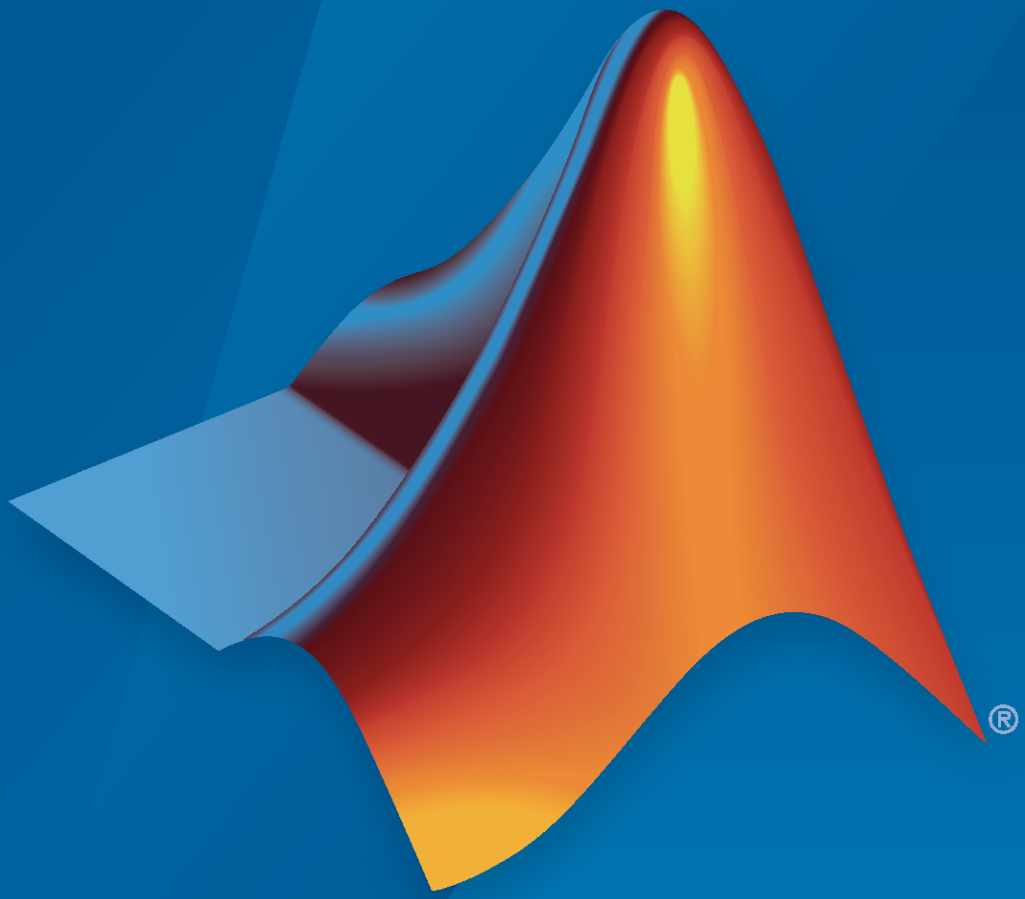
5G Toolbox™

User's Guide



MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

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

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

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

Phone: 508-647-7000

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

5G Toolbox™ User's Guide

Revision History

September 2018	Online only	New for Version 1.0 (Release 2018b)
March 2019	Online only	Revised for Version 1.1 (Release 2019a)
September 2019	Online only	Revised for Version 1.2 (Release 2019b)
March 2020	Online only	Revised for Version 2.0 (Release 2020a)
September 2020	Online only	Revised for Version 2.1 (Release 2020b)
March 2021	Online only	Revised for Version 2.2 (Release 2021a)
September 2021	Online only	Revised for Version 2.3 (Release 2021b)
March 2022	Online only	Revised for Version 2.4 (Release 2022a)
September 2022	Online only	Revised for Version 2.5 (Release 2022b)
March 2023	Online only	Revised for Version 2.6 (Release 2023a)

Contents

1 Downlink Channels

2 Uplink Channels

3 Physical Layer Subcomponents

4 Signal Reception

5 End-To-End Simulation

6 System-Level Simulation

7 Test and Measurement

8 Code Generation and Deployment

Downlink Channels

5G NR Downlink Vector Waveform Generation

This example shows how to configure and generate a 5G NR downlink vector waveform for a baseband component carrier by using the `nrWaveformGenerator` function.

Introduction

This example shows how to parameterize and generate a 5G new radio (NR) downlink waveform by using the `nrWaveformGenerator` function. The generated waveform contains these channels and signals.

- PDSCH and its associated DM-RS and PT-RS
- PDCCH and its associated DM-RS
- PBCH and its associated DM-RS
- PSS and SSS
- CSI-RS

This example demonstrates how to parameterize and generate a baseband component carrier waveform characterized by multiple subcarrier spacing (SCS) carriers and bandwidth parts (BWP). You can generate multiple instances of the physical downlink shared channel (PDSCH), the physical downlink control channel (PDCCH), and the channel state information reference signal (CSI-RS) over the different BWPs. You can configure sets of control resource sets (CORESETs) and search space monitoring opportunities for mapping the PDCCHs. This example does not apply precoding to the physical channels and signals.

Waveform and Carrier Configuration

Use the `nrDLCarrierConfig` object to parameterize the baseband waveform generation. This object contains a set of additional objects associated with the waveform channels and signals and enables you to set these downlink carrier configuration parameters.

- Label for this DL carrier configuration
- SCS carrier bandwidth in resource blocks
- Carrier cell ID
- Length of the generated waveform in subframes
- Windowing
- Sample rate of the OFDM-modulated waveform
- Carrier frequency for symbol phase compensation

You can control SCS carrier bandwidths and guardbands using the `NStartGrid` and `NSizeGrid` properties of the `nrSCSCarrierConfig` object.

```
waveconfig = nrDLCarrierConfig;    % Create a downlink carrier configuration object
waveconfig.Label = 'DL carrier 1'; % Label for this downlink waveform configuration
waveconfig.NCellID = 0;            % Cell identity
waveconfig.ChannelBandwidth = 40;  % Channel bandwidth (MHz)
waveconfig.FrequencyRange = 'FR1'; % 'FR1' or 'FR2'
waveconfig.NumSubframes = 10;      % Number of 1 ms subframes in generated waveform (1, 2, 4, 8 s
waveconfig.WindowingPercent = 0;   % Percentage of windowing relative to FFT length
waveconfig.SampleRate = [];        % Sample rate of the OFDM modulated waveform
```

```
waveconfig.CarrierFrequency = 0;   % Carrier frequency in Hz. This property is used for symbol ph
                                   % compensation before OFDM modulation

% Define a set of SCS specific carriers, using the maximum sizes for a
% 40 MHz NR channel. See TS 38.101-1 for more information on defined
% bandwidths and guardband requirements
scscarriers = {nrSCSCarrierConfig,nrSCSCarrierConfig};
scscarriers{1}.SubcarrierSpacing = 15;
scscarriers{1}.NSizeGrid = 216;
scscarriers{1}.NStartGrid = 0;

scscarriers{2}.SubcarrierSpacing = 30;
scscarriers{2}.NSizeGrid = 106;
scscarriers{2}.NStartGrid = 1;
```

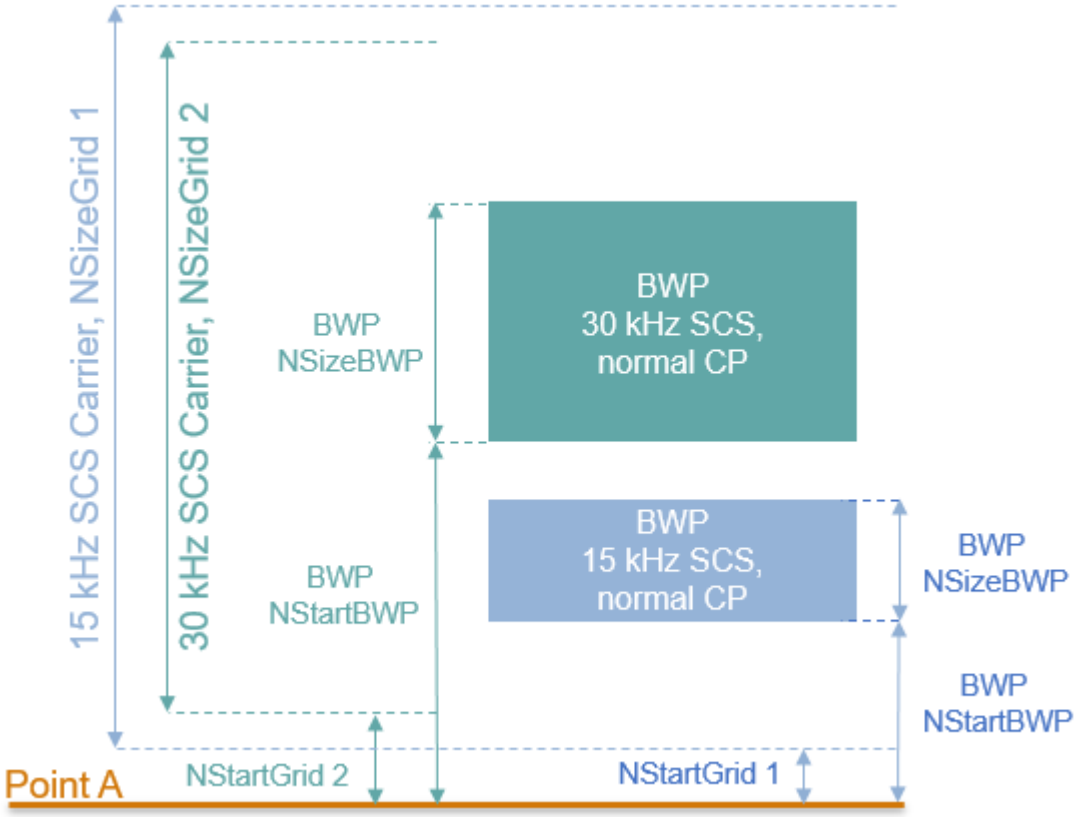
SS Burst

In this section you can set the parameters for the signal synchronization (SS) burst. The numerology of the SS burst can be different from other parts of the waveform. This is specified via the block pattern parameter, as specified in TS 38.213 Section 4.1. A bitmap specifies the blocks to transmit in a 5 ms half-frame burst. You can also set the periodicity in milliseconds and the power of the burst. For a full list of configurable SS burst properties, see `nrWavegenSSBurstConfig`.

```
% SS burst configuration
ssburst = nrWavegenSSBurstConfig;
ssburst.Enable = 1;                     % Enable SS Burst
ssburst.Power = 0;                      % Power scaling in dB
ssburst.BlockPattern = 'Case B';        % Case B (30kHz) subcarrier spacing
ssburst.TransmittedBlocks = [1 1 1 1];  % Bitmap indicating blocks transmitted in a 5ms half-fram
ssburst.Period = 20;                    % SS burst set periodicity in ms (5, 10, 20, 40, 80, 160)
ssburst.NCRBSSB = [];                   % Frequency offset of SS burst (CRB), use [] for the wave
```

BWPs

A BWP is formed by a set of contiguous resources sharing a numerology on a given SCS carrier. You can define multiple BWPs using a cell array. Each element in the cell array of `nrWavegenBWPConfig` objects defines a BWP. For each BWP, you can specify the SCS, the cyclic prefix (CP) length, and the bandwidth. The `SubcarrierSpacing` property links the BWP to one of the SCS specific carriers defined earlier. The `NStartBWP` property controls the location of the BWP in the carrier, relative to point A. `NStartBWP` is expressed in common resource blocks (CRB) in terms of the BWP numerology. Different BWPs can overlap with each other.




```
% BWP configurations
bwp = {nrWavegenBWPConfig,nrWavegenBWPConfig};
bwp{1}.BandwidthPartID = 1;         % BWP ID
bwp{1}.Label = 'BWP 1 @ 15 kHz';    % Label for this BWP
bwp{1}.SubcarrierSpacing = 15;      % BWP subcarrier spacing
bwp{1}.CyclicPrefix = 'Normal';     % BWP cyclic prefix for 15 kHz
bwp{1}.NSizeBWP = 25;               % Size of BWP in PRBs
bwp{1}.NStartBWP = 12;              % Position of BWP, relative to point A, in CRBs

bwp{2}.BandwidthPartID = 2;         % BWP ID
bwp{2}.Label = 'BWP 2 @ 30 kHz';    % Label for this BWP
bwp{2}.SubcarrierSpacing = 30;      % BWP subcarrier spacing
bwp{2}.CyclicPrefix = 'Normal';     % BWP cyclic prefix for 30 kHz
bwp{2}.NSizeBWP = 50;               % Size of BWP in PRBs
bwp{2}.NStartBWP = 51;              % Position of BWP, relative to point A, in CRBs
```

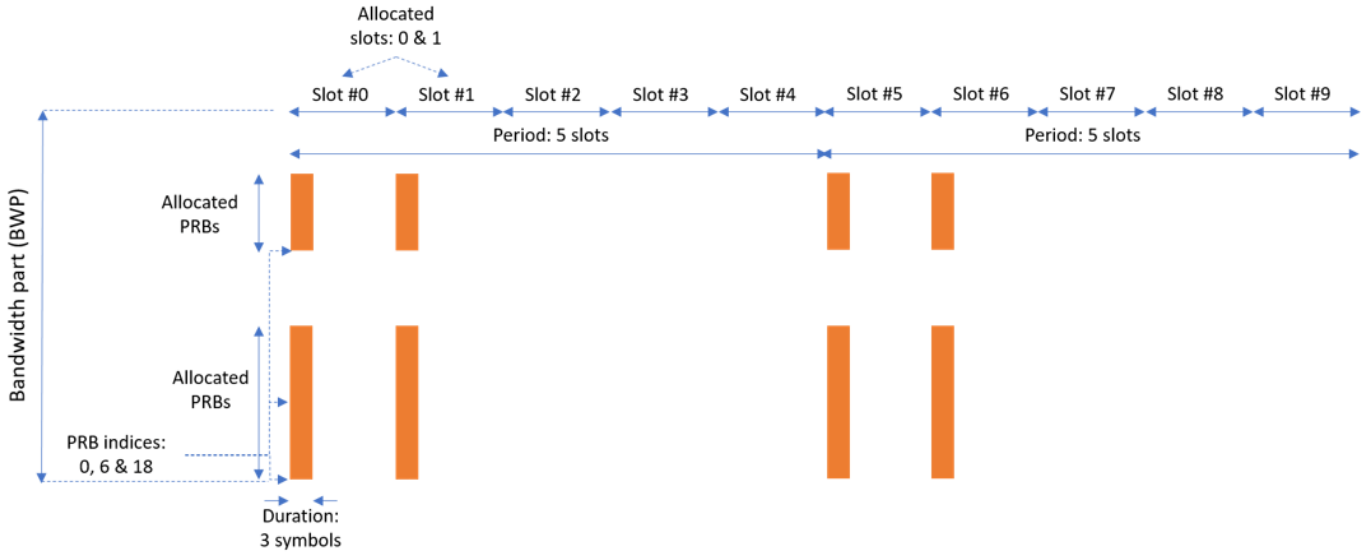
CORESET and Search Space Configuration

Specify the CORESET and the PDCCH search space configuration. The CORESET and search spaces specify the possible locations (in time and frequency) of the control channel transmissions for a given numerology. Each element in the cell array of `nrCORESETConfig` objects defines a CORESET and each element in the cell array of `nrSearchSpaceConfig` objects defines a search space.

