info' or 'whole'.

If `'OutputFormat'` is specified as 'info', the number of rows in `out` is equal to the length of the information bits.

If `'OutputFormat'` is specified as 'whole', the number of rows in `out` is equal to the length of the codeword.

Data Types: `character string`

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

Decision method used for decoding, specified as 'hard' or 'soft'.

If `'DecisionType'` is specified as 'hard', `out` contains `'int8'` type decoded bits.

If `'DecisionType'` is specified as 'soft', `out` contains log-likelihood ratios of the same data type as `in`.

Data Types: `character string`

`Termination` — Decoding termination criteria
`'early'` (default) | `'max'`

Decoding termination criteria, specified as 'early' or 'max'.

If `'Termination'` is specified as 'early', decoding terminates when all parity checks are satisfied, or until `maxNumIter` number of iterations are completed.

If `'Termination'` is specified as 'max', decoding terminates after `maxNumIter` number of iterations.

Data Types: `character string`

Output Arguments

`out` — Decoded LDPC codeword
matrix

Decoded LDPC codeword or information bits (default), 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'`.

- When `'OutputFormat'` is specified as `'info'`, the number of rows in `out` is the number of information bits in an LDPC codeword.
- When `'OutputFormat'` is specified as `'whole'`, the number of rows in `out` is equal to the length of an LDPC codeword.

The data type of the output depends on the name-value pair argument `'DecisionType'`.

- When `'DecisionType'` is specified as `'hard'`, the data type of `out` is `int8`.
- When `'DecisionType'` is specified as `'soft'`, the data type of `out` is same as `in`.

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

`actNumIter` — Actual number of iterations
row vector | scalar

Actual number of iterations, returned as a row vector of positive integer(s).

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 *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`

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` | `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 filler bits, represented by `-1`, are present in the input `in` before encoding, they are replaced by `0`s. After encoding, the filler bits are again replaced by `-1`s.

The encoding includes puncturing of some of the systematic information bits.

Examples

Generate Low-Density Parity-Check Codeword

LDPC-encode two code block segments, each with length 2560. Include 36 filler bits at the end. Use a base graph number of 2.

```
bgn = 2;        % Base graph number
K = 2560;       % Code block segment length
F = 36;         % Number of filler bits per code block segment
C = 2;          % Number of code blocks
cbs = ones(K-F,C);
fillers = -1*ones(F,C);
cbs = [cbs;fillers];    % Input data for encoding
codedcbs = nrLDPCEncode(cbs,bgn);
size(codedcbs)
```

```
ans =

       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 treated as `0` when performing encoding.

Data Types: `double` | `int8`

`bgn` — Base graph number

1 | 2

Base graph number, specified as a scalar of value `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 of the 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

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 `out` of length `outlen` for input data matrix `in` with specified redundancy version `rv`, modulation type `mod`, and total number of transmission layers `nLayers`. 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 Rate-Matching of Two LDPC-Encoded Code Blocks

Perform rate-matching of two LDPC-encoded code blocks of length 3960 to a vector of length 8000. Use a single transmission layer with QPSK modulation with redundancy version `0`.

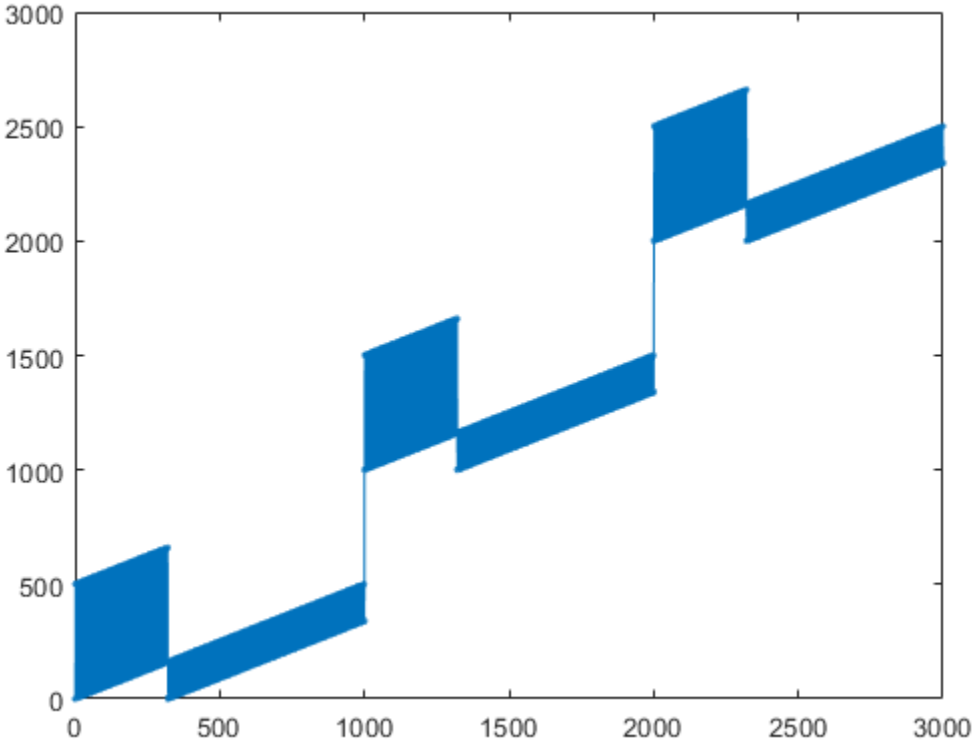
```
rv = 0;
mod = 'QPSK';
nLayers = 1;
```

```
encoded = ones(3960,2);
outlen = 8000;
ratematched = nrRateMatchLDPC(encoded,outlen,rv,mod,nLayers);
size(ratematched)

ans =

        8000           1
```

Observe Output Locations in Rate-Matched Code Blocks

Input some integer ramps as separate code blocks to the rate matching and observe the output locations after rate matching.

```
rv = 0;
in = [0 1000 2000] + (1:66*10)';
out = nrRateMatchLDPC(in,3000,rv,'QPSK',1);
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 scalar

Length of the rate-matched and concatenated output vector, specified as a positive integer scalar. `outlen` is the total number of coded bits available for transmission of the transport block, as specified in TS 38.212 Section 5.4.2.1 [1].

Data Types: `double`

`rv` — Redundancy version
`0-3` | positive integer scalar

Redundancy version, specified as a positive integer scalar between `0` and `3`.

Data Types: `double`

`mod` — Modulation scheme
`pi/2-BPSK` | `QPSK` | `16QAM` | `64QAM` | `256QAM`

Modulation scheme, specified as one of the following: pi/2-BPSK, QPSK, 16QAM, 64QAM, 256QAM.

Data Types: `character string`

`nLayers` — Number of transmission layers
`1-4` | positive integer scalar

Total number of transmission layers associated with the transport block, specified as a positive integer scalar between `1` and `4`.

Data Types: `double`

`Nref` — Parameter used for limited buffer rate matching
integer scalar

Parameter used for limited buffer rate matching, specified as an integer scalar. `Nref` is defined in TS 38.212 Section 5.4.2.1 [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

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` with specified transport block length `trblklen`, target code rate `R`, redundancy version `rv`, modulation type `mod`, and total number of transmission layers `nLayers`. 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 Rate-Recovery of Input Vector to Code Block

Perform rate-recovery of an input vector of 4500 bits to a code block. Use a single transmission layer with QPSK modulation with redundancy version `0`. The length of the original transport block is `4000`, and the number of scheduled block segments is `1`.

```
R = 0.5;                % Target code rate
trblklen = 4000;        % Transport block length
rv = 0;                 % Redundancy version
mod = 'QPSK';           % Modulation type
nlayers = 1;            % Number of layers
numCB = 1;              % Number of scheduled code blocks

sbits = ones(4500,1);
raterec = nrRateRecoverLDPC(sbits,trblklen,R,rv,mod,nlayers,numCB);
size(raterec)
```

```
ans =

      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 scalar

Original transport block length, specified as a nonnegative integer scalar.

Data Types: `double`

`R` — Target code rate

`(0,1)` | real scalar

Target code rate, specified as a real scalar where `0<R<1`.

Data Types: `double`

`rv` — Redundancy version
`0-3` | positive integer scalar

Redundancy version, specified as a positive integer scalar between `0` and `3`.

Data Types: `double`

`mod` — Modulation scheme
`pi/2-BPSK` | `QPSK` | `16QAM` | `64QAM` | `256QAM`

Modulation scheme, specified as one of the following: pi/2-BPSK, QPSK, 16QAM, 64QAM, 256QAM.

Data Types: `character string`

`nLayers` — Number of transmission layers
`1-4` | positive integer scalar

Total number of transmission layers associated with the transport block, specified as a positive integer scalar between `1` and `4`.

Data Types: `double`

`numCB` — Number of scheduled code block segments
positive integer scalar

Number of scheduled code block segments, specified as a positive integer scalar. `numCB` is less than or equal to the number of code block segments for a transport block.

Data Types: `double`

`Nref` — Parameter used for limited buffer rate matching
integer scalar

Parameter used for limited buffer rate matching, specified as an integer scalar. `Nref` is defined in TS 38.212 Section 5.4.2.1 [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

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)
```