Set these parameters for each CORESET and search space.

- The OFDM symbols which specify the first symbol of each CORESET monitoring opportunity in a slot.
- The duration of the block of allocated slots within a period.
- Periodicity of the allocation pattern.
- The CORESET duration in symbols, either 1, 2 or 3.
- A bitmap defining the allocated physical resource blocks (PRB) of the CORESET. The CORESET frequency allocation is defined in blocks of 6 PRBs, aligned in CRB numbering, relative to point A. Each bit in the bitmap selects all 6 PRBs in the CRB aligned block that contains it.
- CCE-to-REG mapping which can be 'interleaved' or 'noninterleaved'.
- Resource element group (REG) bundle size (L), either (2,6) or (3,6), based on CORESET duration.
- Interleaver size, either 2, 3, or 6.
- Shift index, a scalar value in the range 0...274.

The figure below shows the meaning of some of the CORESET parameters.




```
% CORESET and search space configurations
coresets = {nrCORESETConfig};
coresets{1}.CORESETID = 1;                          % CORESET ID
coresets{1}.Duration = 3;                           % CORESET symbol duration (1,2,3)
coresets{1}.FrequencyResources = [1 1 0 1];         % Bitmap indicating blocks of 6 PRB for CORESET
coresets{1}.CCEREGMapping = 'noninterleaved';       % Mapping: 'interleaved' or 'noninterleaved'
coresets{1}.REGBundleSize = 3;                      % L (2,6) or (3,6)
coresets{1}.InterleaverSize = 2;                    % R (2,3,6)
coresets{1}.ShiftIndex = waveconfig.NCellID;        % Set to NCellID

searchspaces = {nrSearchSpaceConfig};
searchspaces{1}.SearchSpaceID = 1;                  % Search space ID
searchspaces{1}.CORESETID = 1;                      % CORESET associated with this search space
searchspaces{1}.SearchSpaceType = 'ue';             % Search space type, 'ue' or 'common'
searchspaces{1}.SlotPeriodAndOffset = [5 0];        % Allocated slot period and slot offset of search
searchspaces{1}.Duration = 2;                       % Number of slots in the block of slots in patte
searchspaces{1}.StartSymbolWithinSlot = 0;          % First symbol of each CORESET monitoring opportu
searchspaces{1}.NumCandidates = [8 8 4 2 0];        % Number of candidates at each AL (set to 0 if th
```

PDCCH Instances Configuration

Specify the set of PDCCH transmission instances in the waveform by using a cell array. Each element in the cell array of `nrWavegenPDCCHConfig` objects defines a sequence of PDCCH instances.

Set these parameters for each PDCCH sequence.

- Enable or disable this PDCCH sequence.
- Specify a label for this PDCCH sequence.
- Specify the BWP carrying the PDCCH. The PDCCH uses the SCS specified for this BWP.
- Power scaling in dB.
- Enable or disable downlink control information (DCI) channel coding.
- Allocated search spaces within the CORESET monitoring occasion sequence.
- Search space (and CORESET) that carries the PDCCH instances.
- Period of the allocation in slots. Empty period indicates no repetition of the slot pattern.
- The aggregation level (AL) of the PDCCH (number of control channel elements (CCEs)).
- The allocated candidate which specifies the CCE used for the transmission of the PDCCH.
- RNTI.
- Scrambling NID for this PDCCH and its associated DM-RS.
- DM-RS power boosting in dB.
- DCI message payload size.
- DCI message data source. You can use an array of bits or one of these standard PN sequences: `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, `'PN23'`. You can specify the seed for the generator as a cell array in the form `{'PN9', seed}`. If you do not specify a seed, the generator is initialized with all ones.

```
pdcch = {nrWavegenPDCCHConfig};
pdcch{1}.Enable = 1 ;                       % Enable PDCCH sequence
pdcch{1}.Label = 'UE 1 - PDCCH @ 15 kHz'; % Label for this PDCCH sequence
pdcch{1}.BandwidthPartID = 1;               % Bandwidth part of PDCCH transmission
pdcch{1}.Power = 1.1;                       % Power scaling in dB
pdcch{1}.Coding = 1;                        % Enable DCI coding
pdcch{1}.SearchSpaceID = 1;                 % Search space
pdcch{1}.SlotAllocation = 0;                % Allocated slots indices for PDCCH sequence
pdcch{1}.Period = 5;                        % Allocation period in slots
pdcch{1}.AggregationLevel = 8;              % Aggregation level (1,2,4,8,16 CCEs)
pdcch{1}.AllocatedCandidate = 1;            % PDCCH candidate in search space (1 based)
pdcch{1}.RNTI = 11;                         % RNTI
pdcch{1}.DMRSScramblingID = 1;              % PDCCH and DM-RS scrambling NID
pdcch{1}.DMRSPower = 0;                     % Additional DM-RS power boosting in dB
pdcch{1}.DataBlockSize = 20;                % DCI payload size
pdcch{1}.DataSource = 'PN9';                % DCI data source
```

PDSCH Instances Configuration

Specify the set of PDSCH transmission instances in the waveform by using a cell array. Each element in the cell array of `nrWavegenPDSCHConfig` objects defines a sequence of PDSCH instances. This example defines two PDSCH sequences that model two user equipment (UE) transmissions.

General Parameters

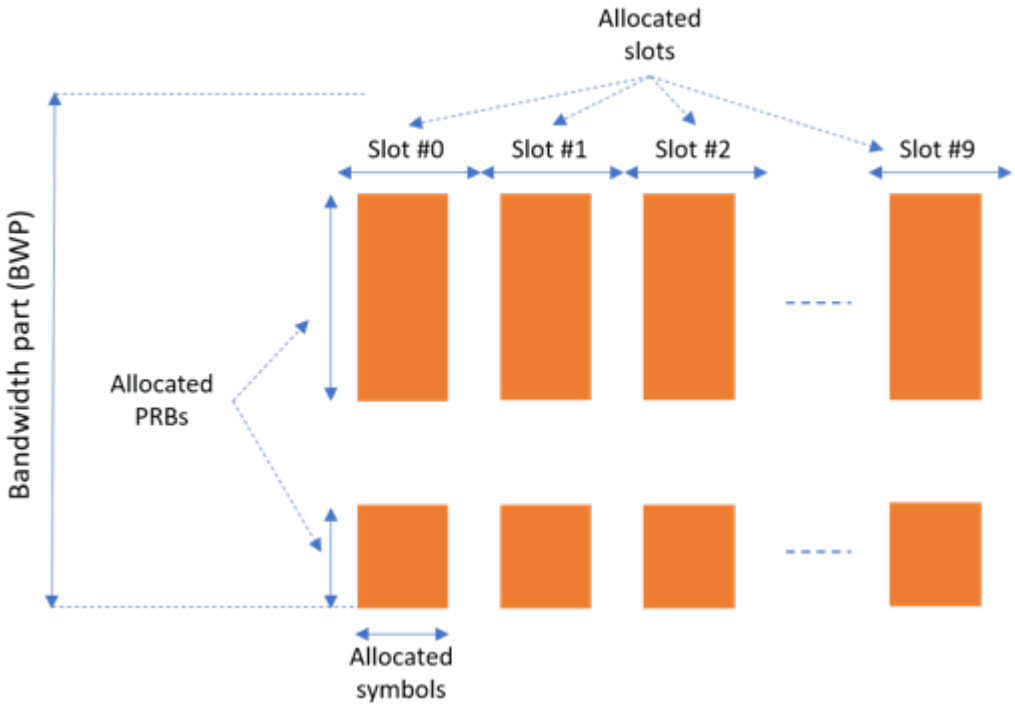
Set these parameters for each PDSCH sequence.

- Enable or disable this PDSCH sequence.
- Specify a label for this PDSCH sequence.
- Specify the BWP carrying the PDSCH. The PDSCH uses the SCS specified for this BWP.
- Power scaling in dB.
- Enable or disable the DL-SCH transport channel coding.
- Transport block data source. You can use an array of bits or one of these standard PN sequences: `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, `'PN23'`. You can specify the seed for the generator as a cell array in the form `{'PN9', seed}`. If you do not specify a seed, the generator is initialized with all ones.
- Target code rate used to calculate the transport block sizes.
- Overhead parameter.
- Symbol modulation.
- Number of layers.
- Redundancy version (RV) sequence.
- Enable or disable the interleaving of the virtual to physical resource block mapping.
- Bundle size for the interleaved map, specified by the higher layer parameter vrb-ToPRB-Interleaver.

```
pdsch = {nrWavegenPDSCHConfig};           % Create a PDSCH configuration object for the first UE
pdsch{1}.Enable = 1;                      % Enable PDSCH sequence
pdsch{1}.Label = 'UE 1 - PDSCH @ 15 kHz'; % Label for this PDSCH sequence
pdsch{1}.BandwidthPartID = 1;             % Bandwidth part of PDSCH transmission
pdsch{1}.Power  = 0;                      % Power scaling in dB
pdsch{1}.Coding = 1;                      % Enable the DL-SCH transport channel coding
pdsch{1}.DataSource = 'PN9';              % Channel data source
pdsch{1}.TargetCodeRate = 0.4785;         % Code rate used to calculate transport block sizes
pdsch{1}.XOverhead = 0;                   % Rate matching overhead
pdsch{1}.Modulation = 'QPSK';             % 'QPSK', '16QAM', '64QAM', '256QAM'
pdsch{1}.NumLayers = 2;                   % Number of PDSCH layers
pdsch{1}.RVSequence = [0 2 3 1];          % RV sequence to be applied cyclically across the PDSCH
pdsch{1}.VRBToPRBInterleaving = 0;        % Disable interleaved resource mapping
pdsch{1}.VRBBundleSize = 2;               % vrb-ToPRB-Interleaver parameter
```

Allocation

This figure shows the parameters of the PDSCH allocation.



You can set these parameters to control the PDSCH allocation. These parameters are relative to the BWP. The specified PDSCH allocation will avoid the locations used for the SS burst.

- Symbols in a slot allocated to each PDSCH instance.
- Slots in a frame used for the sequence of PDSCH.
- Period of the allocation in slots. Empty period indicates no repetition of the slot pattern.
- The allocated PRBs relative to the BWP.
- RNTI. This value is used to link the PDSCH to an instance of the PDCCH.
- NID for scrambling the PDSCH bits.

```
pdsch{1}.SymbolAllocation = [2 9];    % First symbol and length
pdsch{1}.SlotAllocation = 0:9;        % Allocated slot indices for PDSCH sequence
pdsch{1}.Period = 15;                 % Allocation period in slots
pdsch{1}.PRBSet = [0:5, 10:20];       % PRB allocation
pdsch{1}.RNTI = 11;                   % RNTI for the first UE
pdsch{1}.NID = 1;                     % Scrambling for data part
```

CORESETs and sets of PRB can be specified for rate matching around, if required

- The PDSCH can be rate matched around one or more CORESETs.
- The PDSCH can be rate matched around other resource allocations.

```
pdsch{1}.ReservedCORESET = 1;           % Rate matching pattern, defined by CORESET IDs
pdsch{1}.ReservedPRB{1}.PRBSet = [];    % Rate matching pattern, defined by set of PRB (RRC 'bitm
pdsch{1}.ReservedPRB{1}.SymbolSet = [];
pdsch{1}.ReservedPRB{1}.Period = [];
```

PDSCH DM-RS Configuration

Set the DM-RS parameters.

```
% Antenna port and DM-RS configuration (TS 38.211 section 7.4.1.1)
pdsch{1}.MappingType = 'A';                % PDSCH mapping type ('A'(slot-wise),'B'(non slot-wise
pdsch{1}.DMRSPower = 0;                    % Additional power boosting in dB

pdsch{1}.DMRS.DMRSConfigurationType = 2;   % DM-RS configuration type (1,2)
pdsch{1}.DMRS.NumCDMGroupsWithoutData = 1; % Number of DM-RS CDM groups without data. The value c
pdsch{1}.DMRS.DMRSPortSet = [];            % DM-RS antenna ports used ([] gives port numbers 0:Nu
pdsch{1}.DMRS.DMRSTypeAPosition = 2;       % Mapping type A only. First DM-RS symbol position (2
pdsch{1}.DMRS.DMRSLength = 1;              % Number of front-loaded DM-RS symbols (1(single symbo
pdsch{1}.DMRS.DMRSAdditionalPosition = 0;  % Additional DM-RS symbol positions (max range 0...3)
pdsch{1}.DMRS.NIDNSCID = 1;                % Scrambling identity (0...65535)
pdsch{1}.DMRS.NSCID = 0;                   % Scrambling initialization (0,1)
```

PDSCH PT-RS Configuration

Set the PT-RS parameters.

```
% PT-RS configuration (TS 38.211 section 7.4.1.2)
pdsch{1}.EnablePTRS = 0;             % Enable or disable the PT-RS (1 or 0)
pdsch{1}.PTRSPower = 0;              % Additional PT-RS power boosting in dB

pdsch{1}.PTRS.TimeDensity = 1;       % Time density (L_PT-RS) of PT-RS (1,2,4)
pdsch{1}.PTRS.FrequencyDensity = 2;  % Frequency density (K_PT-RS) of PT-RS (2,4)
pdsch{1}.PTRS.REOffset = '00';       % PT-RS resource element offset ('00','01','10','11')
pdsch{1}.PTRS.PTRSPortSet = 0;       % PT-RS antenna ports must be a subset of DM-RS ports
```

When PT-RS is enabled, the DM-RS ports must be in the range from 0 to 3 for DM-RS configuration type 1, and in the range from 0 to 5 for DM-RS configuration type 2. Nominally, the antenna port of PT-RS is the lowest DM-RS port number.

Specifying Multiple PDSCH Instances

Specify the second PDSCH sequence for the second BWP.

```
pdsch{2} = pdsch{1};                    % Create a PDSCH configuration object for the second UE
pdsch{2}.Enable = 1;
pdsch{2}.Label = 'UE 2 - PDSCH @ 30 kHz';
pdsch{2}.BandwidthPartID = 2;           % PDSCH mapped to the second BWP
pdsch{2}.RNTI = 12;                     % RNTI for the second UE
pdsch{2}.SymbolAllocation = [0 12];
pdsch{2}.SlotAllocation = [2:4, 6:20];
pdsch{2}.PRBSet  = [25:30, 35:38];      % PRB allocation, relative to BWP
```

CSI-RS Instances Configuration

This section configures CSI-RS in the waveform. Each element in the cell array of `nrWavegenCSIRSConfig` objects defines a set of CSI-RS resources associated with a BWP. Define two disabled sets of CSI-RS resources.