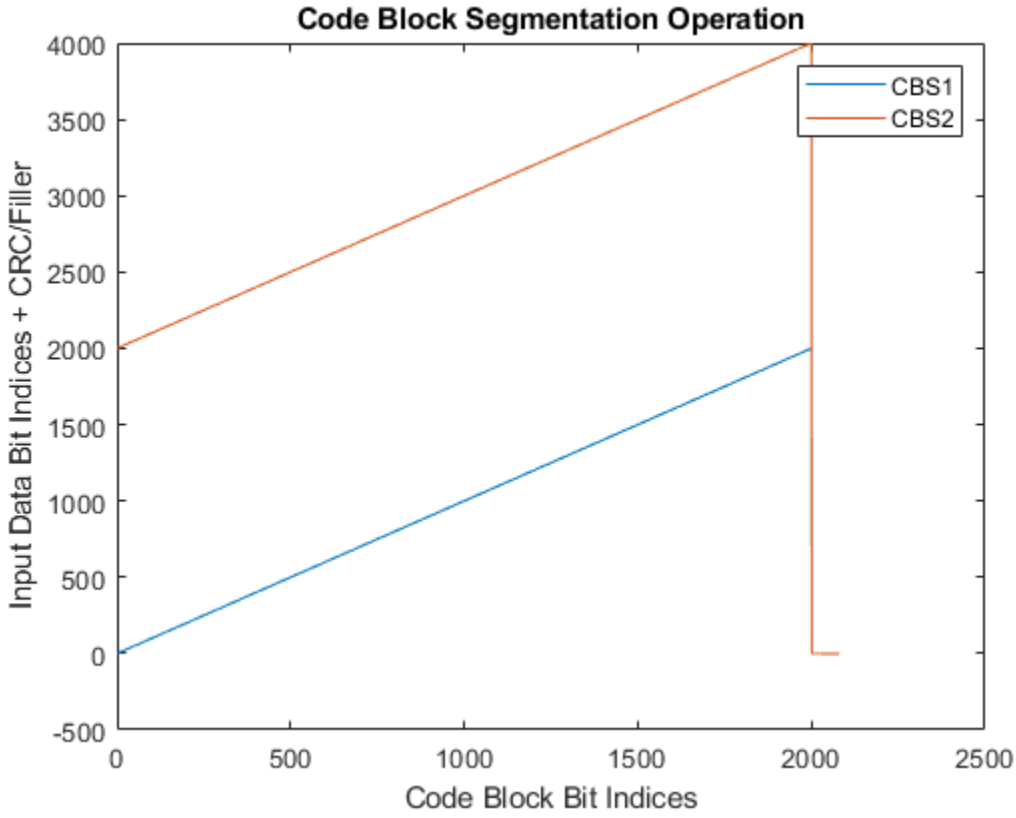
```
ans =
```

```
    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 $B \le Kcb$, then the function does not perform segmentation and does not append CRC to the resulting code block. If $B > Kcb$, the segmentation results in several smaller code blocks with a type-24B CRC bits appended.

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

Data Types: `double` | `int8`

References

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

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions
`nrCodeBlockDesegmentLDPC` | `nrLDPCEncode` | `nrLDPCEncode` | `nrRateMatchLDPC` | `nrRateMatchLDPC`

Introduced in R2018b

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.

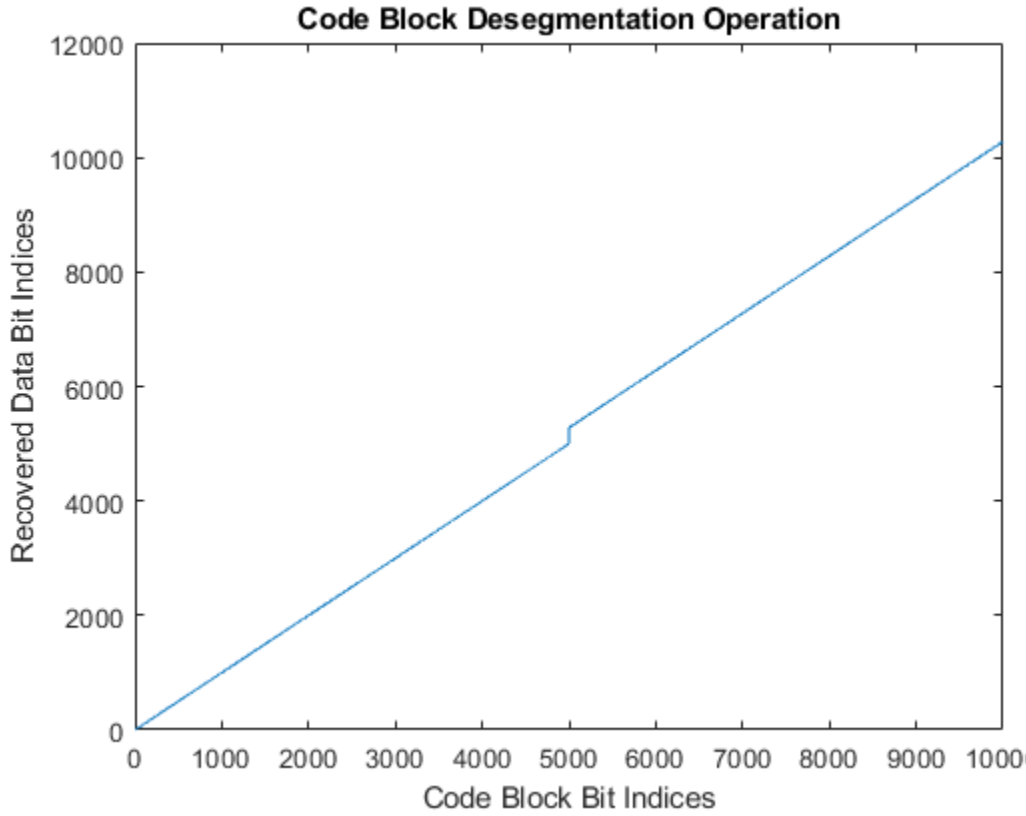
```
bgn = 1;
blklen = 10000;
cbs = nrCodeBlockSegmentLDPC(randi([0 1],blklen,1),bgn);
```

```
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

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

DCI Decoding of Sample Codeword

Perform DCI encoding of a random sequence of binary values of length 32. Set the radio network temporary identifier (RNTI) to 100. The RNTI masks the CRC parity bits. Set the length of the rate-matched DCI codeword to 240 bits.

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

Perform DCI decoding of 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);
```

Verify that the transmitted and received message bits are identical. The recovered mask value is the RNTI value used for CRC masking.

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

```
ans =

   logical

    1

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™.

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.

Examples

Encode DCI Message Bits

Perform DCI encoding of a random sequence, and return the rate-matched DCI codeword.

Create a random sequence of binary values of length 32. Set RNTI to 100 and the length of the rate-matched output to 240 bits.

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

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` — Radio network temporary identifier
integer

Radio network temporary identifier of user equipment, specified as an integer from 0 to 65535.

Data Types: `double`

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

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

Data Types: `double`

Output Arguments

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

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

Data Types: `double` | `int8`

References

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

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrDCIDecode` | `nrPDCCH` | `nrPDCCHDecode`

Introduced in R2018b

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 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™.

See Also

Functions
`nrPBCHDMRS` | `nrPBCHDMRSIndices` | `nrPBCHDecode` | `nrPBCHIndices` | `nrPBCHPRBS` | `nrPRBS` | `nrPSS` | `nrSSS`

Introduced in R2018b

nrPBCHIndices

Get 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™.

See Also

Functions

`nrPBCH` | `nrPBCHDMRSIndices` | `nrPSSIndices` | `nrSSSIndices`

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` 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

nrPBCHPRBS

Generate PBCH pseudorandom 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 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 returned sequence

nonnegative integer

Number of elements in the returned 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 pseudorandom scrambling sequence
logical column vector | numeric column vector

PBCH pseudorandom 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`. Otherwise, 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™.

See Also

Functions

`nrPBCH` | `nrPBCHDecode` | `nrPBCHIndices` | `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 the 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. Specify with `nid` the physical layer cell identity number (0 to 1007) or the higher layer parameter *`dataScramblingIdentityPDSCH`* (0 to 1023). For more information on these values, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — Radio network temporary identifier
integer

Radio network temporary identifier of user equipment, specified as an integer from 0 to 65535.

Data Types: `double`

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

Data type of 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`

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

`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 the scrambling identity `nid`. The radio network temporary identifier (RNTI) of the user equipment is specified by `rnti`.

`[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. Specify with `nid` the physical layer cell identity number (0 to 1007) or the higher layer parameter *`dataScramblingIdentityPDSCH`* (0 to 1023). For more information on these values, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — Radio network temporary identifier
integer

Radio network temporary identifier of user equipment, specified as an integer from 0 to 65535.

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` 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
`nrLayerDemap` | `nrPDSCH` | `nrPDSCHPRBS` | `nrSymbolDemodulate`

Introduced in R2018b

nrPDSCHPRBS

Generate PDSCH pseudorandom 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 `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 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. Specify with `nid` the physical layer cell identity number (0 to 1007) or the higher layer parameter *`dataScramblingIdentityPDSCH`* (0 to 1023). For more information on these values, see TS 38.331 Section 6.3.2.

Data Types: `double`

`rnti` — Radio network temporary identifier

integer

Radio network temporary identifier of user equipment, specified as an integer from 0 to 65535.

Data Types: `double`

`q` — Codeword number

`0` | `1`

Codeword number, specified as `0` or `1`.

Data Types: `double`

n — Number of elements in returned sequence
nonnegative integer

Number of elements in returned 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 pseudorandom scrambling sequence
logical column vector | numeric column vector

PDSCH pseudorandom 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`. Otherwise, the output data type is `logical`.

Data Types: `double` | `single` | `logical`

`cinit` — Initialization value for PRBS generator
integer from 0 to 2147468287

Initialization value for PRBS generator, returned as an integer from 0 to 2147468287.

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™.

See Also

Functions
`nrPDSCH` | `nrPDSCHDecode` | `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);
```

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);
```

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

Scrambling identity, specified as an integer from 0 to 65535. Specify with `nid` the physical layer cell identity number (0 to 1007) or the higher layer parameter *pdcch-DMRS-ScramblingID* (0 to 65535). For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — User equipment identifier

integer

User equipment identifier, specified as an integer from 0 to 65535.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a user equipment 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 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

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
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);
```

Demodulate and compare the soft bits with the input codeword.

```
nVar = 0;
rxdicw = nrPDCCHDecode(sym,nid,nrnti,nVar);

isequal(dcicw,rxdicw<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

Scrambling identity, specified as an integer from 0 to 65535. Specify with `nid` the physical layer cell identity number (0 to 1007) or the higher layer parameter *`pdcch-DMRS-ScramblingID`* (0 to 65535). For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — User equipment identifier

integer

User equipment identifier, specified as an integer from 0 to 65535.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a user equipment 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` 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 pseudorandom 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 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 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

Scrambling identity, specified as an integer from 0 to 65535. Specify with `nid` the physical layer cell identity number (0 to 1007) or the higher layer parameter *pdcch-DMRS-ScramblingID* (0 to 65535). For more information on these values, see TS 38.211 Section 7.3.2.3 and 7.4.1.3.

Data Types: `double`

`nrnti` — User equipment identifier

integer

User equipment identifier, specified as an integer from 0 to 65535.

- If `nid` is the PDCCH DMRS scrambling identity, `nrnti` is the cell radio network temporary identifier (C-RNTI) in a user equipment 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 returned sequence

nonnegative integer

Number of elements in the returned 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 pseudorandom scrambling sequence
logical column vector | numeric column vector

PDCCH pseudorandom 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`. Otherwise, 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™.

See Also

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

Introduced in R2018b

nrBCH

Broadcast channel (BCH) encoding

Syntax

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

Description

`cdblk = nrBCH(mib,sfn,hrf,lssb,idxoffset,ncellid)` encodes the master information block (MIB) payload, `mib`, in accordance with TS 38.212, Section 7.1 [1]. The function returns the encoded broadcast channel (BCH) transport block `cdblk`. The other `nrBCH` input arguments are as follows.

- The system frame number `sfn`.
- The half frame bit `hrf.`
- The number of candidate synchronization signal / physical broadcast channel (SS/PBCH) blocks in a half frame `lssb`.
- The subcarrier offset / SS block index `idxoffset`.
- The physical layer cell identity number `ncellid`.

Examples

Encode MIB Payload with Subcarrier Offset

Perform encoding on a random input MIB payload with subcarrier offset.

Specify the physical layer cell identity number, system frame number, and half frame bit.

```
ncellid = 321;                  % Physical layer cell identity number
sfn = 10;                       % System frame number, as a decimal
hrf = 1;                        % Half frame bit
```

Generate a random MIB payload, specify the number of candidate SS/PBCH blocks as eight, and set the subcarrier offset to 18.

```
mib = randi([0 1],24,1,'int8');  % MIB payload
lssb = 8;                        % Number of candidate SS/PBCH blocks
idxoffset = 18;                  % Subcarrier offset in the range 0 to 23
```

Encode input MIB payload and return encoded BCH transport block.

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

Encode MIB with SS/PBCH Block Index

Perform encoding on a random input MIB payload with SS/PBCH block index.

Specify the physical layer cell identity number, system frame number, and half frame bit.

```
ncellid = 321;                   % Physical layer cell identity number
sfn = 10;                        % System frame number, as a decimal
hrf = 1;                         % Half frame bit
```

Generate a random MIB payload, specify the number of candidate SS/PBCH blocks as 64, and set the SS/PBCH block index to 13.

```
mib = randi([0 1],24,1,'int8');  % MIB payload
lssb = 64;                       % Number of candidate SS/PBCH blocks
idxoffset = 13;                  % SS/PBCH in the range 0 to 63
```

Encode input MIB payload and return encoded BCH transport block.

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

Input Arguments

`mib` — MIB payload to be encoded

24-by-1 binary column vector

MIB payload to be encoded, specified as a 24-by-1 binary column vector.

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
`0` | `1`

Half frame bit, specified as either `0` or `1`.

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` — SS/PBCH block index
nonnegative integer

Subcarrier offset / SS/PBCH 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/PBCH 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 of 864 bits, returned as an 864-by-1 binary column vector.

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

`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,mib,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 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,mib,lsbofsfn,hrf,msbidxoffset] = nrBCHDecode(softbits,L,lssb,ncellid)` also returns the following.

- The unscrambled master information block (MIB) payload `mib`.
- The four least significant bits (LSBs) of the system frame number `lsbofsfn`.
- The half frame bit `hrf`.
- The decoded most significant bits (MSBs) `msbidxoffset`.

The other `nrBCHDecode` input arguments are the number of candidate synchronization signal / physical broadcast channel (SS/PBCH) blocks `lssb` and the physical layer cell identity number `ncellid`.

Examples

Decode Scrambled Transport Block

Return the scrambled transport block, unscrambled MIB payload, and other relevant information.

Specify the physical layer cell identity number, system frame number, and half frame bit.

```
ncellid = 321;                    % Physical layer cell identity number
sfn = 10;                         % System frame number, as a decimal
hrf = 1;                          % Half frame bit
```

Generate a random MIB payload, specify the number of candidate SS/PBCH blocks as eight, and set the subcarrier offset to 18.

```
mib = randi([0 1],24,1,'int8');  % MIB payload
lssb = 8;                        % Number of candidate SS/PBCH blocks
idxoffset = 18;                  % Subcarrier offset in the range 0 to 23
```

Encode input MIB payload and return the encoded BCH transport block.

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

Specify the LLRs in terms of the encoded BCH transport block.

```
softbits = double(1-2*cdblk);    % LLRs to be decoded
```

Decode to recover the scrambled transport block and unscrambled MIB payload.

```
L = 8;                           % Polar decoding list length

[scrblk,errFlag,rxMIB,rxlsbofsfn,rxHRF,rxidxoffset] = …
nrBCHDecode(softbits,L,lssb,ncellid);
```

Input Arguments

`softbits` — LLRs to be decoded

864-by-1 real column vector

LLRs to be decoded, specified as an 864-by-1 real 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 transport block
32-by-1 binary column vector

Decoded scrambled 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 either `0` or `1`. If `errFlag` is returned as `1`, then an error has occurred.

Data Types: `uint32`

`mib` — Decoded and unscrambled MIB payload
24-by-1 binary column vector

Decoded and unscrambled MIB payload, returned as a 24-by-1 binary column vector.

Data Types: `logical`

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

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

Data Types: `logical`

`hrf` — Half frame bit
`0` | `1`

Half frame bit, returned as either `0` or `1`.

Data Types: `logical`

`msbidxoffset` — Decoded MSBs
scalar | 3-by-1 column vector

Decoded MSBs, 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*.

Extended Capabilities

C/C++ Code Generation

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

See Also

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

Topics

“NR Synchronization Procedures”

Introduced in R2018b

nrDLSCHInfo

Get downlink shared channel (DL-SCH) information

Syntax

```
info = nrDLSCHInfo(tbs,tcr)
```

Description

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

Examples

Get DL-SCH Information

Get DL-SCH information for an input transport block of size 8456 and target code rate 517/1024. The displayed DL-SCH information shows the following.

- 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 13728.

For more details on all the fields in the `nrDLSCHInfo` output structure, see the output argument `info`.

```
tbs = 8456;
tcr = 517/1024;
nrDLSCHInfo(tbs,tcr)
```

```
ans =

  struct with fields:

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

Input Arguments

`tbs` — Transport block size

nonnegative integer

transport block size, specified as a nonnegative integer.

Data Types: `double`

`tcr` — Target code rate

real number

Target code rate, specified as a real number in the range $0 < \texttt{tcr} < 1$.

Data Types: `double`

Output Arguments

`info` — DL-SCH information

structure

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

Parameter Field	Values	Description
CRC	`'16'`, `'24A'`	CRC polynomial selection

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

Extended Capabilities

C/C++ Code Generation

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

See Also

Functions

`nrPDSCH` | `nrPDSCHDecode`

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 1-116. 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 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

Initialization value for the PRBS generator, specified as an integer from 0 to 2147483647.

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`. Otherwise, the data type of `seq` 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™.

See Also

Functions
nrPBCHPRBS | nrPDCCHPRBS | nrPDSCHPRBS

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™.

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™.

See Also

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

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

nrPSSIndices

Get 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™.

See Also

Functions
`nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPSS` | `nrSSSIndices`

Introduced in R2018b

nrSSSIndices

Get 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™.

See Also

Functions
`nrPBCHDMRSIndices` | `nrPBCHIndices` | `nrPSSIndices` | `nrSSS`

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 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™.

See Also

Functions

`nrPBCH` | `nrPBCHDMRS` | `nrPSSIndices` | `nrSSS`

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 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™.

See Also

Functions

`nrPBCH` | `nrPBCHDMRS` | `nrPSS` | `nrSSSIndices`

Introduced in R2018b

nrPBCHDMRSIndices

Get 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™.

See Also

Functions
`nrPBCHDMRS` | `nrPBCHIndices` | `nrPSSIndices` | `nrSSSIndices`

Introduced in R2018b

nrPBCHDMRS

Generate PBCH DM-RS symbols

Syntax

```
sym = nrPBCHDMRS(ncellid,ibar_SSB)
sym = nrPBCHDMRS(ncellid,ibar_SSB,'OutputDataType',datatype)
```

Description

`sym = nrPBCHDMRS(ncellid,ibar_SSB)` returns the physical broadcast channel (PBCH) demodulation reference signal (DM-RS) symbols for the physical layer cell, identified by `ncellid`. The `ibar_SSB` input specifies the time-dependent part of the DM-RS scrambling initialization. The function implements TS 38.211 Section 7.4.1.4.1 [1].

`sym = nrPBCHDMRS(ncellid,ibar_SSB,'OutputDataType',datatype)` specifies the data type of the DM-RS symbol.

Examples

Generate PBCH DM-RS Symbols

Generate the sequence of 144 PBCH DM-RS symbols associated with the third SS block (`i_SSB` = 2) in the second half frame (`n_hf` = 1) of a frame.