General Parameters

Set these parameters for a set of CSI-RS resources.

- Enable or disable this set of CSI-RS resources.
- Specify a label for this set of CSI-RS resources.

- Specify the BWP carrying this set of CSI-RS resources. The CSI-RS resource(s) configuration uses the SCS specified for this BWP.
- Specify the power scaling in dB. Providing a scalar defines the power scaling for a single CSI-RS resource or all configured CSI-RS resources. Providing a vector defines a separate power level for each of the CSI-RS resources.

```
csirs = {nrWavegenCSIRSConfig};
csirs{1}.Enable = 0;
csirs{1}.Label = 'CSI-RS @ 15 kHz';
csirs{1}.BandwidthPartID = 1;
csirs{1}.Power = 3; % Power scaling in dB
```

CSI-RS Configuration

You can configure these parameters for one or more zero-power (ZP) or non-zero-power (NZP) CSI-RS resource configurations.

- Type of CSI-RS resource(s) ('nzp','zp').
- Row number corresponds to CSI-RS resource(s) as defined in TS 38.211 Table 7.4.1.5.3-1 (1...18).
- Frequency density of CSI-RS resource(s). It can be `'one'`, `'three'`, `'dot5even'`, or `'dot5odd'`.
- Subcarrier locations of CSI-RS resource(s) within a resource block (RB)
- Number of RBs allocated to CSI-RS resource(s) (1...275).
- Starting RB index of CSI-RS resource(s) allocation relative to the carrier resource grid (0...274).
- OFDM symbol locations of CSI-RS resource(s) within a slot.
- The period and offset of slots (0-based) of CSI-RS resource(s). This parameter can be a vector or a cell array of vectors. In the latter case, each cell corresponds to an individual CSI-RS resource. In case of a vector, the same set of slots is used for all CSI-RS resources.
- Scrambling identity corresponds to CSI-RS resource(s) for pseudo-random sequence generation (0...1023).

```
csirs{1}.CSIRSType = {'nzp','zp'};
csirs{1}.RowNumber = [3 5];
csirs{1}.Density = {'one','one'};
csirs{1}.SubcarrierLocations = {6 4};
csirs{1}.NumRB = 25;
csirs{1}.RBOffset = 12;
csirs{1}.SymbolLocations = {13 9};
csirs{1}.CSIRSPeriod = {[5 0], [5 0]};
csirs{1}.NID = 5;
```

Specifying Multiple CSI-RS Instances

Specify the second set of CSI-RS resources for the second BWP.

```
csirs{2} = nrWavegenCSIRSConfig;
csirs{2}.Enable = 0;
csirs{2}.Label = 'CSI-RS @ 30 kHz';
csirs{2}.BandwidthPartID = 2;
csirs{2}.Power = 3; % Power scaling in dB
csirs{2}.CSIRSType = {'nzp','nzp'};
csirs{2}.RowNumber = [1 1];
csirs{2}.Density = {'three','three'};
```

```
csirs{2}.SubcarrierLocations = {0 0};
csirs{2}.NumRB = 50;
csirs{2}.RBOffset = 50;
csirs{2}.SymbolLocations = {6 10};
csirs{2}.CSIRSPeriod = {[10 1], [10 1]};
csirs{2}.NID = 0;
```

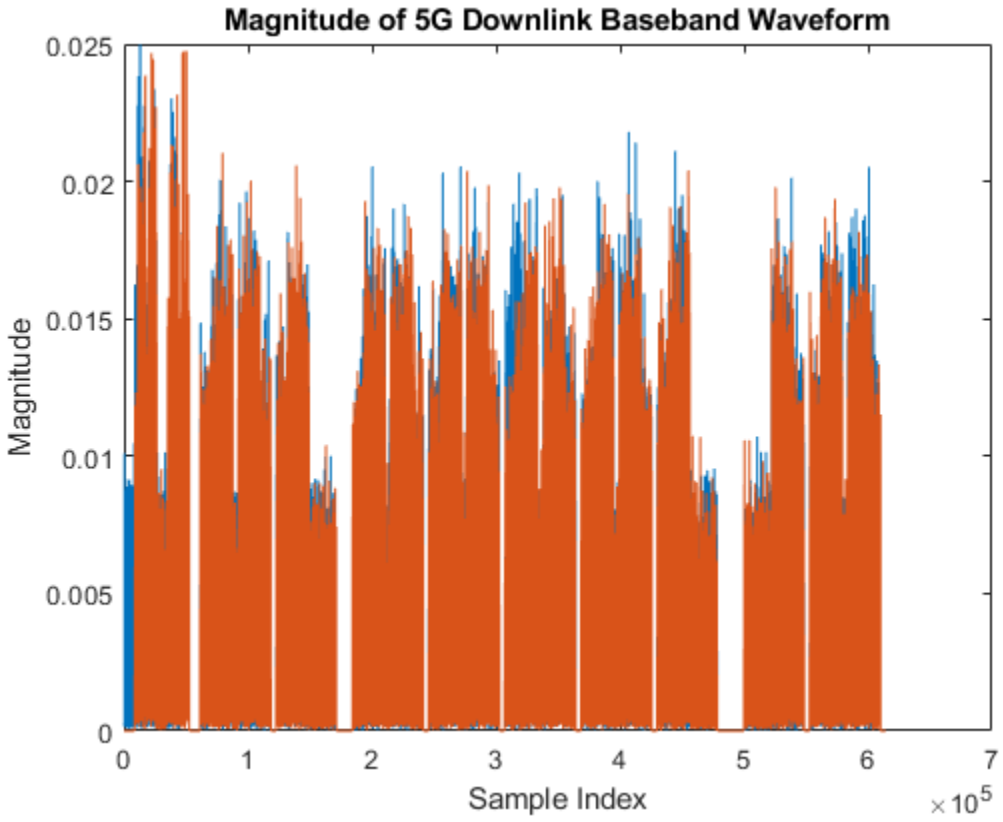
Waveform Generation

Assign all the channel and signal parameters into the main carrier configuration object `nrDLCarrierConfig`, then generates and plot the waveform.

```
waveconfig.SSBurst = ssburst;
waveconfig.SCSCarriers = scscarriers;
waveconfig.BandwidthParts = bwp;
waveconfig.CORESET = coresets;
waveconfig.SearchSpaces = searchspaces;
waveconfig.PDCCH = pdcch;
waveconfig.PDSCH = pdsch;
waveconfig.CSIRS = csirs;

% Generate complex baseband waveform
[waveform,info] = nrWaveformGenerator(waveconfig);
```

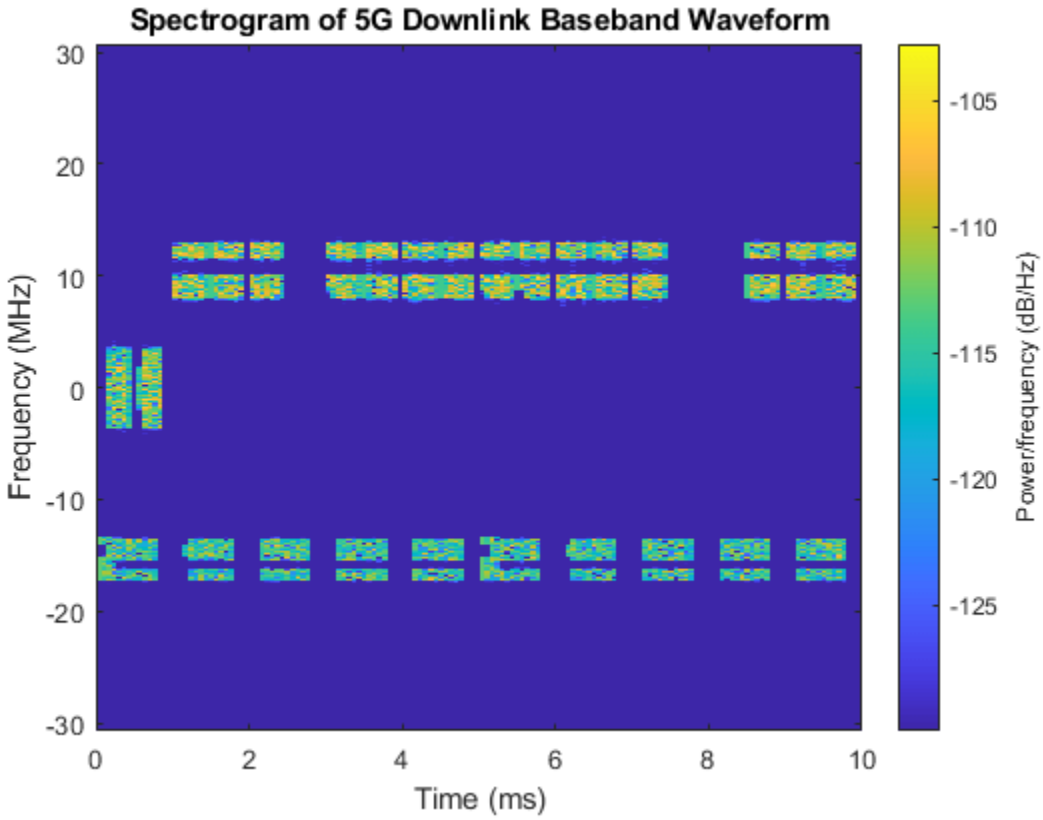
Plot the magnitude of the baseband waveform for the set of antenna ports defined.

```
figure;
plot(abs(waveform));
title('Magnitude of 5G Downlink Baseband Waveform');
xlabel('Sample Index');
ylabel('Magnitude');
```



Plot the spectogram of the waveform for the first antenna port.

```
samplerate = info.ResourceGrids(1).Info.SampleRate;
nfft = info.ResourceGrids(1).Info.Nfft;
figure;
spectrogram(waveform(:,1),ones(nfft,1),0,nfft,'centered',samplerate,'yaxis','MinThreshold',-130)
title('Spectrogram of 5G Downlink Baseband Waveform');
```



The waveform generator function returns the time-domain waveform and a structure `info`. The `info` structure contains the underlying resource element grid and a breakdown of the resources that all the PDSCH and PDCCH instances use in the waveform.

The `ResourceGrids` field is a structure array, which contains these fields.

- The resource grid corresponding to each BWP.
- The resource grid of the overall bandwidth containing the channels and signals in each BWP.
- An info structure with information corresponding to each BWP. For example, display the information of the first BWP.

```
disp('Modulation information associated with BWP 1:')
disp(info.ResourceGrids(1).Info)
```

```
Modulation information associated with BWP 1:
                  Nfft: 4096
            SampleRate: 61440000
    CyclicPrefixLengths: [320 288 288 288 288 288 288 320 288 288 288 ... ]
          SymbolLengths: [4416 4384 4384 4384 4384 4384 4384 4416 4384 ... ]
              Windowing: 0
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         SymbolsPerSlot: 14
       SlotsPerSubframe: 1
          SlotsPerFrame: 10
                     k0: 0
```

The generated resource grid is a 3-D matrix. The different planes in the grid represent the antenna ports in increasing port number order.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrWavegenBWPConfig` | `nrSCSCarrierConfig` | `nrULCarrierConfig` | `nrWavegenPDSCHConfig` | `nrWavegenSSBurstConfig` | `nrSearchSpaceConfig` | `nrWavegenPDCCHConfig` | `nrCORESETConfig` | `nrWavegenCSIRSConfig`

More About

- “5G NR Uplink Vector Waveform Generation” on page 2-2

NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals

This example shows the time-frequency aspects of the new radio (NR) physical downlink shared channel (PDSCH), the associated demodulation reference signal (DM-RS), and phase tracking reference signal (PT-RS). The example shows how PDSCH resource allocation affects the time-frequency structure of DM-RS and PT-RS.

Introduction

In 5G NR, PDSCH is the physical downlink channel that carries user data. DM-RS and PT-RS are the reference signals associated with PDSCH. These signals are generated within the PDSCH allocation, as defined in TS 38.211 Sections 7.4.1.1 and 7.4.1.2 [1] on page 1-29. DM-RS is used for channel estimation as part of coherent demodulation of PDSCH. To compensate for the common phase error (CPE), 3GPP 5G NR introduced PT-RS. Phase noise produced in local oscillators introduces a significant degradation at mmWave frequencies. It produces CPE and inter-carrier interference (ICI). CPE leads to an identical rotation of a received symbol in each subcarrier. ICI leads to loss of orthogonality between the subcarriers. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance.

The 5G Toolbox™ provides the functions for physical (PHY) layer modeling with varying levels of granularity. The levels of granularity range from PHY channel level functions that perform the transport and physical channel processing to individual channel processing stage functions performing cyclic redundancy check (CRC) coding, code block segmentation, low density parity check (LDPC) channel coding, and so on. The toolbox offers the reference signals functionality associated with the PDSCH as functions `nrPDSCHDMRS`, `nrPDSCHDMRSIndices`, `nrPDSCHPTRS`, and `nrPDSCHPTRSIndices`.

PDSCH

PDSCH is the physical channel that carries the user data. The resources allocated for PDSCH are within the bandwidth part (BWP) of the carrier, as defined in TS 38.214 Section 5.1.2 [2] on page 1-29. The resources in the time domain for PDSCH transmission are scheduled by downlink control information (DCI) in the field *Time domain resource assignment.* This field indicates the slot offset K_0, starting symbol **S**, the allocation length **L**, and the mapping type of PDSCH. The valid combinations of **S** and **L** are shown in Table 1. For mapping type A, value of *S* is 3 only when the DM-RS type A position is set to 3.

PDSCH Mapping Type	Normal Cyclic Prefix			Extended Cyclic Prefix		
	S	*L*	*S+L*	*S*	*L*	*S+L*
Type A	{0,1,2,3}	{3,...,14}	{3,...,14}	{0,1,2,3}	{3,...,12}	{3,...,12}
Type B	{0,...,12}	{2,4,7}	{2,...,14}	{0,...,10}	{2,4,6}	{2,...,12}

Table 1: Valid *S* and *L* Combinations

The resources in the frequency domain for PDSCH transmission are scheduled by a DCI in the field *Frequency domain resource assignment*. This field indicates whether the resource allocation of resource blocks (RBs) is contiguous or noncontiguous, based on the allocation type. The RBs allocated are within the BWP.