```
ncellid = 17;
i_SSB = 2;
n_hf = 1;
ibar_SSB = i_SSB + (4*n_hf);
```

```
dmrs = nrPBCHDMRS(ncellid,ibar_SSB);
```

Input Arguments

`ncellid` — Physical layer cell identity number

integer

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

Data Types: `double`

`ibar_SSB` — Time-dependent part of DM-RS scrambling initialization

integer from 0 to 7 (default)

Time-dependent part of the DM-RS scrambling initialization, specified as an integer from 0 to 7. `ibar_SSB` is derived in a synchronization signal (SS) burst configuration, from the least significant bits (LSBs) of the SS/PBCH block index and the half-frame number.

- If the number of SS/PBCH blocks per half-frame is 4, `ibar_SSB` = $i_{SSB} + 4 \times n_{hf}$, where i_{SSB} is the two LSBs of the SS/PBCH block index (0 to 3). n_{hf} is the half-frame number within the frame (0,1).
- If the number of SS/PBCH blocks per half-frame is 8 or 64, `ibar_SSB` is the three LSBs of the SS/PBCH block index (0 to 7).

Data Types: `double`

`datatype` — Data type of output symbols

`'double'` (default) | `'single'`

Data type of 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™.

See Also

Functions

`nrPBCH` | `nrPBCHDMRSIndices` | `nrPRBS` | `nrPSS` | `nrSSS` | `nrSymbolModulate`

Introduced in R2018b

nrPerfectChannelEstimate

Perfect channel estimation

Syntax

```
hest = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs,
initialSlot)
hest = nrPerfectChannelEstimate( ___ ,toffset)
hest = nrPerfectChannelEstimate( ___ ,toffset,sampleTimes)
hest = nrPerfectChannelEstimate( ___ ,cpl)
```

Description

`hest = nrPerfectChannelEstimate(pathGains,pathFilters,nrb,scs, initialSlot)` performs perfect channel estimation and returns the perfect channel estimate. 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`.

`hest = 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.

`hest = nrPerfectChannelEstimate( ___ ,toffset,sampleTimes)` also specifies the sample times of the channel snapshots in addition to the input arguments in the first previous syntax.

`hest = nrPerfectChannelEstimate( ___ ,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.

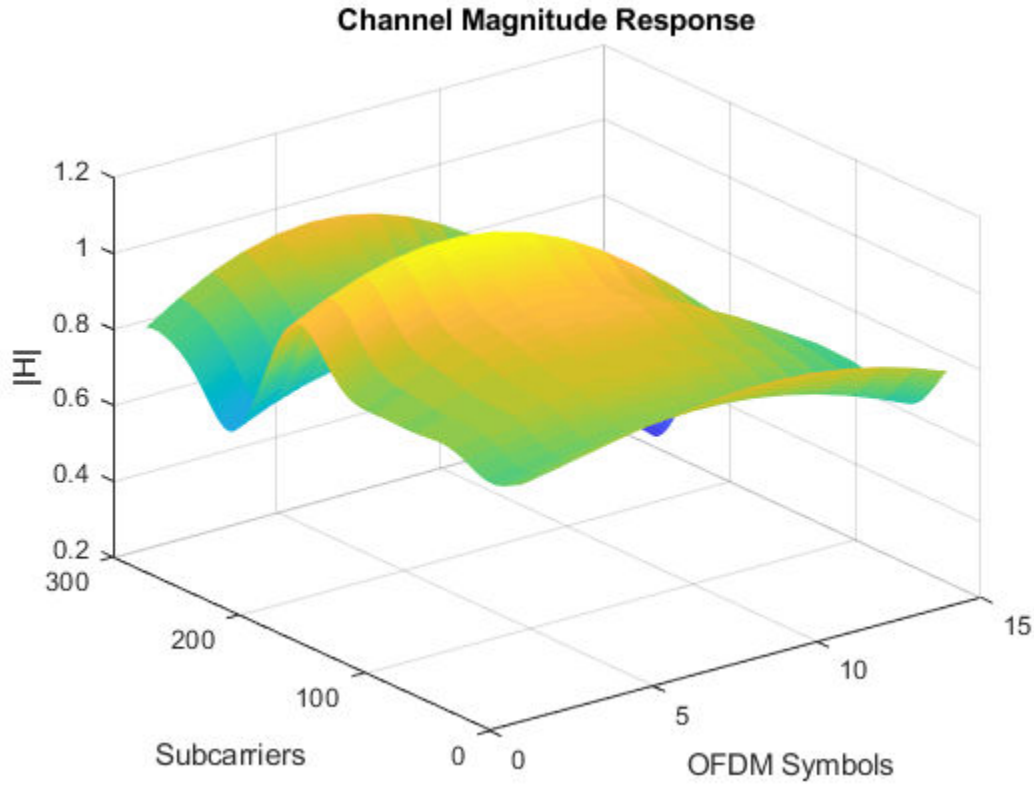
```
NRB = 25;
SCS = 15;
nSlot = 0;

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');
```



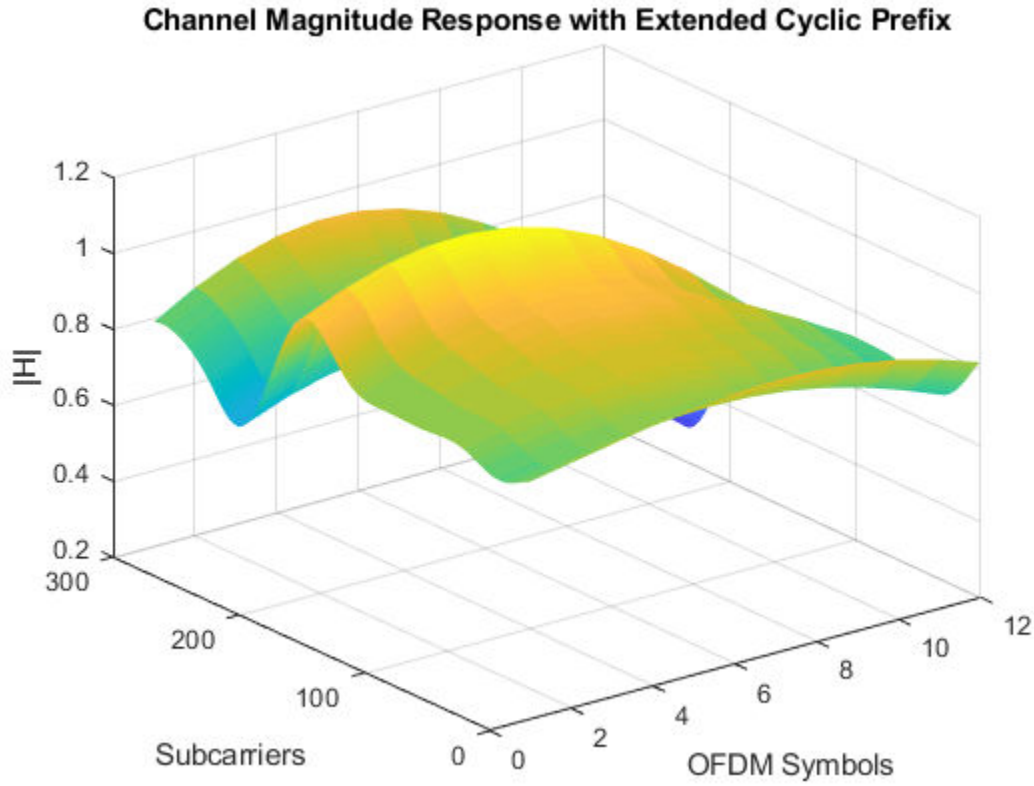
Repeat the channel estimate for extended cyclic prefix.

```
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 pahh filter and path gains.

```
offset = nrPerfectTimingEstimate(pathGains,pathFilters);
```

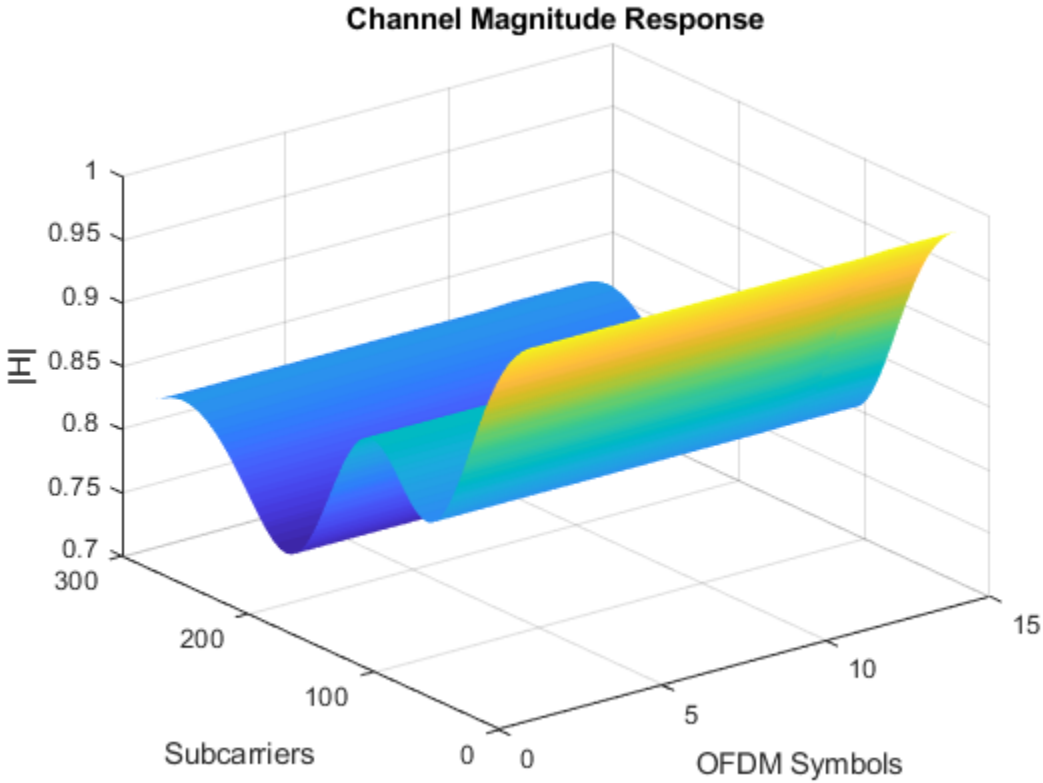
Perform perfect channel estimation. Use the specified number of blocks, subcarrier spacing, slot number, timing offset, and sample times.

```
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

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 lengths 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'` | `'extended'`

Cyclic prefix length, specified as one of these values.

- `'normal'` — Use this value to specify normal cyclic prefix.
- `'extended'` — Use this value to specify extended cyclic prefix. Note that for the numerologies specified in TS 38.211 Section 4.2, extended cyclic prefix length is applicable only for 60 kHz subcarrier spacing.

Data Types: `char` | `string`

Output Arguments

`hest` — Perfect channel estimate
N_{SC}-by-N_{SYM}-by-N_R-by-N_T array of complex numbers

Perfect channel estimate, returned as an N_{SC}-by-N_{SYM}-by-N_R-by-N_T array of complex or real numbers, 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.

`hest` 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
`nrPerfectTimingEstimate`

System Objects
`nrCDLChannel` | `nrTDLChannel`

Introduced in R2018b

nrPerfectTimingEstimate

Perfect timing estimation

Syntax