The 5G Toolbox™ provides the `nrCarrierConfig` and `nrPDSCHConfig` objects to set the parameters related to the PDSCH within the BWP.

```
% Setup the carrier with 15 kHz subcarrier spacing and 10 MHz bandwidth
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 52;
carrier.NStartGrid = 0;

% Configure the physical downlink shared channel parameters
pdsch = nrPDSCHConfig;
pdsch.NSizeBWP = [];   % Empty implies that the value is equal to NSizeGrid
pdsch.NStartBWP = [];  % Empty implies that the value is equal to NStartGrid
pdsch.PRBSet = 0:51;   % Allocate the complete carrier
pdsch.SymbolAllocation = [0 14]; % Symbol allocation [S L]
pdsch.MappingType = 'A'; % PDSCH mapping type ('A' or 'B')
```

DM-RS

DM-RS is used to estimate the radio channel. The signal is present only in the RBs allocated for the PDSCH. The DM-RS structure is designed to support different deployment scenarios and use cases. A front-loaded design supports low-latency transmissions, twelve orthogonal antenna ports for MIMO transmissions, and up to four reference signal transmission instances in a slot to support high-speed scenarios. The front-loaded reference signals indicate that the signal occurs early in the transmission. The DM-RS is present in each RB allocated for PDSCH.

Parameters That Control Time Domain Resources

The parameters that control DM-RS OFDM symbol locations are:

- PDSCH symbol allocation
- Mapping type
- DM-RS type A position
- DM-RS length
- DM-RS additional position

The symbol allocation of PDSCH indicates the OFDM symbol locations used by the PDSCH transmission in a slot. DM-RS symbol locations lie within the PDSCH symbol allocation. The positions of DM-RS OFDM symbols depend on the mapping type. The mapping type of PDSCH is either slot-wise (type A) or non-slot-wise (type B). The positions of any additional DM-RS symbols are defined by a set of tables, as specified in TS 38.211 Section 7.4.1.1.2 [1] on page 1-29. For the purpose of indexing the tables, the specification defines the term l_d indicating the duration of OFDM symbols to be accounted for, depending on the mapping type.

For mapping type A, the DM-RS OFDM symbol locations are defined relative to the first OFDM symbol of the slot (symbol #0). The location of first DM-RS OFDM symbol (l_0) is provided by the DM-RS type A position, which is either 2 or 3. For any additional DM-RS, the duration of OFDM symbols (l_d) is the number of OFDM symbols between the first OFDM symbol of the slot (symbol #0) and the

last OFDM symbol of the allocated PDSCH resources. Note that l_d may differ from the number of OFDM symbols allocated for PDSCH, when the first OFDM symbol of PDSCH is other than symbol #0.

For mapping type B, the DM-RS OFDM symbol locations are defined relative to the first OFDM symbol of allocated PDSCH resources. The location of first DM-RS OFDM symbol (l_0) is always 0, meaning that the first DM-RS OFDM symbol location is the first OFDM symbol location of the allocated PDSCH resources. For any additional DM-RS, the duration of OFDM symbols (l_d) is the duration of the allocated PDSCH resources.

Figure 1 illustrates the DM-RS symbol locations depending on the mapping type for an RB within a slot, having single-symbol DM-RS. The figure shows a configuration with PDSCH occupying the OFDM symbols from 1 to 10 (0-based) with l_d equal to 11 for mapping type A, and from 3 to 9 (0-based) with l_d equal to 7 for mapping type B respectively.

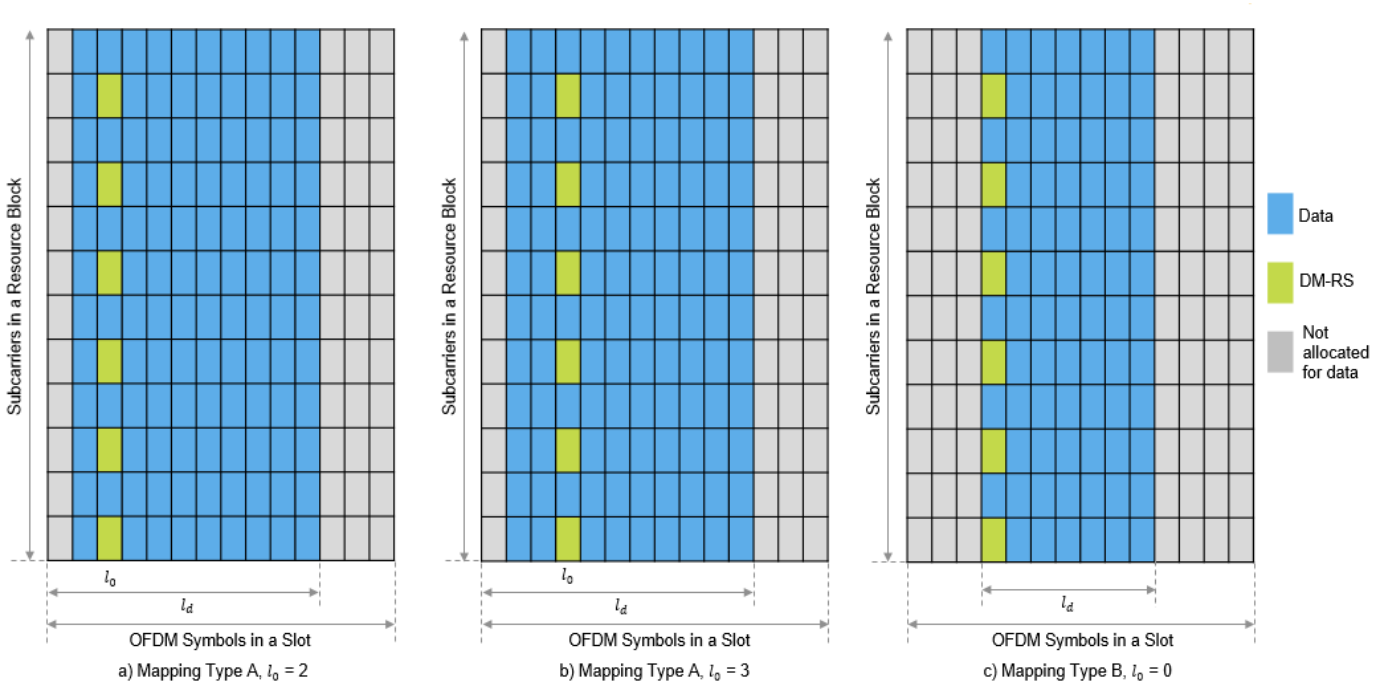


Figure 1: DM-RS Symbol Locations Based on Mapping Type

The maximum number of DM-RS OFDM symbols used by a UE is configured by RRC signaling (*dmrs-AdditionalPosition* and *maxLength*). The *maxLength* RRC parameter configures the length of DM-RS symbol, single symbol DM-RS or double symbol DM-RS. For double-symbol DM-RS, the actual selection is signaled in the DCI format 1_1 message. Figure 2 illustrates the single-symbol and double-symbol DM-RS locations.

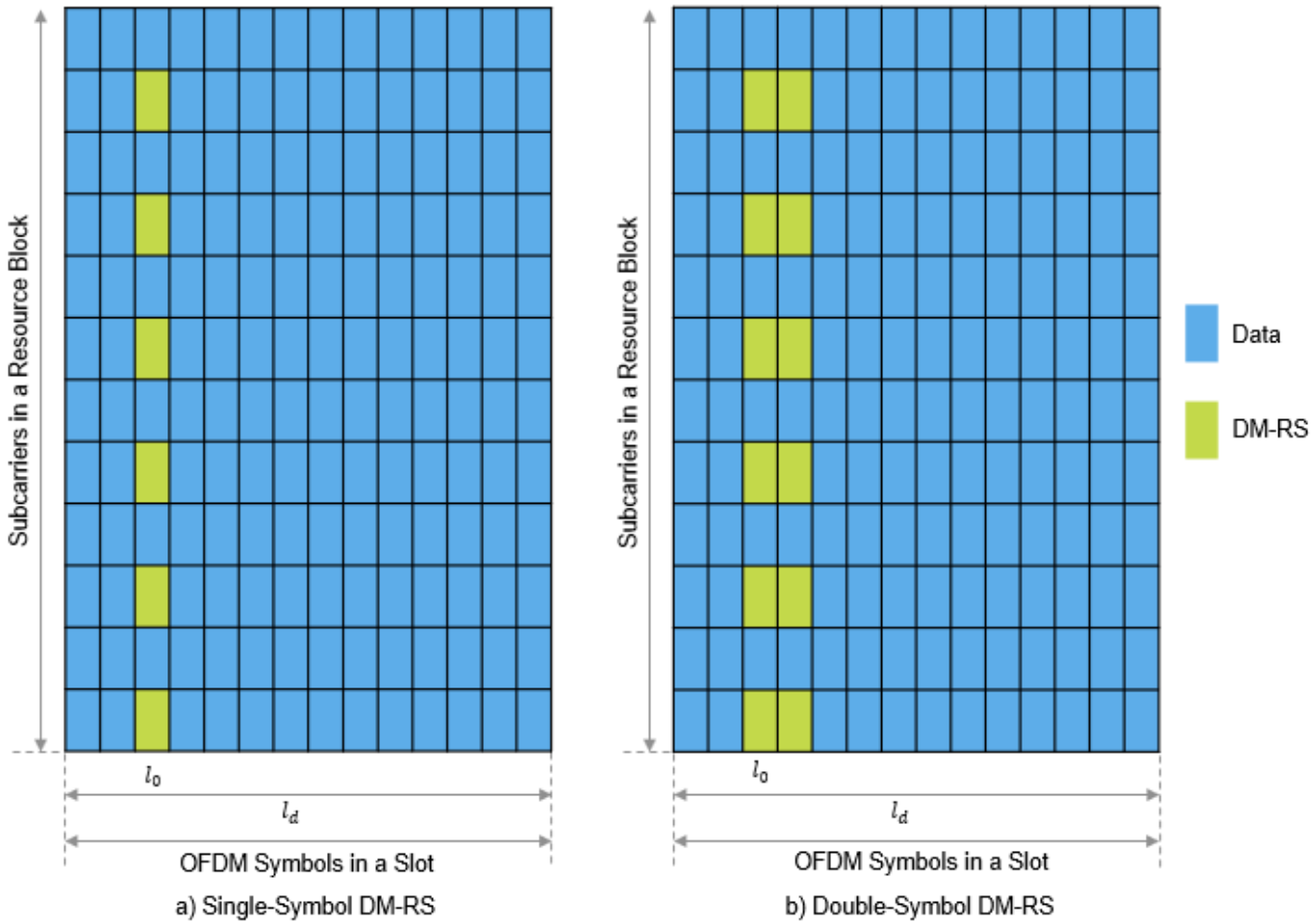


Figure 2: DM-RS Symbol Locations Based on Number of Front-Loaded DM-RS Symbols

The higher-layer parameter *dmrs-AdditionalPosition* defines the maximum number of additional single- or double-symbol DM-RS transmitted. The number of additional positions is in the range of 0 to 3 and depends on the mapping type, DM-RS length, and PDSCH symbol allocation. The DM-RS symbol locations are given by TS 38.211 Tables 7.4.1.1.2-3, and 7.4.1.1.2-4. Figure 3 illustrates the DM-RS additional positions in combination with single-symbol and double-symbol DM-RS.

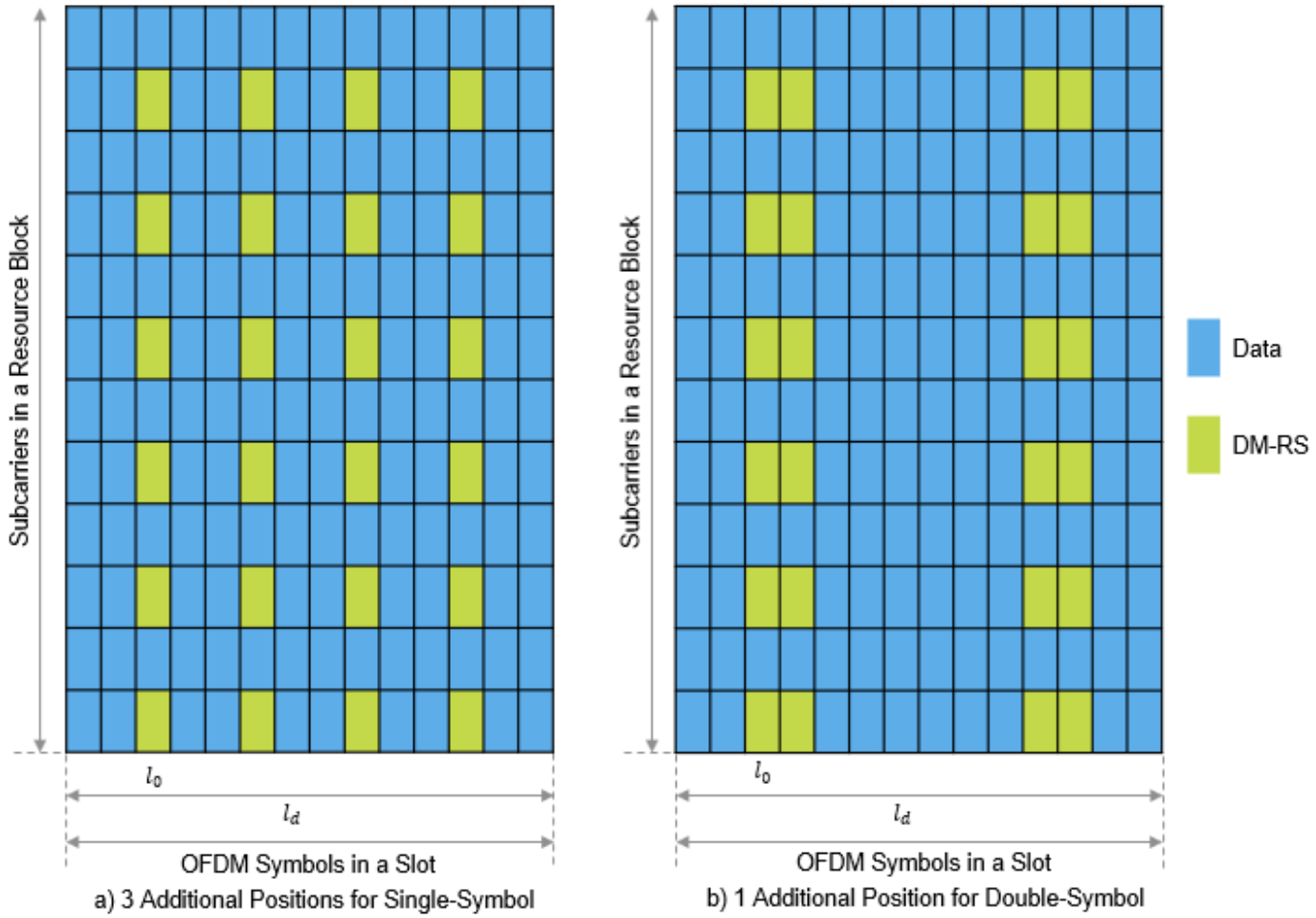


Figure 3: DM-RS Symbol Locations Based on Number of Additional DM-RS Positions

```
% Set the parameters that control the time resources of DM-RS
pdsch.DMRS.DMRSTypeAPosition = 2;      % 2 or 3
pdsch.DMRS.DMRSLength = 1;             % 1 or 2
pdsch.DMRS.DMRSAdditionalPosition = 1; % 0...3
```

Parameters That Control Frequency Domain Resources

The parameters that control the subcarrier locations of DM-RS are:

- DM-RS configuration type
- DM-RS antenna ports

The configuration type indicates the frequency density of DM-RS and is signaled by RRC message *dmrs-Type*. Configuration type 1 defines six subcarriers per physical resource block (PRB) per antenna port, comprising alternate subcarriers. Configuration type 2 defines four subcarriers per PRB per antenna port, consisting of two groups of two consecutive subcarriers. Figure 4 indicates the DM-RS subcarrier locations based on configuration type.