```
[toffset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters)
```

Description

`[toffset,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.

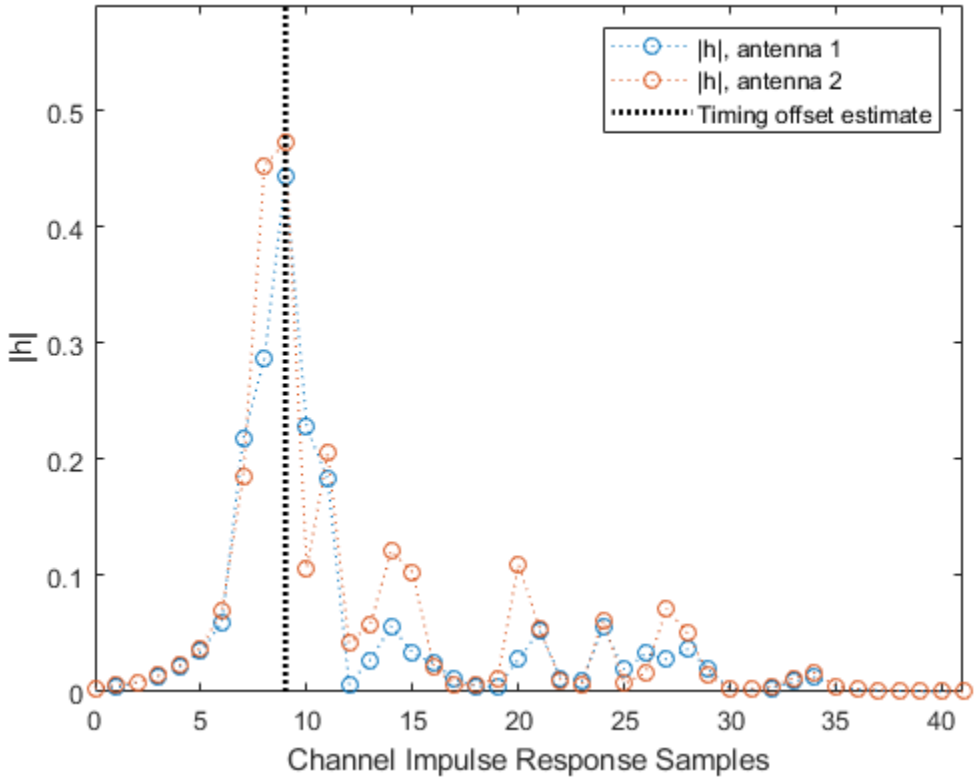
```
pathFilters = getPathFilters(tdl);
```

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

`toffset` — 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

System Objects

`nrCDLChannel` | `nrTDLChannel`

Functions

`nrPerfectChannelEstimate`

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.

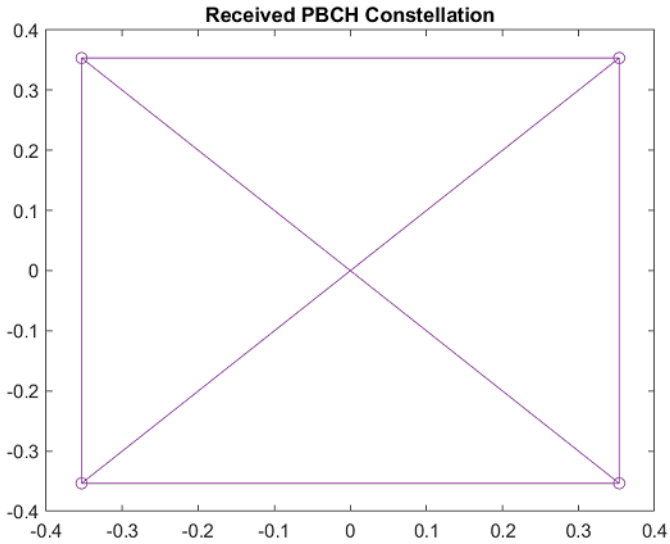
```
hEstGrid = repmat(permute(H.',[3 4 1 2]),[240 4]);
nEst = 0.1;
```

Perform OFDM demodulation.

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

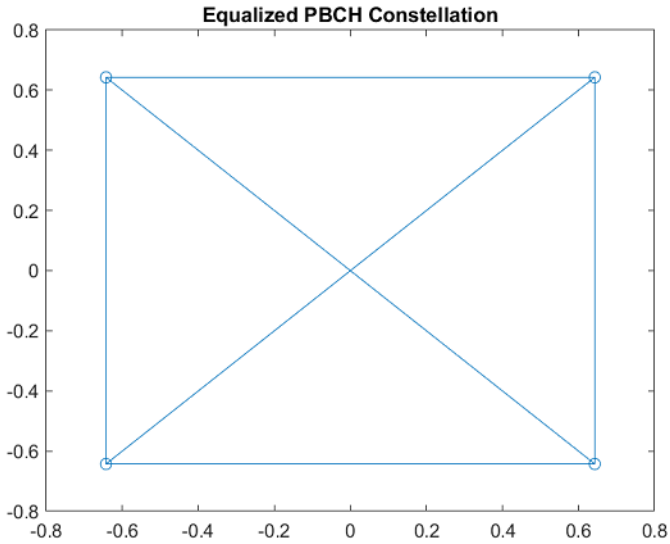
To prepare for PBCH decoding, use `nrExtractResources` to extract symbols from received and channel estimate grids. Plot the received PBCH constellation.

```
[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`. You can extract resource elements from a resource array with different dimensionality than the resource array addressed by the given indices. In this syntax, the specified indices are one-based using linear indexing form. For more details on the resource extraction process, see “Algorithms” on page 1-178.

`[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.

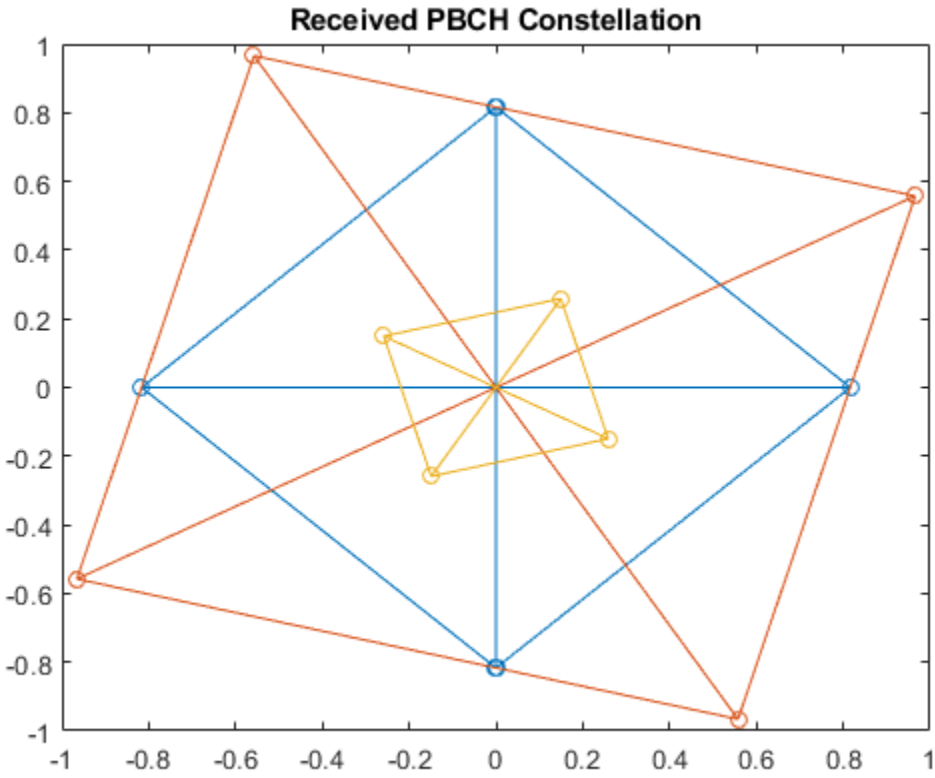
```
hEstGrid = repmat(permute(H.'*F.',[3 4 1 2]),[240 4]);
nEst = 0;
```