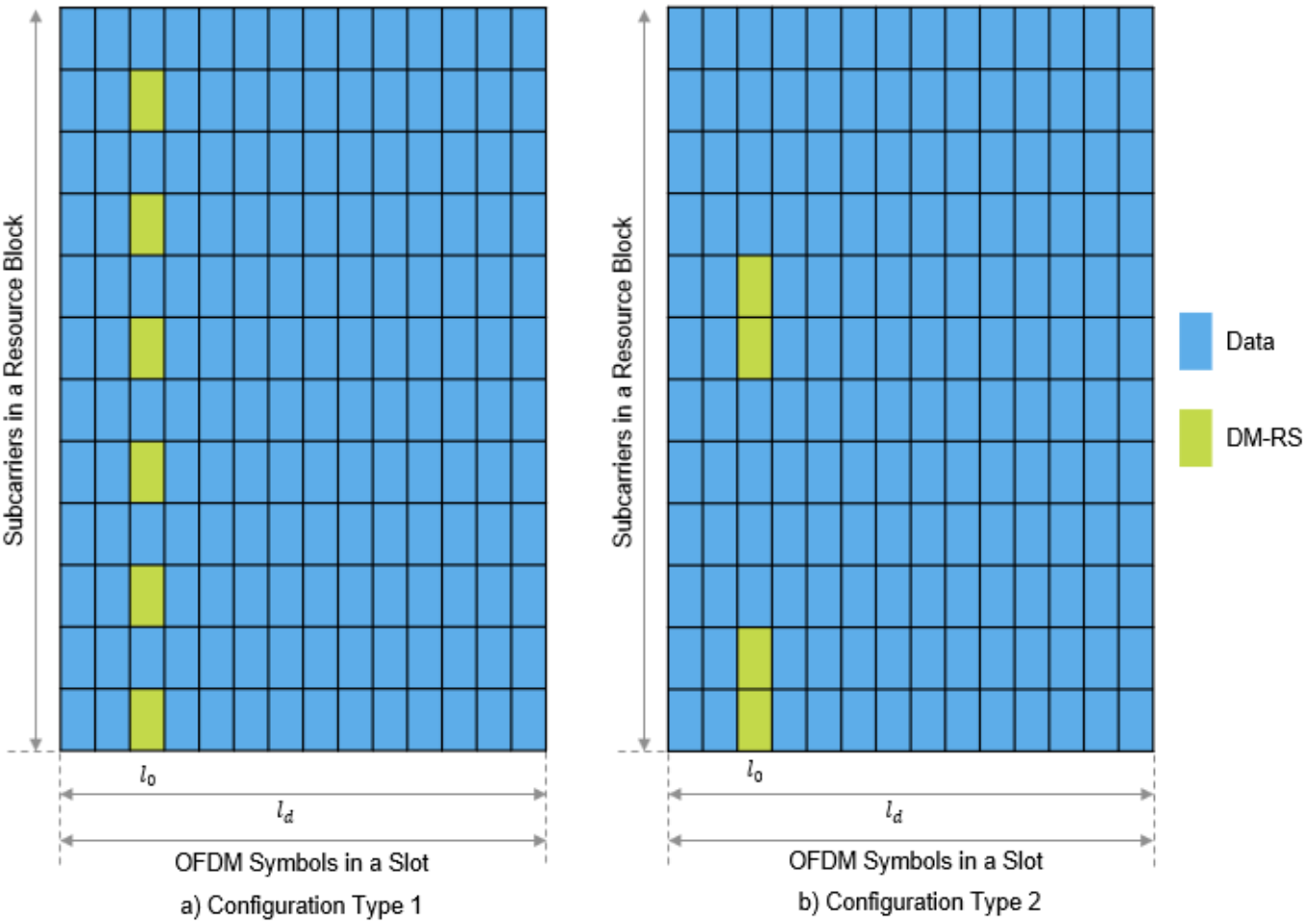


Figure 4: DM-RS Subcarrier Locations Based on DM-RS Configuration Type

Different delta shifts are applied to the sets of subcarriers used, depending on the associated antenna port or code division multiplexing (CDM) group. For configuration type 1, there are two possible CDM groups/shifts across eight possible antenna ports (p=0...7). Figure 5 illustrates the different shifts associated for DM-RS subcarrier locations with the DM-RS configuration type set to 1. Notice that the resource elements (REs) corresponding to the DM-RS subcarrier locations of lower CDM group (i.e. antenna port 0) are blocked for data transmission in the antenna ports of higher CDM group (i.e. antenna port 2).

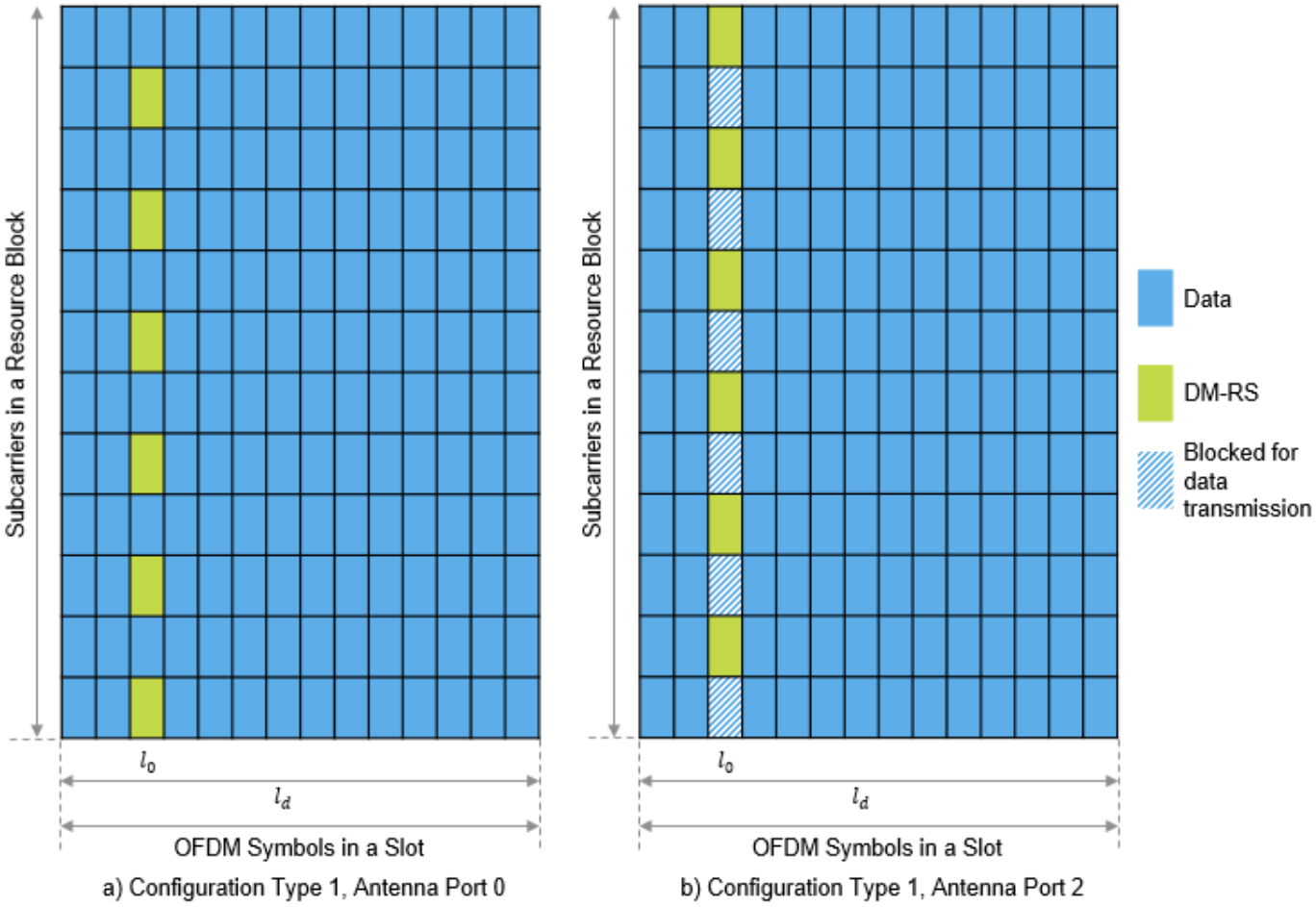


Figure 5: DM-RS Subcarrier Locations Based on DM-RS Antenna Ports for Configuration Type 1

For configuration type 2, there are three possible CDM groups/shifts across twelve antenna ports (p=0...11). Figure 6 illustrates the different shifts associated with DM-RS subcarrier locations in DM-RS configuration type 2. For full configuration details, see TS 38.211 Section 7.4.1.1. Notice that the REs corresponding to the DM-RS subcarrier locations of lower CDM groups are blocked for data transmission in the antenna ports of higher CDM groups.

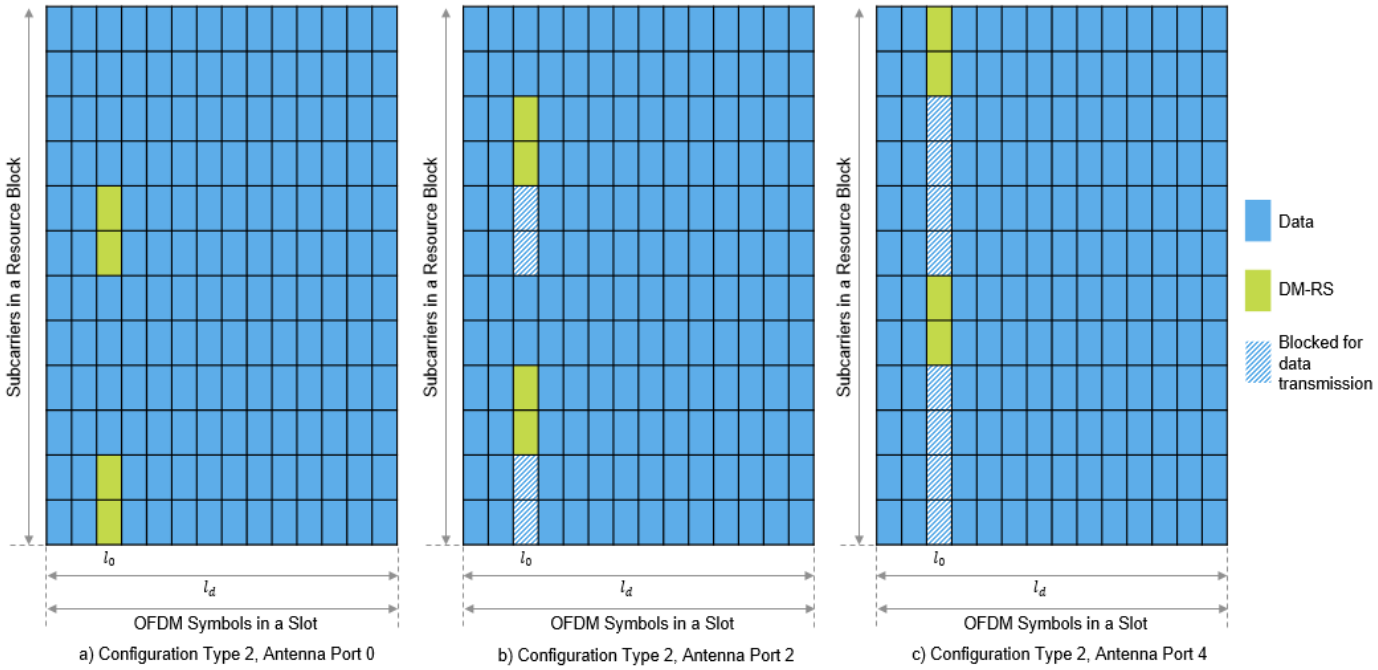


Figure 6: DM-RS Subcarrier Locations Based on DM-RS Antenna Ports for Configuration Type 2

```
% Set the parameters that control the frequency resources of DM-RS
pdsch.DMRS.DMRSConfigurationType = 1; % 1 or 2
pdsch.DMRS.DMRSPortSet = 0;

% Set the parameter that controls the number of REs available for data
% transmission in a DM-RS carrying OFDM symbol. This value is nominally
% greater than the maximum configured CDM group number.
pdsch.DMRS.NumCDMGroupsWithoutData = 1; % 1 corresponds to CDM group number 0

% The read-only properties DeltaShifts and DMRSSubcarrierLocations of DMRS
% property of pdsch object provides the values of delta shift(s) and DM-RS
% subcarrier locations in an RB for each antenna port configured.
pdsch.DMRS.DeltaShifts

ans = 0

pdsch.DMRS.DMRSSubcarrierLocations

ans = 6×1

     0
     2
     4
     6
     8
    10
```

Sequence Generation

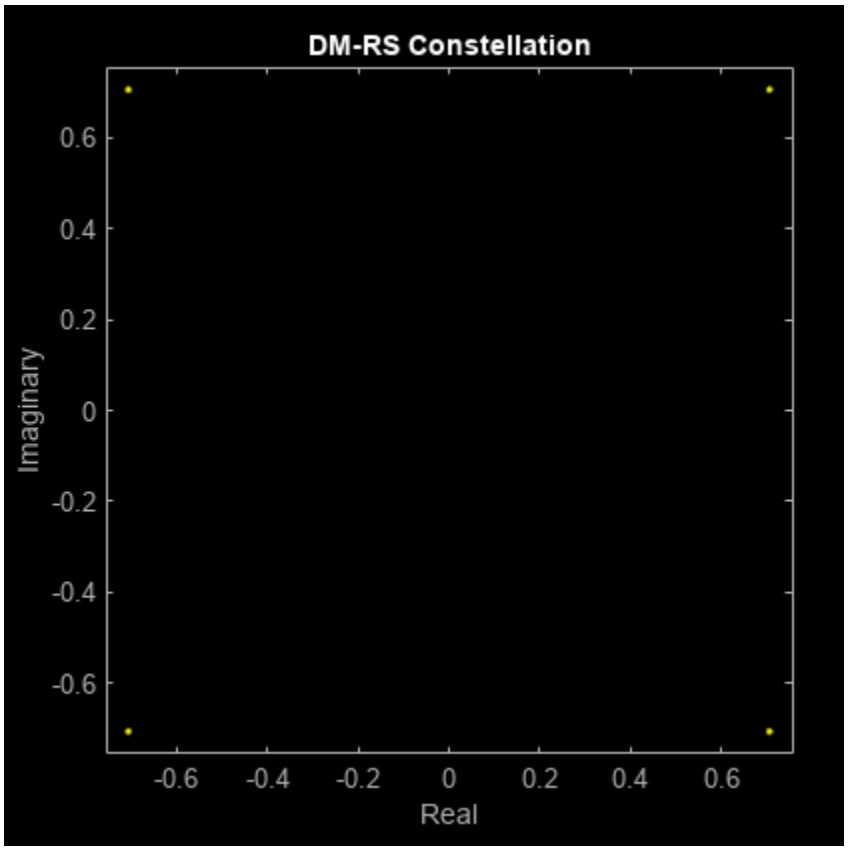
The pseudorandom sequence used for DM-RS is $2^{31} - 1$length gold sequence. The sequence is generated across all the common resource blocks (CRBs) and is transmitted only in the RBs allocated

for data because it is not required to estimate the channel outside the frequency region in which data is not transmitted. Generating the reference signal sequence across all the CRBs ensures that the same underlying pseudorandom sequence is used for multiple UEs on overlapping time-frequency resources in the case of a multi-user MIMO. The parameters that control the sequence generation are:

- DM-RS scrambling identity ($N_{\mathrm{ID}}^{n_{\mathrm{SCID}}}$)
- DM-RS scrambling initialization (n_{SCID})
- Number of OFDM symbols in a slot
- Slot number in a radio frame
- DM-RS symbol locations
- PRBs allocation

The CyclicPrefix property of the carrier object controls the number of OFDM symbols in a slot. The NSlot property of the carrier object controls the slot number.

```
% Set the parameters that only control the DM-RS sequence generation
pdsch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pdsch.DMRS.NSCID = 0;    % 0 or 1

% Generate DM-RS symbols
pdsch.NumLayers = numel(pdsch.DMRS.DMRSPortSet);
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);

% Plot the constellation
scatterplot(dmrsSymbols)
title('DM-RS Constellation')
xlabel('Real')
ylabel('Imaginary')
```




```
% The read-only properties TimeWeights and FrequencyWeights of DMRS
% property of pdsch object provides the values of time and frequency
% weights applied to the DM-RS symbols.
pdsch.DMRS.TimeWeights
```

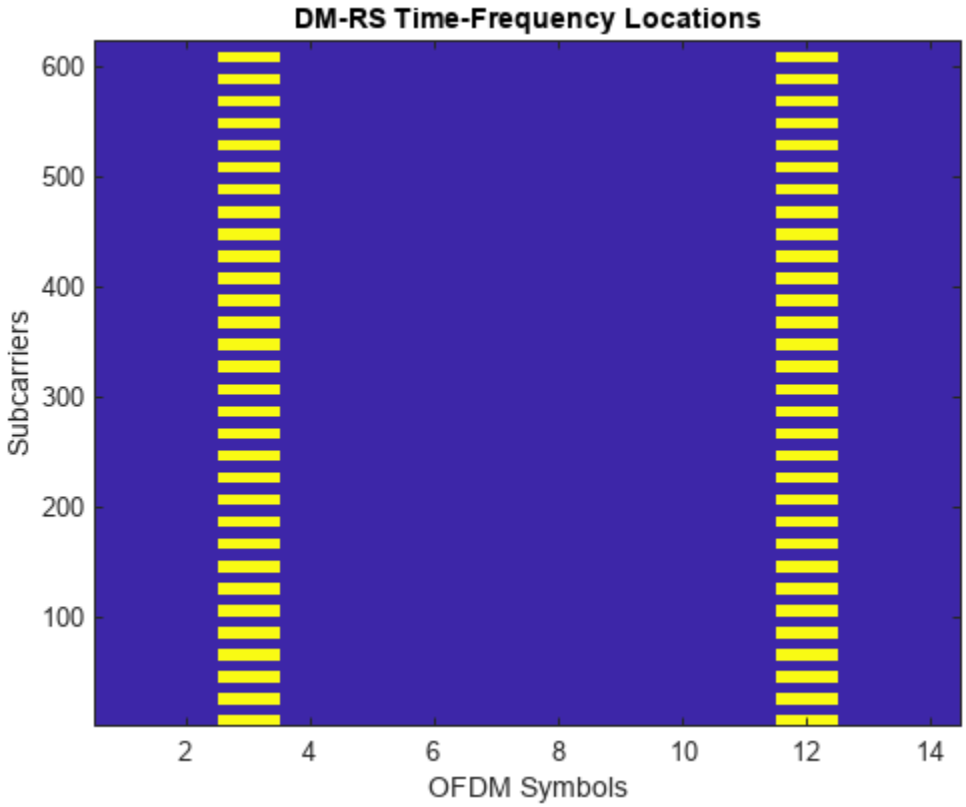
```
ans = 2×1

     1
     1
```

```
pdsch.DMRS.FrequencyWeights
```

```
ans = 2×1

     1
     1
```

```
% Generate DM-RS indices
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);

% Map the DM-RS symbols to the grid with the help of DM-RS indices
grid = zeros([12*carrier.NSizeGrid carrier.SymbolsPerSlot pdsch.NumLayers]);
grid(dmrsIndices) = dmrsSymbols;
figure
imagesc(abs(grid(:,:,1)));
axis xy;
```

```
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('DM-RS Time-Frequency Locations');
```



PT-RS

PT-RS is the phase tracking reference signal. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance. Due to the phase noise properties, PT-RS signal has a low density in the frequency domain and a high density in the time domain. PT-RS always occurs in combination with DM-RS and only when the network has configured PT-RS to be present.

Parameters That Control Time Domain Resources

PT-RS is configured through the higher layer parameter *DMRS-DownlinkConfig* for downlink. The parameters that control the time resources of PT-RS are:

- DM-RS symbol locations
- Time density of PT-RS (L_{PT-RS})

L_{PT-RS} depends on the scheduled modulation and coding scheme. Value of L_{PT-RS} must be from the set {1, 2, 4}. For the parameters that control DM-RS symbol locations, refer to Parameters that Control DM-RS Time Resources on page 1-16.

The PT-RS symbol locations in a slot start from the first OFDM symbol in the shared channel allocation and hop every L_{PT-RS} symbols, if no DM-RS symbol is present in this interval. In the case where a DM-RS symbol or symbols are present in between or at the hop interval, the hop starts from

the last DM-RS symbol location to provide the next PT-RS symbol. Figure 7 shows the PT-RS symbol locations in an RB for single slot with time-density set to 4 and DM-RS symbol locations set to 2 and 11 (0-based).

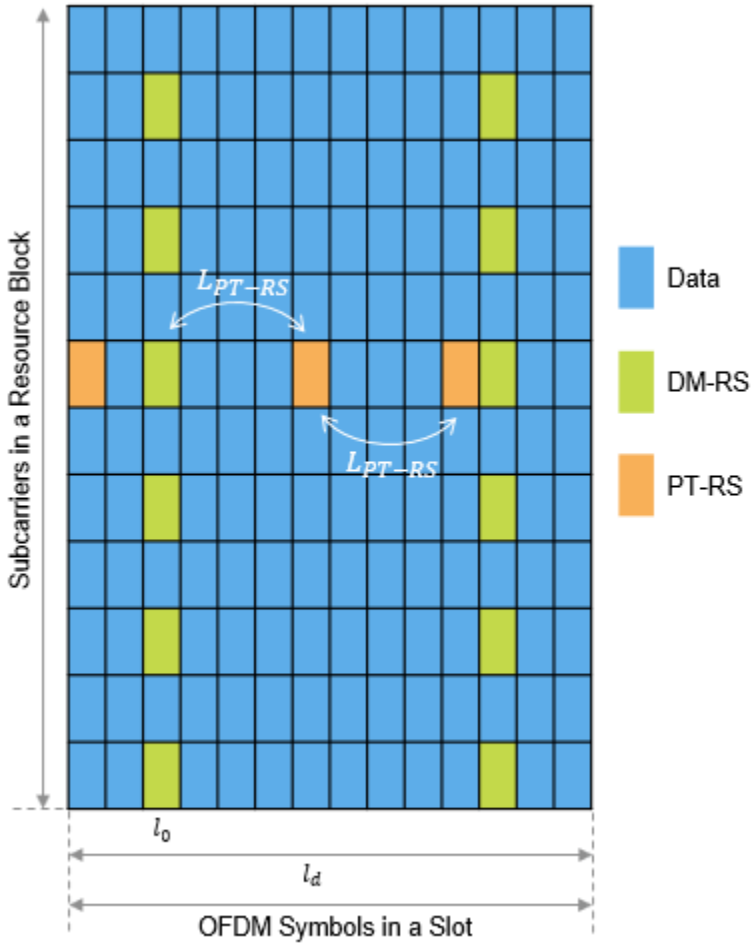


Figure 7: PT-RS Symbol Locations

```
% Set the EnablePTRS property in pdsch to 1
pdsch.EnablePTRS = 1;

% Set the parameters that control the time resources of PT-RS
pdsch.PTRS.TimeDensity = 4;
```

Parameters That Control Frequency Domain Resources

PT-RS occupies only one subcarrier in an RB for one OFDM symbol. The parameters that control the frequency resources of PT-RS are:

- PRB allocation
- DM-RS configuration type
- Frequency density of PT-RS ($K_{\mathrm{PT-RS}}$)
- Radio network temporary identifier (n_{RNTI})

- Resource element offset
- PT-RS antenna port

K_{PT-RS} depends on the scheduled bandwidth. The value of K_{PT-RS} is either 2 or 4, which indicates whether PT-RS is present in every two RBs or every four RBs.

The starting RB at which PT-RS is present (k_{ref}^{RB}), depends on K_{PT-RS}, n_{RNTI}, and the number of RBs (N_{RB}) allocated for PDSCH. For the purpose of mapping PT-RS, all the RBs of PDSCH are numbered in increasing order from 0 to $N_{RB} - 1$. The subcarrier location of PT-RS (k_{ref}^{RE}) within a resource block depends on the DM-RS configuration type, resource element (RE) offset, and PT-RS antenna port. The PT-RS antenna port must be a subset of DM-RS antenna ports. The PT-RS subcarrier location always aligns with one of the DM-RS subcarrier locations in an RB.

PT-RS in an RB occupies the same subcarrier locations in all the OFDM symbols where PT-RS is present.

```
% Set the parameters that affect the PT-RS subcarrier locations
pdsch.RNTI = 1;
pdsch.PTRS.FrequencyDensity = 2; % 2 or 4
pdsch.PTRS.REOffset = '10';      % '00', '01', '10', '11'
pdsch.PTRS.PTRSPortSet = min(pdsch.DMRS.DMRSPortSet);

% Set the other parameters that control PT-RS subcarrier locations
pdsch.DMRS.DMRSConfigurationType = 1;
pdsch.DMRS.DMRSPortSet = 0;
```

Sequence Generation

The sequence used to generate PT-RS is the same pseudorandom sequence used for DM-RS sequence generation. The values of the PT-RS sequence depend on the position of the first DM-RS symbol. For more details, refer to DM-RS sequence generation on page 1-22.

```
% Set the parameters that control the PT-RS sequence generation
pdsch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pdsch.DMRS.NSCID = 0;    % 0 or 1

% Generate PT-RS symbols
carrier.NSizeGrid = 4;
pdsch.PRBSet = 0:carrier.NSizeGrid-1;
pdsch.NumLayers = numel(pdsch.DMRS.DMRSPortSet);
ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);

% Generate PT-RS indices
ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
```

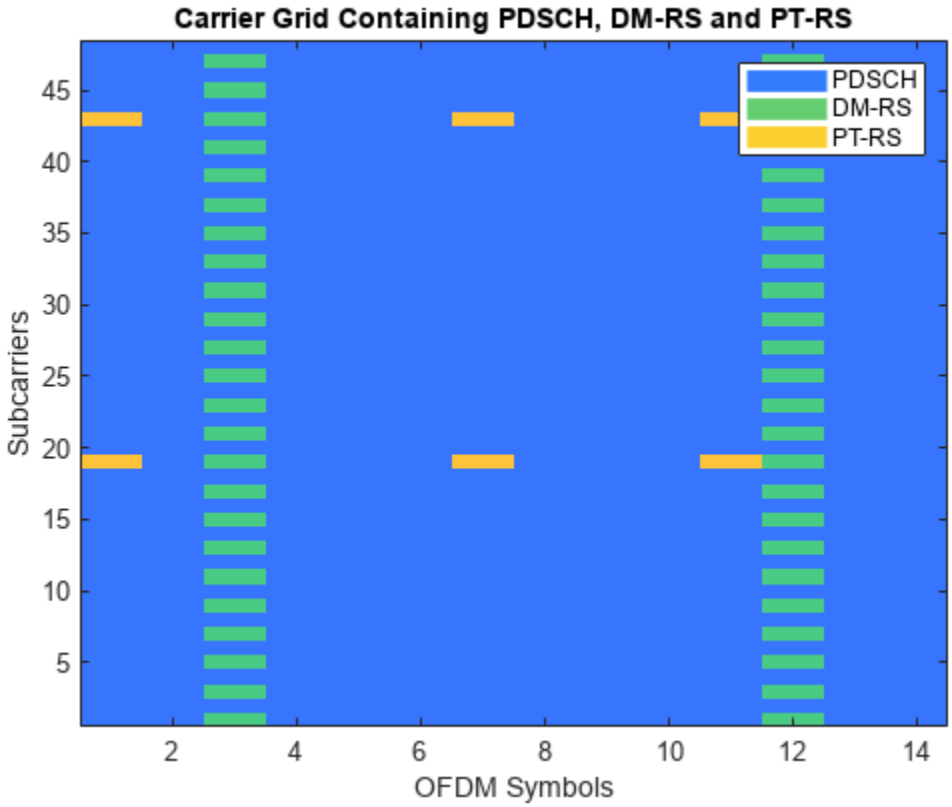
Get DM-RS symbols, RE indices of PDSCH, and DM-RS.

```
% PDSCH indices, DM-RS symbols and indices
[pdschIndices, pdschInfo] = nrPDSCHIndices(carrier,pdsch);
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
```

Map PDSCH, DM-RS, and PT-RS RE indices to the grid with scaled values to visualize the respective locations on the grid.

```
chpLevel = struct;
chpLevel.PDSCH = 0.4;
```

```
chpLevel.DMRS = 1;
chpLevel.PTRS = 1.4;
nSlotSymb = carrier.SymbolsPerSlot;
grid = complex(zeros(carrier.NSizeGrid*12,nSlotSymb,pdsch.NumLayers));
grid(pdschIndices) = chpLevel.PDSCH;
grid(dmrsIndices) = chpLevel.DMRS*dmrsSymbols;
grid(ptrsIndices) = chpLevel.PTRS*ptrsSymbols;
map = parula(64);
cscaling = 40;
im = image(cscaling*abs(grid(:,:,1)));
colormap(im.Parent,map);

% Add legend to the image
chpval = struct2cell(chpLevel);
clevels = cscaling*[chpval{:}];
N = length(clevels);
L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
% Index the color map and associated the selected colors with the lines
set(L,{'color'},mat2cell(map( min(1+clevels,length(map) ),:),ones(1,N),3)); % Set the colors acc
% Create legend
fnames = {'PDSCH','DM-RS','PT-RS'};
legend(fnames{:});
axis xy
title('Carrier Grid Containing PDSCH, DM-RS and PT-RS')
xlabel('OFDM Symbols')
ylabel('Subcarriers')
```



In the preceding figure, PT-RS is located at the start of the OFDM symbol in the PDSCH allocation. The symbols are present at every L_{PT-RS} hop interval from each other or from DM-RS symbols. PT-RS symbols in the frequency domain are located at subcarrier 19 (first RB) and at subcarrier 43 (third RB) of each OFDM symbol where PT-RS is present. The difference in consecutive subcarrier locations of PT-RS is 24, which is the number of subcarriers in an RB (12) times the frequency density of PT-RS (2).