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

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

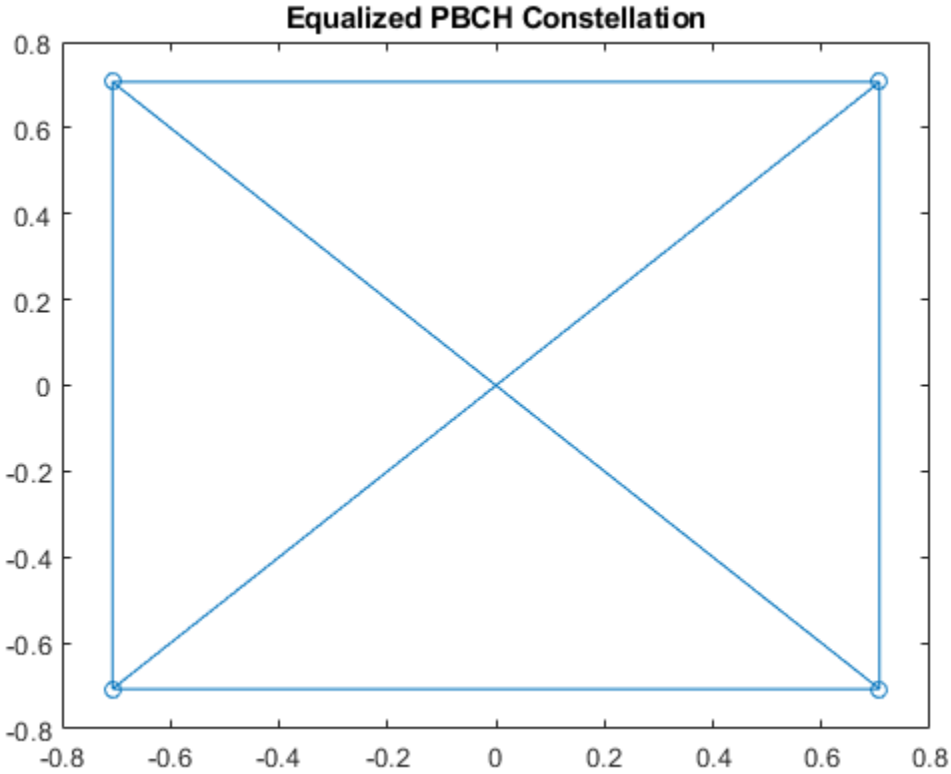
In preparation for PBCH decoding, extract symbols from the received grid and the channel estimate grid.

```
[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 softbits by performing PBCH decoding on the equalized symbols.

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

```
pbchBits = 864×1
10^10 ×
```

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

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 1-179.
- `'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 1-180.

For more details on the resource extraction process, see “Algorithms” on page 1-178.

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.

`reind` — Indices of extracted resource elements

numeric array

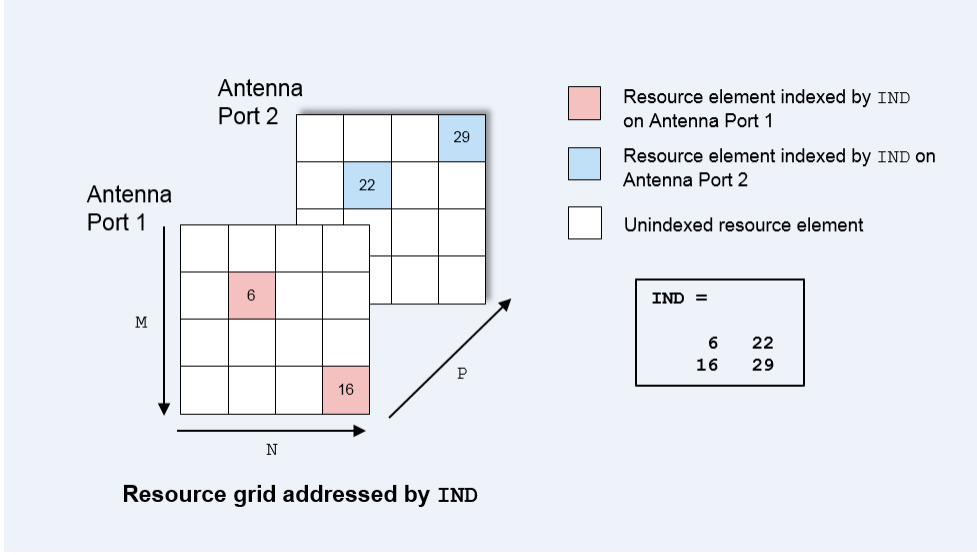
Indices of extracted resource elements within `grid`, returned as numeric array. `reind` is the same size as the extracted resource elements array `re`. The `reind` output inherits the indexing style and index base from `ind`.

Algorithms

Resource Element Indices

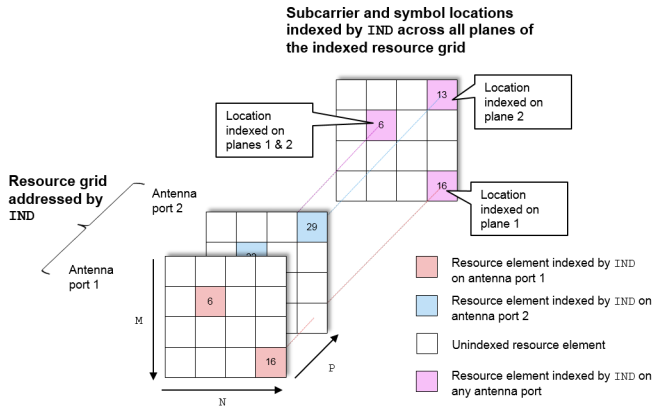
Typically, channel or signal specific functions generate resource element indices to map the channel or signal symbols to a resource grid. The indices address resource elements in an *M*-by-*N*-by-*P* array. *M* is the number of subcarriers, *N* is the number of OFDM symbols, and *P* is the number of 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 ($P = 2$).

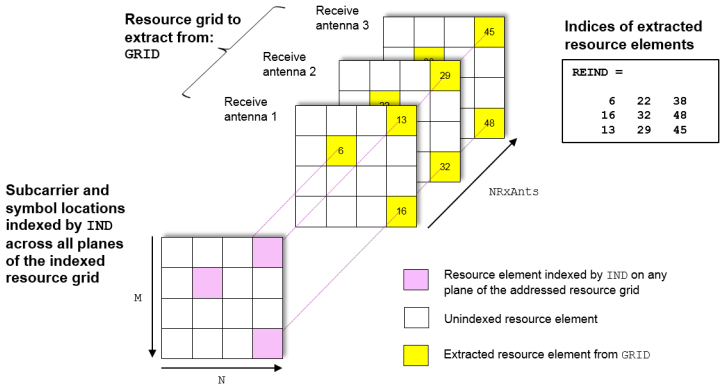


All-Planes Extraction Method (Default)

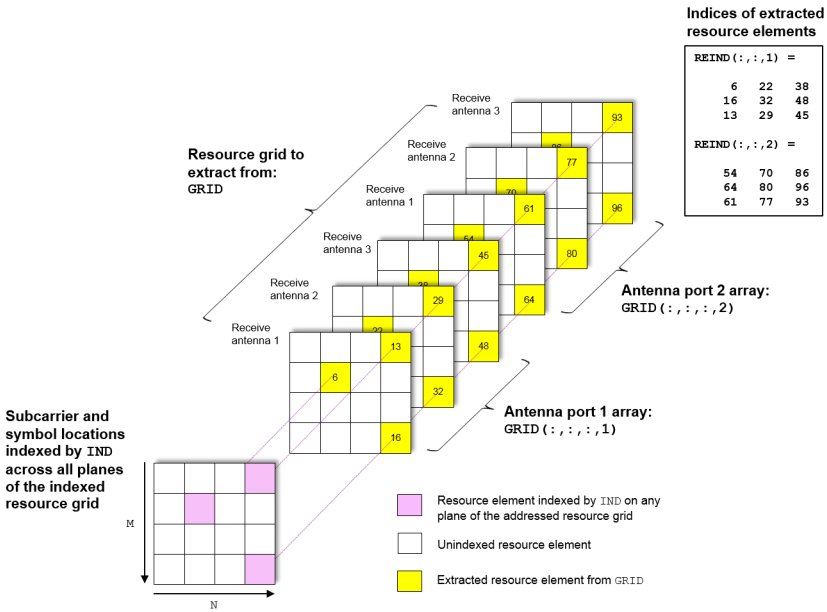
To use this method, set `'ExtractionMethod'` to `'allplanes'`. This method extracts resource elements from each *M*-by-*N* plane within `grid`. The indices address unique subcarrier and symbol locations over all the planes of the indexed resource array. The diagram highlights the indices used to extract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



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.



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.



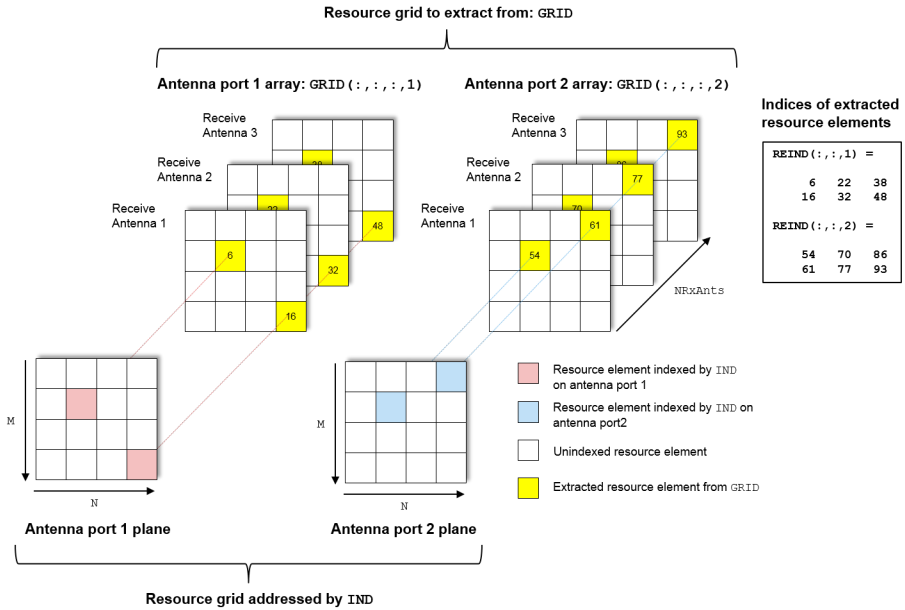
Direct Extraction Method

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`.

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™.

See Also

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

Introduced in R2018b

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

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.

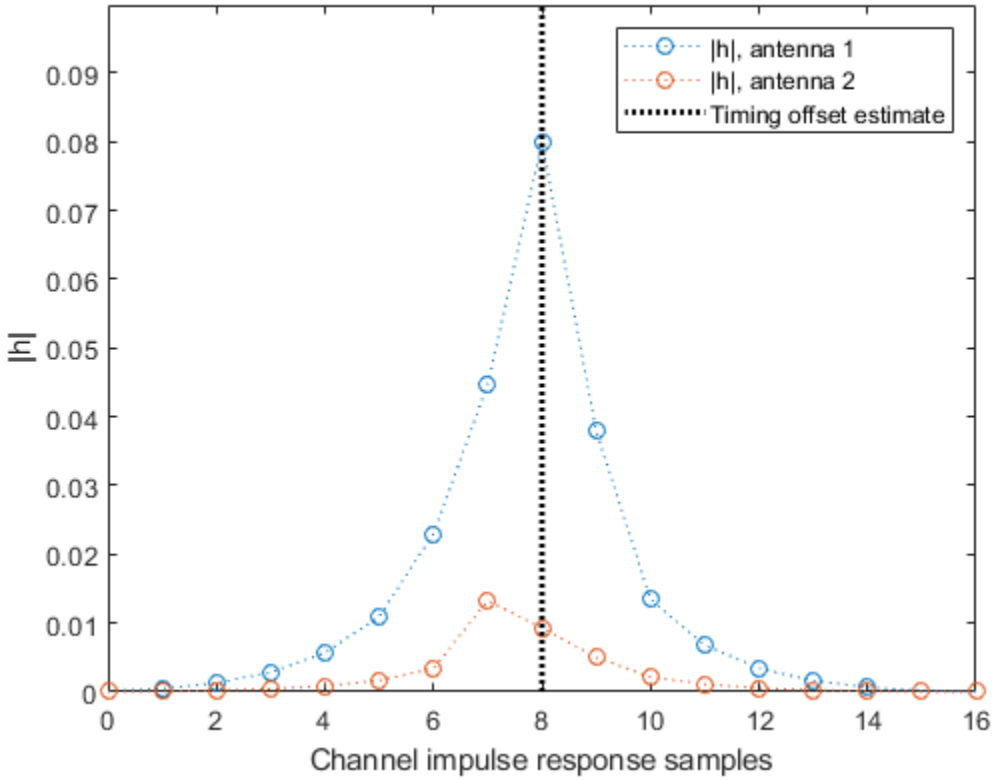
```
pathFilters = getPathFilters(cdl);
```

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

System Objects — Alphabetical List

nrCDLChannel System object

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 $[\alpha; \beta; \gamma]$. 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);
```