Further Exploration

You can try changing the parameters that affect the time and frequency resources of reference signals and observe the variations in the RE positions for the respective signals.

Try performing channel estimation and phase tracking by using the reference signals. Compute the throughput by following the steps outlined in “NR PDSCH Throughput” on page 1-58.

This example shows how to generate the DM-RS and PT-RS sequences, and how to map the sequences to the OFDM carrier resource grid. It highlights the properties that control the time-frequency structure of reference signals.

References

1. 3GPP TS 38.211. "NR; Physical channels and modulation" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
2. 3GPP TS 38.214. "NR; Physical layer procedures for data" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
3. 3GPP TS 38.212. "NR; Multiplexing and channel coding" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions

`nrPDSCHIndices` | `nrPDSCHDMRSIndices` | `nrPDSCHDMRS` | `nrPDSCHPTRSIndices` | `nrPDSCHPTRS`

Objects

`nrCarrierConfig` | `nrPDSCHConfig`

Related Examples

- “NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 2-25

NR Cell Search and MIB and SIB1 Recovery

This example demonstrates how to use 5G Toolbox™ to synchronize, demodulate, and decode a live gNodeB signal. The example decodes the MIB and the first of the system information blocks (SIB1). Decoding MIB and SIB1 requires a comprehensive receiver, capable of demodulating and decoding the majority of the downlink channels and signals.

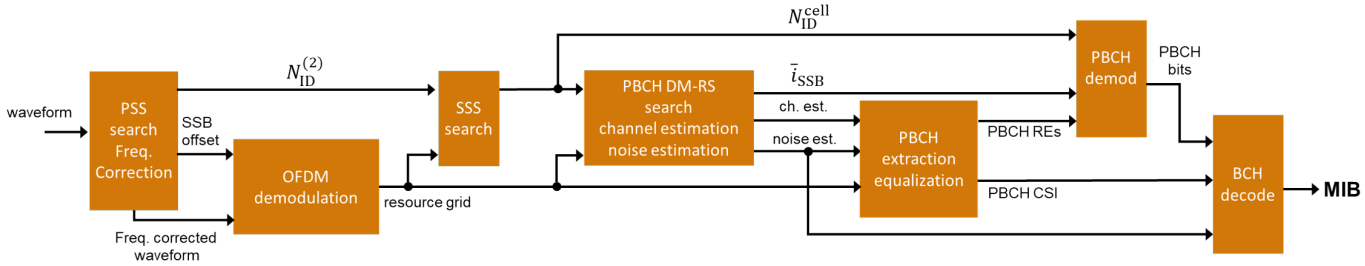
Introduction

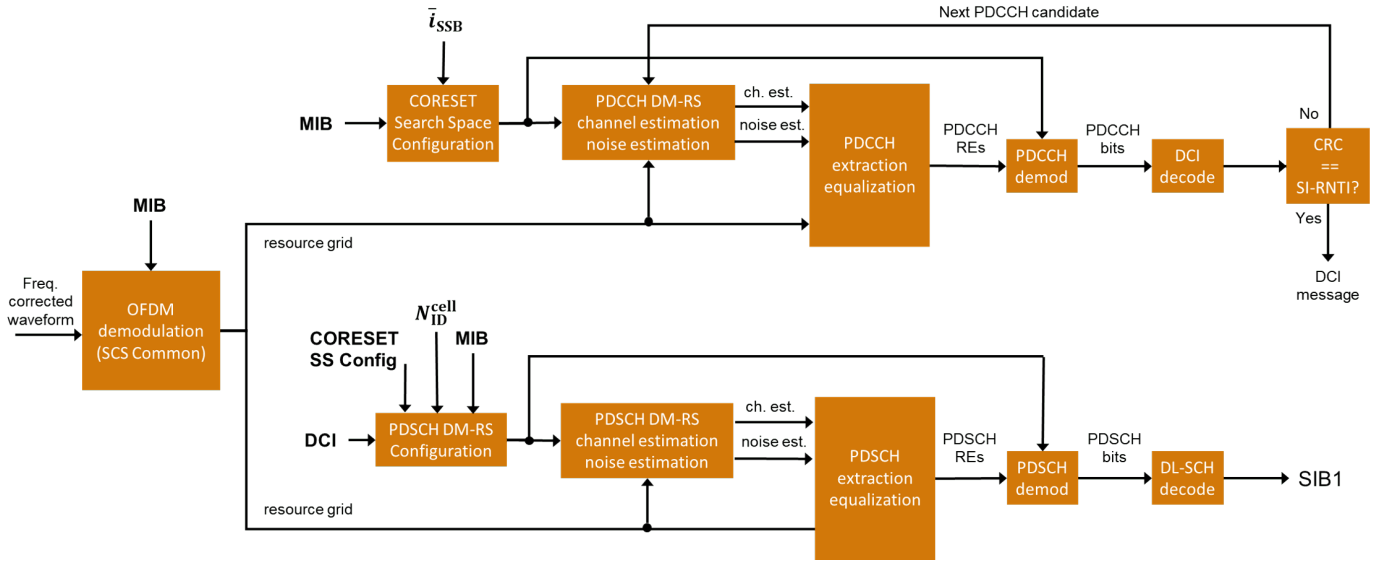
Before user equipment (UE) can communicate with the network, it must perform cell search and selection procedures and obtain initial system information. The first steps in that process are acquiring frame synchronization, finding out the cell identity and decoding the MIB and SIB1. This example shows how to perform these steps by using 5G Toolbox.

You can use this example with a captured waveform of I/Q samples or generate a local waveform containing a synchronization signal (SS) burst and SIB1 using `nrWaveformGenerator`. For locally generated waveforms, the example performs these steps:

- *Waveform generation*: Configure and generate a synchronization signal burst carrying the MIB, CORESET 0, PDCCH and PDSCH carrying SIB1 by using the downlink waveform generator from 5G Toolbox. The transmitter can enhance the SNR of one SS block, but it does not perform beamforming. For more information on SSB beamforming, see “NR SSB Beam Sweeping” on page 1-98.
- *AWGN*: Apply additive white Gaussian noise (AWGN) to the waveform.
- *Receiver*: Apply various synchronization and demodulation processes to the received waveform to establish the system frame number, cell identity and SSB, and decode the MIB. These provide the information required for blind decoding of downlink control information (DCI) in a PDCCH. The receiver uses DCI to configure the PDSCH demodulator, decode DL-SCH and finally recover the SIB1.

These figures show the processing steps inside the receiver.





Receiver Configuration

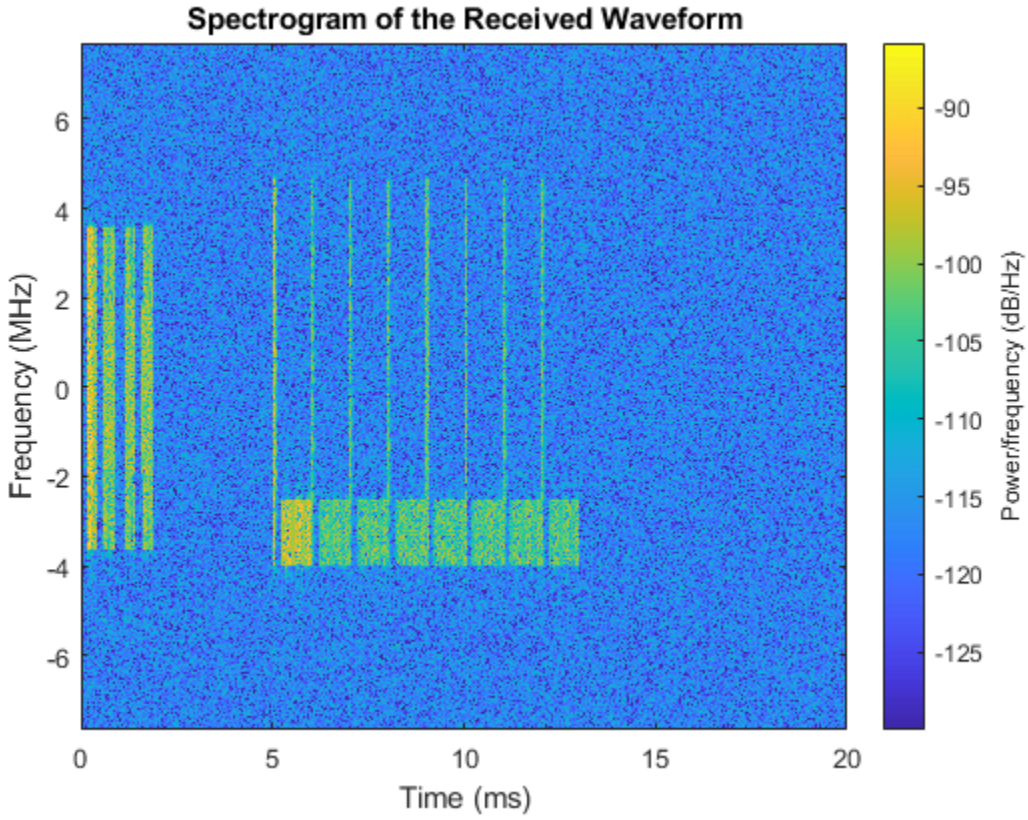
To synchronize and demodulate the received waveform, this information is needed:

- The waveform sample rate to demodulate the received waveform.
- The carrier center frequency to apply symbol phase compensation to the received waveform.
- The minimum channel bandwidth to determine CORESET 0 frequency resources. TS 38.101-1 Table 5.3.5-1 [1] describes the channel bandwidths for each NR band.
- The SS block pattern (Case A...E) to determine the subcarrier spacing of the SS/PBCH blocks. UE searches for SS block patterns based on the NR operating band. For more information, see TS 38.104 Tables 5.4.3.3-1 and 5.4.3.3-2 [2].
- The number of SS/PBCH blocks in a burst (L_{max}) to calculate parameters of the PBCH DM-RS sequences and PBCH descrambling. These parameters depend on the SS/PBCH block index as described in TS 38.211 Sections 7.3.3.1 and 7.4.1.4.1 [3]. TS 38.213 Section 4.1 [5] describes the set of SS/PBCH blocks in a burst in each case. UE knows the value of L_{max} based on the SS block pattern and the NR operating band.

```
loadFromFile = 0; % Set to 1 to load a captured waveform

if loadFromFile
    % Load captured waveform
    rx = load('capturedWaveformSIB1.mat');
    rxWaveform = rx.waveform;

    % Configure receiver sample rate (samples/second)
    sampleRate = rx.sampleRate;

    % Symbol phase compensation frequency. Specify the carrier center
    % frequency or set to 0 to disable symbol phase compensation
    fPhaseComp = rx.fPhaseComp; % Carrier center frequency (Hz)

    % Set the minimum channel bandwidth for the NR band required to
    % configure CORESET 0 in FR1 (See TS 38.101-1 Table 5.3.5-1)
    minChannelBW = rx.minChannelBW; % 5, 10, 40 MHz
```

```
    % Configure necessary burst parameters at the receiver. The SSB pattern
    % can be 'Case A','Case B','Case C' for FR1 or 'Case D','Case E' for
    % FR2. The maximum number of blocks L_max can be 4 or 8 for FR1 and 64
    % for FR2.
    refBurst.BlockPattern = rx.ssbBlockPattern;
    refBurst.L_max = rx.L_max;
else
    % Generate waveform containing SS burst and SIB1
    % Configure the cell identity
    config = struct();
    config.NCellID = 102;

    % Configure an SS burst
    config.BlockPattern = 'Case B';         % FR1: 'Case A','Case B','Case C'. FR2: 'Case D','Cas
    config.TransmittedBlocks = ones(1,8);   % Bitmap of SS blocks transmitted
    config.SubcarrierSpacingCommon = 15;    % SIB1 subcarrier spacing in kHz (15 or 30 for FR1. 6
    config.EnableSIB1 = 1;                  % Set to 0 to disable SIB1

    % Set the minimum channel bandwidth for the NR band required to
    % configure CORESET 0 in FR1 (See TS 38.101-1 Table 5.3.5-1)
    config.MinChannelBW = 5; % 5, 10, 40 MHz

    % Introduce a beamforming gain by boosting the power (and therefore
    % SNR) of one SSB and associated SIB1 PDCCH and PDSCH
    boost = 6; % SNR boost in dB
    config.Power = zeros(size(config.TransmittedBlocks));
    config.Power(1) = boost; % boost the first SSB

    % Configure and generate a waveform containing an SS burst and SIB1
    wavegenConfig = hSIB1WaveformConfiguration(config);
    [txWaveform,waveInfo] = nrWaveformGenerator(wavegenConfig);
    txOfdmInfo = waveInfo.ResourceGrids(1).Info;

    % Add white Gaussian noise to the waveform. Note that the SNR only
    % applies to the boosted SSB / SIB1
    rng('default'); % Reset the random number generator
    SNRdB = 20; % SNR for AWGN
    rxWaveform = awgn(txWaveform,SNRdB-boost,-10*log10(double(txOfdmInfo.Nfft)));

    % Configure receiver
    % Sample rate
    sampleRate = txOfdmInfo.SampleRate;

    % Symbol phase compensation frequency (Hz). The function
    % nrWaveformGenerator does not apply symbol phase compensation to the
    % generated waveform.
    fPhaseComp = 0; % Carrier center frequency (Hz)

    % Minimum channel bandwidth (MHz)
    minChannelBW = config.MinChannelBW;

    % Configure necessary burst parameters at the receiver
    refBurst.BlockPattern = config.BlockPattern;
    refBurst.L_max = numel(config.TransmittedBlocks);
end

% Get OFDM information from configured burst and receiver parameters
```

```
nrbSSB = 20;
scsSSB = hSSBurstSubcarrierSpacing(refBurst.BlockPattern);
rxOfdmInfo = nrOFDMInfo(nrbSSB,scsSSB,'SampleRate',sampleRate);

% Display spectrogram of received waveform
figure;
nfft = rxOfdmInfo.Nfft;
spectrogram(rxWaveform(:,1),ones(nfft,1),0,nfft,'centered',sampleRate,'yaxis','MinThreshold',-130
title('Spectrogram of the Received Waveform')
```