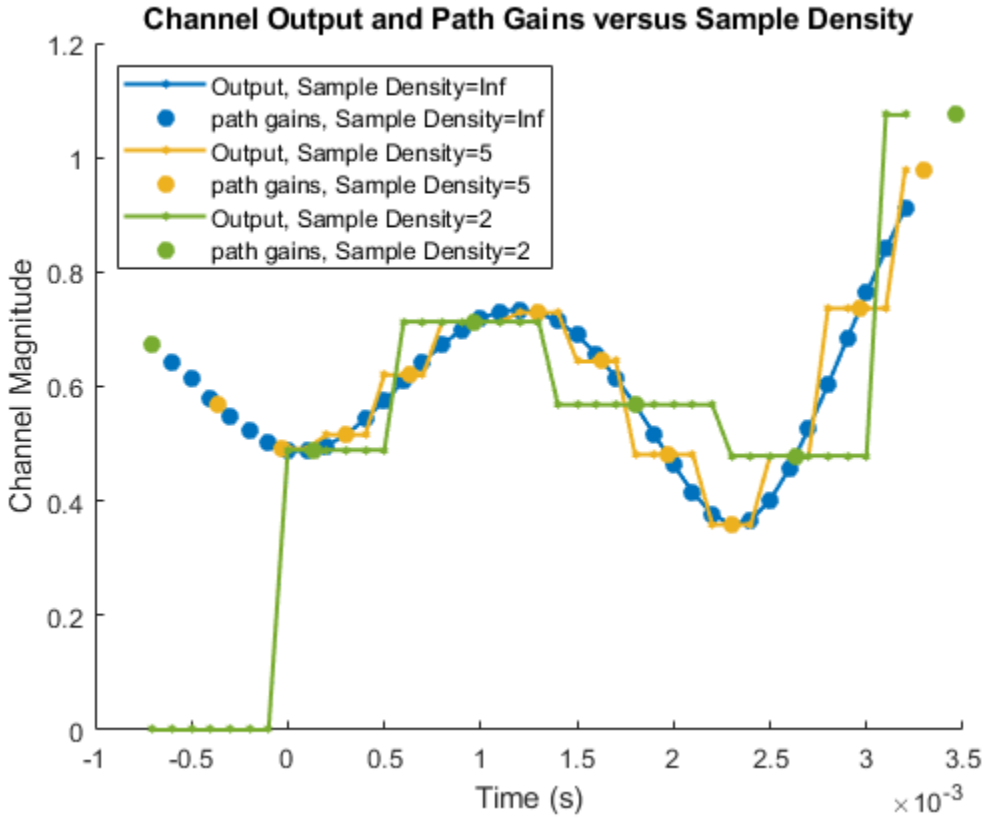
Plot the step response of the channel (displayed as lines) and the corresponding path gain snapshots (displayed circles) for various values of the `SampleDensity` property. The sample density property controls how often the channel snapshots are taken relative to the Doppler frequency.

- 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`

System Objects

`comm.MIMOChannel` | `nrTDLChannel`

Introduced in R2018b

nrTDLChannel System object

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 × N_R)-by-(N_T × N_R).The magnitude of any off-diagonal element must be no larger than the geometric mean of the two corresponding diagonal elements.
- If the channel is frequency-selective (`PathDelays` is a row vector of length N_P), specify `SpatialCorrelationMatrix` as one of these arrays:
  - 2-D Hermitian matrix of size (N_T × N_R)-by-(N_T × N_R) with off-diagonal element properties as previously described. Each path has the same spatial correlation matrix.
  - 3-D array of size (N_T × N_R)-by-(N_T × N_R)-by-N_P array — where each matrix of size (N_T × N_R)-by-(N_T × N_R) is a Hermitian matrix with off-diagonal element properties as previously described. Each path has its own spatial correlation matrix.

Dependencies

To enable this property, set `MIMOCorrelation` to `'Custom'` and `Polarization` to `'Custom'`.

Data Types: `double`

`NormalizePathGains` — Normalize path gains

`true` (default) | `false`

Normalize path gains, specified as `true` or `false`. Use this property to normalize the fading processes. When this property is set to `true`, the total power of the path gains, averaged over time, is 0 dB. When this property is set to `false`, the path gains are not normalized. The average powers of the path gains are specified by the selected delay profile, or if `DelayProfile` is set to `'Custom'`, by the `AveragePathGains` property.

Data Types: `logical`

`InitialTime` — Time offset of fading process in seconds

`0.0` (default) | numeric scalar

Time offset of fading process in seconds, specified as a numeric scalar.

Tunable: Yes

Data Types: `double`

`NumSinusoids` — Number of modeling sinusoids

`48` (default) | positive integer

Number of modeling sinusoids, specified as a positive integer. These sinusoids model the fading process.

Data Types: `double`

`RandomStream` — Source of random number stream

`'mt19937ar with seed'` (default) | `'Global stream'`

Source of random number stream, specified as one of the following:

- `'mt19937ar with seed'` — The object uses the mt19937ar algorithm for normally distributed random number generation. Calling the `reset` function resets the filters and reinitializes the random number stream to the value of the `Seed` property.
- `'Global stream'` — The object uses the current global random number stream for normally distributed random number generation. Calling the `reset` function resets only the filters.

`Seed` — Initial seed of mt19937ar random number stream

`73` (default) | nonnegative numeric scalar

Initial seed of mt19937ar random number stream, specified as a nonnegative numeric scalar.

Dependencies

To enable this property, set RandomStream to `'mt19937ar with seed'`. When calling the `reset` function, the seed reinitializes the mt19937ar random number stream.

Data Types: `double`

`NormalizeChannelOutputs` — Normalize channel outputs by the number of receive antennas
`true` (default) | `false`

Normalize channel outputs by the number of receive antennas, specified as `true` or `false`.

Data Types: `logical`

Usage

Syntax

```
signalOut = tdl(signalIn)
[signalOut,pathGains] = tdl(signalIn)
[signalOut,pathGains,sampleTimes] = tdl(signalIn)
```

Description

`signalOut = tdl(signalIn)` sends the input signal through a TDL MIMO fading channel and returns the channel-impaired signal.

`[signalOut,pathGains] = tdl(signalIn)` also returns the MIMO channel path gains of the underlying fading process.

`[signalOut,pathGains,sampleTimes] = tdl(signalIn)` also returns the sample times of the channel snapshots of the path gains.

Input Arguments

`signalIn` — Input signal
complex scalar | vector | N_S-by-N_T matrix

Input signal, specified as a complex scalar, vector, or N_S-by-N_T matrix, where:

- N_S is the number of samples.
- N_T is the number of transmit antennas.

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

Output Arguments

`signalOut` — Output signal

complex scalar | vector | N_S-by-N_R matrix

Output signal, returned as a complex scalar, vector, or N_S-by-N_R matrix, where:

- N_S is the number of samples.
- N_R is the number of receive antennas.

The output signal data type is of the same precision as the input signal data type.

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

`pathGains` — MIMO channel path gains of fading process

N_S-by-N_P-by-N_T-by-N_R complex matrix

MIMO channel path gains of the fading process, returned as an N_S-by-N_P-by-N_T-by-N_R complex matrix, where:

- N_S is the number of samples.
- N_P is the number of paths, specified by the length of the `PathDelays` property of `tdl`.
- N_T is the number of transmit antennas.
- N_R is the number of receive antennas.

The path gains data type is of the same precision as the input signal data type.

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

`sampleTimes` — Sample times of channel snapshots

N_S-by-1 column vector of real numbers

Sample times of the channel snapshots of the path gains, returned as an N_S-by-1 column vector of real numbers. N_S is the first dimension of `pathGains` that corresponds to the number of samples.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to `nrTDLChannel`

info	Get characteristic information about link-level MIMO fading channel
getPathFilters	Get path filter impulse response for link-level MIMO fading channel

Common to All System Objects

step	Run System object algorithm
clone	Create duplicate System object
isLocked	Determine if System object is in use
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Transmission over MIMO Channel Model with Delay Profile TDL

Display 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.

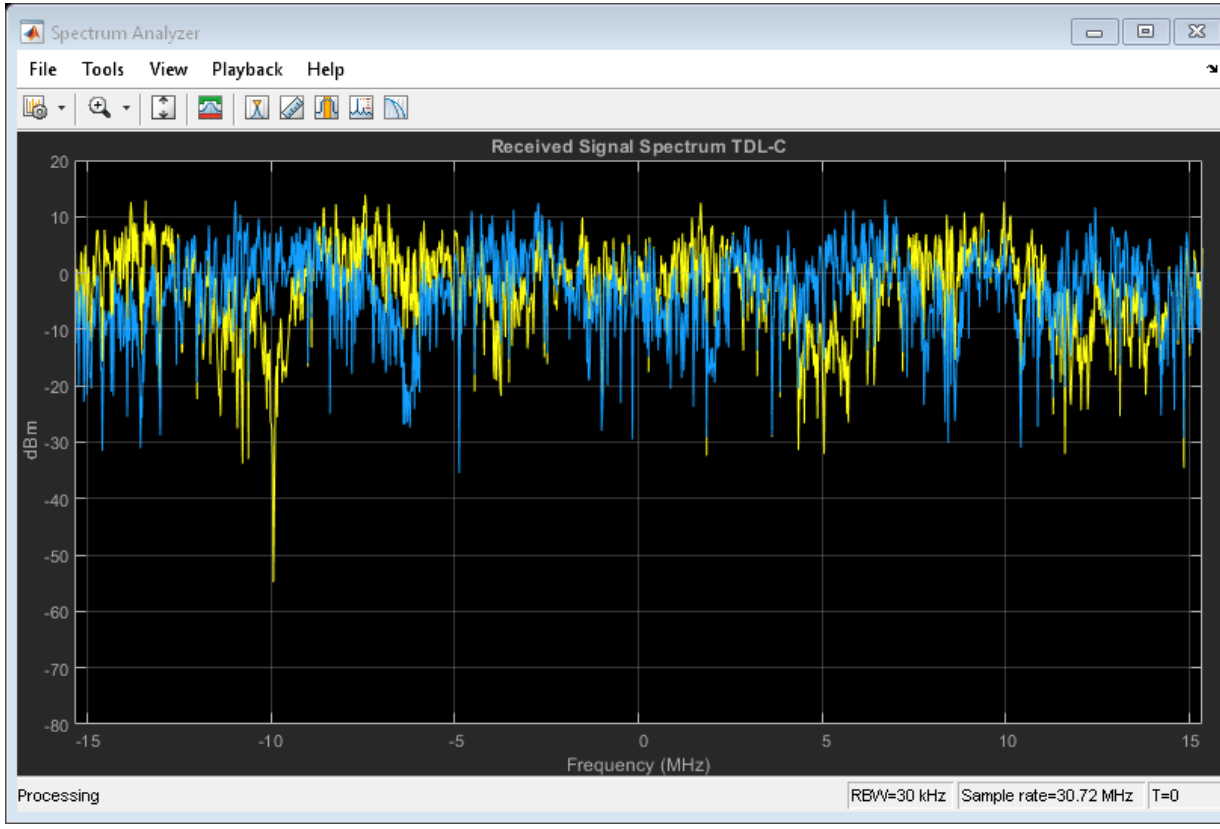
```
SR = 30.72e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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.

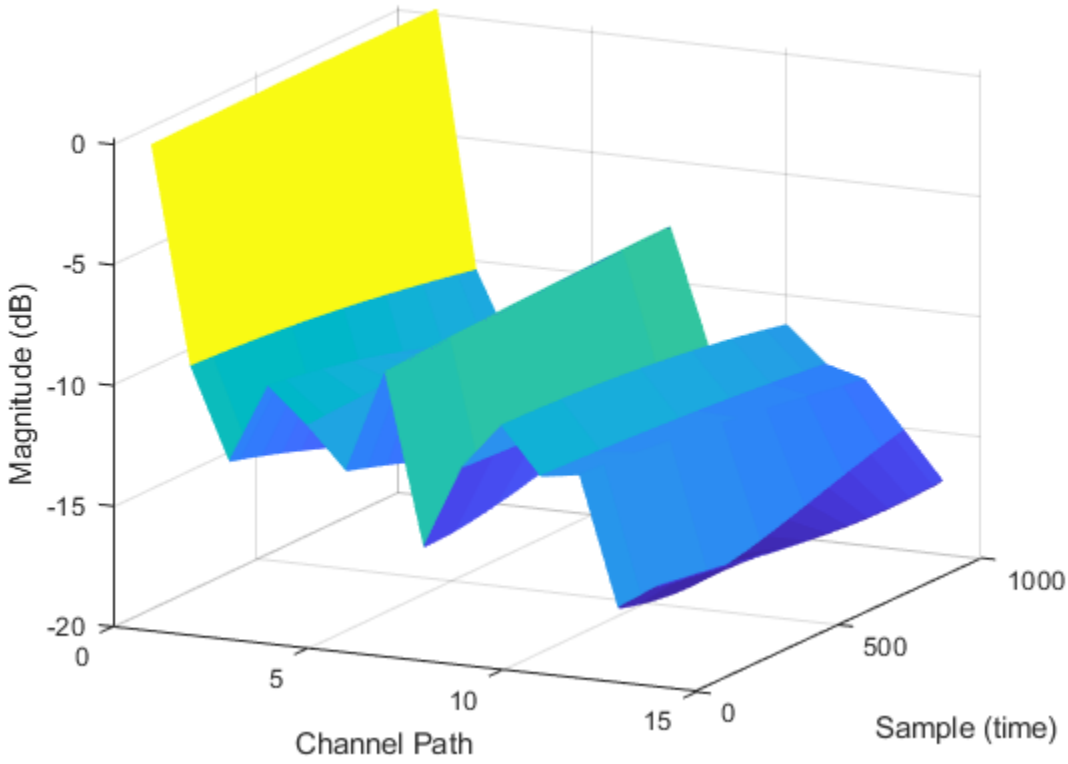
```
tdl.NumTransmitAntennas = 1;
tdl.NumReceiveAntennas = 1;
```

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.

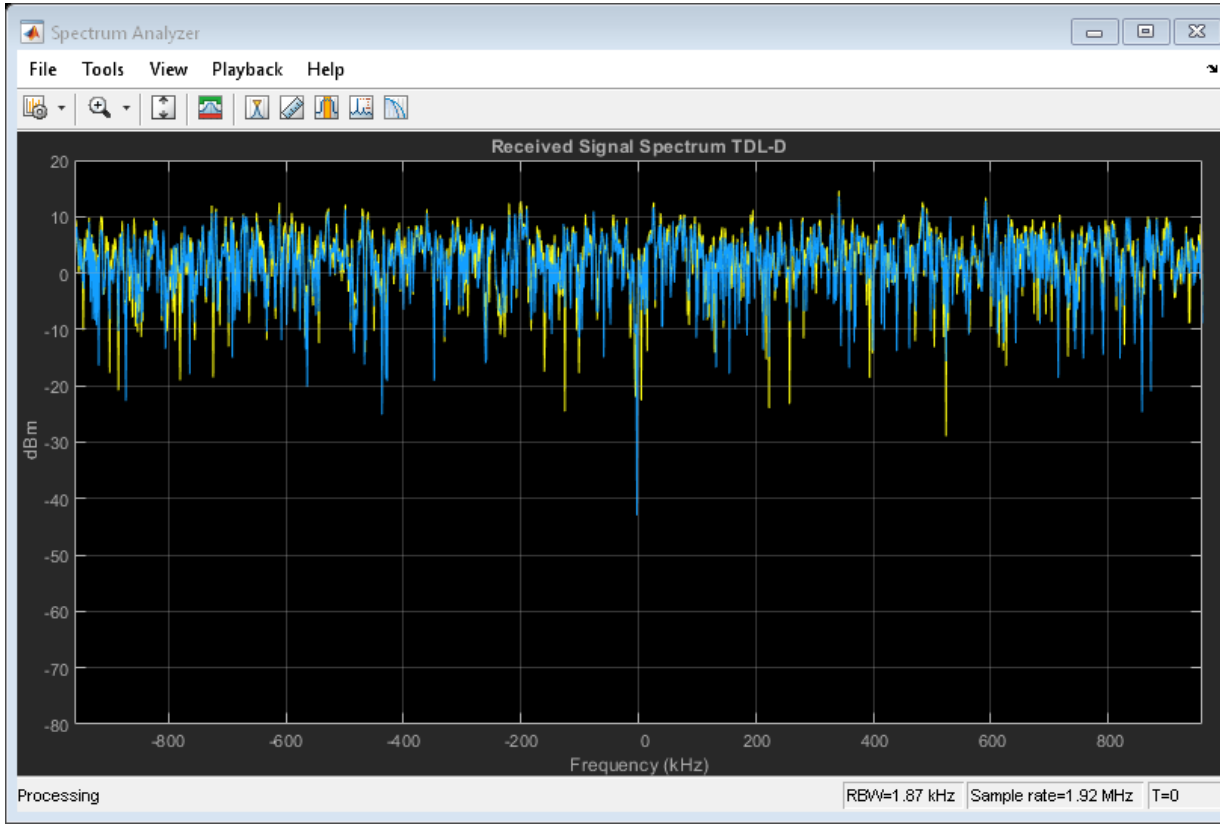
```
SR = 1.92e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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`

System Objects
`comm.MIMOChannel` | `nrCDLChannel`

Introduced in R2018b