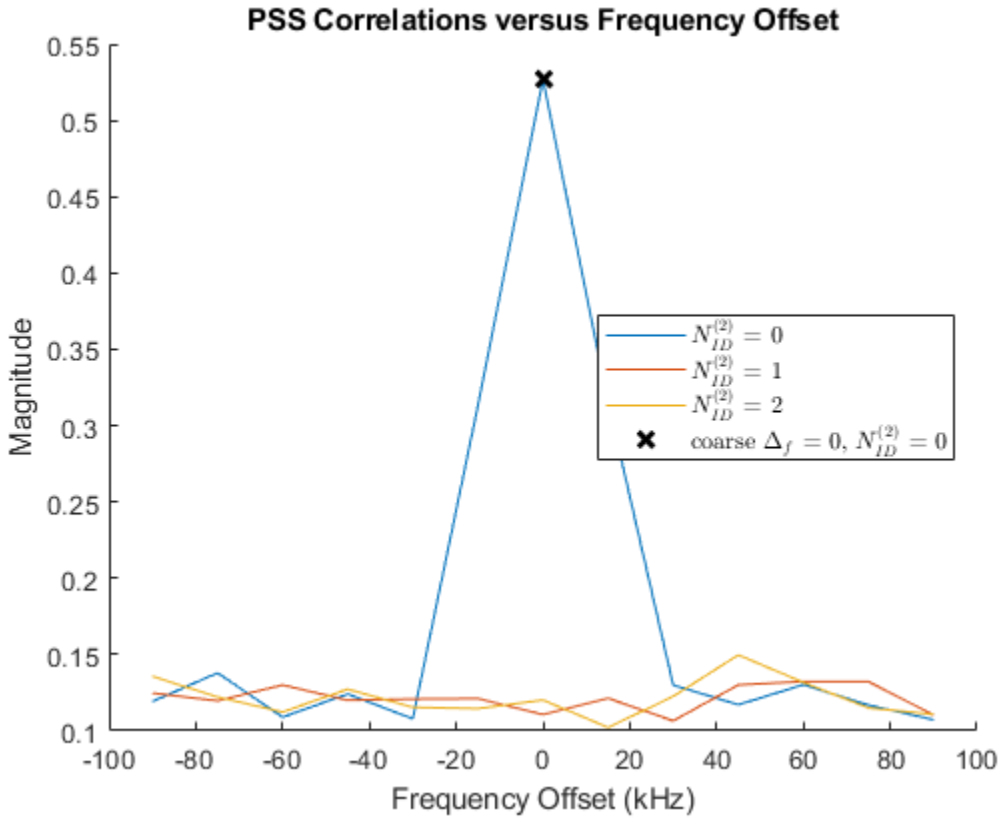
PSS Search and Frequency Offset Correction

The receiver performs PSS search and coarse frequency offset estimation following these steps:

- Frequency shift the received waveform with a candidate frequency offset. Candidate offsets are spaced half subcarrier apart. Use `searchBW` to control the frequency offset search bandwidth.
- Correlate the frequency-shifted received waveform with each of the three possible PSS sequences (NID2) and extract the strongest correlation peak. The reference PSS sequences are centered in frequency. Therefore, the strongest correlation peak provides a measure of coarse frequency offset with respect to the center frequency of the carrier. The peak also indicates which of the three PSS (NID2) has been detected in the received waveform and the time instant of the best channel conditions.
- Estimate frequency offsets below half subcarrier by correlating the cyclic prefix of each OFDM symbol in the SSB with the corresponding useful parts of the OFDM symbols. The phase of this correlation is proportional to the frequency offset in the waveform.

```
disp(' -- Frequency correction and timing estimation --')

% Specify the frequency offset search bandwidth in kHz
searchBW = 6*scsSSB;
[rxWaveform,freqOffset,NID2] = hSSBurstFrequencyCorrect(rxWaveform,refBurst.BlockPattern,sampleRa
disp([' Frequency offset: ' num2str(freqOffset,'%.0f') ' Hz'])
```

```
 -- Frequency correction and timing estimation --
 Frequency offset: 65 Hz
```



Time Synchronization and OFDM Demodulation

The receiver estimates the timing offset to the strongest SS block by using the reference PSS sequence detected in the frequency search process. After frequency offset correction, the receiver can assume that the center frequencies of the reference PSS and received waveform are aligned. Finally, the receiver OFDM demodulates the synchronized waveform and extracts the SS block.

```
% Create a reference grid for timing estimation using detected PSS. The PSS
% is placed in the second OFDM symbol of the reference grid to avoid the
% special CP length of the first OFDM symbol.
refGrid = zeros([nrbSSB*12 2]);
refGrid(nrPSSIndices,2) = nrPSS(NID2); % Second OFDM symbol for correct CP length

% Timing estimation. This is the timing offset to the OFDM symbol prior to
% the detected SSB due to the content of the reference grid
nSlot = 0;
timingOffset = nrTimingEstimate(rxWaveform,nrbSSB,scsSSB,nSlot,refGrid,'SampleRate',sampleRate);
```

```
% Synchronization, OFDM demodulation, and extraction of strongest SS block
rxGrid = nrOFDMDemodulate(rxWaveform(1+timingOffset:end,:),nrbSSB,scsSSB,nSlot,'SampleRate',samp
rxGrid = rxGrid(:,2:5,:);

% Display the timing offset in samples. As the symbol lengths are measured
% in FFT samples, scale the symbol lengths to account for the receiver
% sample rate.
srRatio = sampleRate/(scsSSB*1e3*rxOfdmInfo.Nfft);
firstSymbolLength = rxOfdmInfo.SymbolLengths(1)*srRatio;
str = sprintf(' Time offset to synchronization block: %%.0f samples (%%.%.0ff ms) \n',floor(log1
fprintf(str,timingOffset+firstSymbolLength,(timingOffset+firstSymbolLength)/sampleRate*1e3);
```

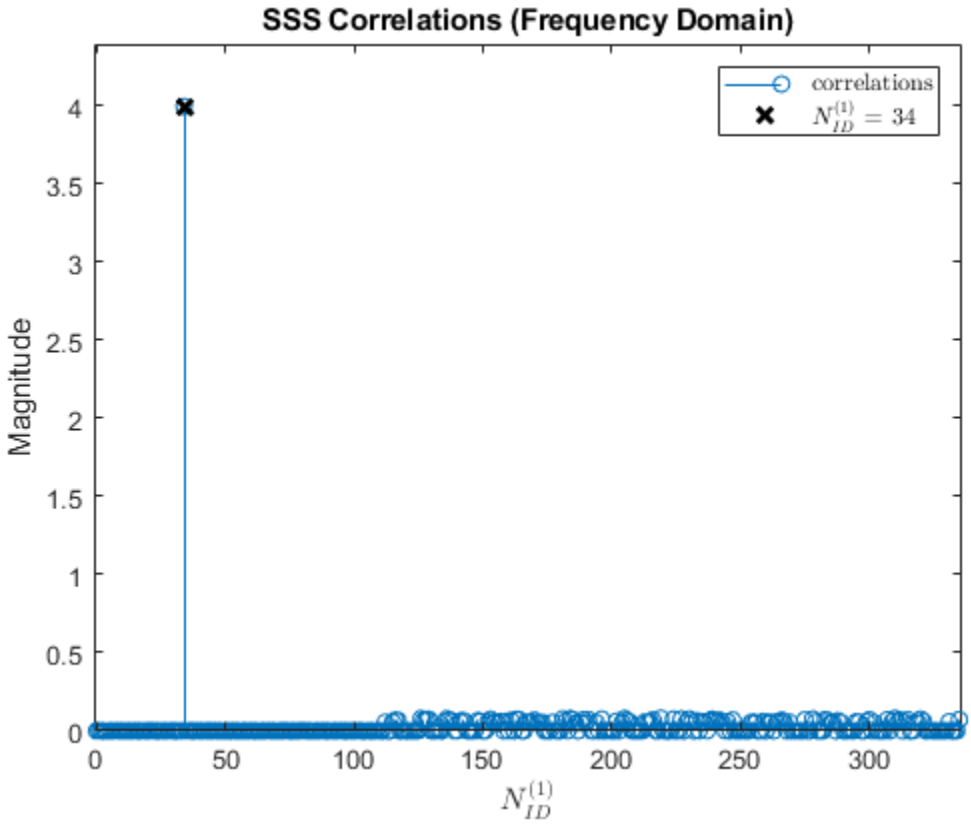
```
 Time offset to synchronization block: 2200 samples (0.1432 ms)
```

SSS Search

The receiver extracts the resource elements associated to the SSS from the received grid and correlates them with each possible SSS sequence generated locally. The indices of the strongest PSS and SSS sequences combined give the physical layer cell identity, which is required for PBCH DM-RS and PBCH processing.

```
% Extract the received SSS symbols from the SS/PBCH block
sssIndices = nrSSSIndices;
sssRx = nrExtractResources(sssIndices,rxGrid);

% Correlate received SSS symbols with each possible SSS sequence
sssEst = zeros(1,336);
for NID1 = 0:335

    ncellid = (3*NID1) + NID2;
    sssRef = nrSSS(ncellid);
    sssEst(NID1+1) = sum(abs(mean(sssRx .* conj(sssRef),1)).^2);

end

% Plot SSS correlations
figure;
stem(0:335,sssEst,'o');
title('SSS Correlations (Frequency Domain)');
xlabel('$N_{ID}^{(1)}$','Interpreter','latex');
ylabel('Magnitude');
axis([-1 336 0 max(sssEst)*1.1]);

% Determine NID1 by finding the strongest correlation
NID1 = find(sssEst==max(sssEst)) - 1;

% Plot selected NID1
hold on;
plot(NID1,max(sssEst),'kx','LineWidth',2,'MarkerSize',8);
legend(["correlations" "$N_{ID}^{(1)}$ = " + num2str(NID1)],'Interpreter','latex');

% Form overall cell identity from estimated NID1 and NID2
ncellid = (3*NID1) + NID2;

disp([' Cell identity: ' num2str(ncellid)])
```

```
 Cell identity: 102
```

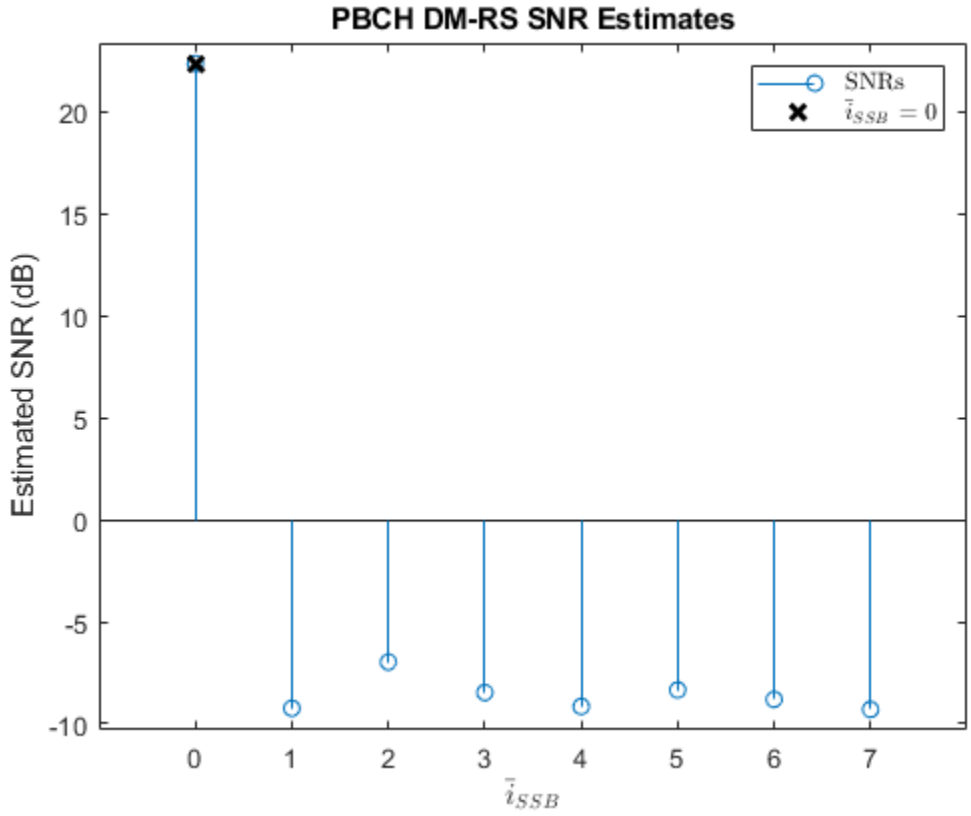


PBCH DM-RS search

In a process similar to SSS search, the receiver constructs each possible PBCH DM-RS sequence and performs channel and noise estimation. The index of the PBCH DM-RS with the best SNR determines the LSBs of the SS/PBCH block index required for PBCH scrambling initialization.

```
% Calculate PBCH DM-RS indices
dmrsIndices = nrPBCHDMRSIndices(ncellid);

% Perform channel estimation using DM-RS symbols for each possible DM-RS
% sequence and estimate the SNR
dmrsEst = zeros(1,8);
for ibar_SSB = 0:7

    refGrid = zeros([240 4]);
    refGrid(dmrsIndices) = nrPBCHDMRS(ncellid,ibar_SSB);
    [hest,nest] = nrChannelEstimate(rxGrid,refGrid,'AveragingWindow',[0 1]);
    dmrsEst(ibar_SSB+1) = 10*log10(mean(abs(hest(:).^2)) / nest);

end

% Plot PBCH DM-RS SNRs
figure;
stem(0:7,dmrsEst,'o');
title('PBCH DM-RS SNR Estimates');
xlabel('$\overline{i}_{SSB}$','Interpreter','latex');
xticks(0:7);
```

```
ylabel('Estimated SNR (dB)');
axis([-1 8 min(dmrsEst)-1 max(dmrsEst)+1]);

% Record ibar_SSB for the highest SNR
ibar_SSB = find(dmrsEst==max(dmrsEst)) - 1;

% Plot selected ibar_SSB
hold on;
plot(ibar_SSB,max(dmrsEst),'kx','LineWidth',2,'MarkerSize',8);
legend(["SNRs" "$\overline{i}_{SSB}$ = " + num2str(ibar_SSB)],'Interpreter','latex');
```



Channel Estimation using PBCH DM-RS and SSS

The receiver estimates the channel for the entire SS/PBCH block using the SSS and PBCH DM-RS detected in previous steps. An estimate of the additive noise on the PBCH DM-RS / SSS is also performed.

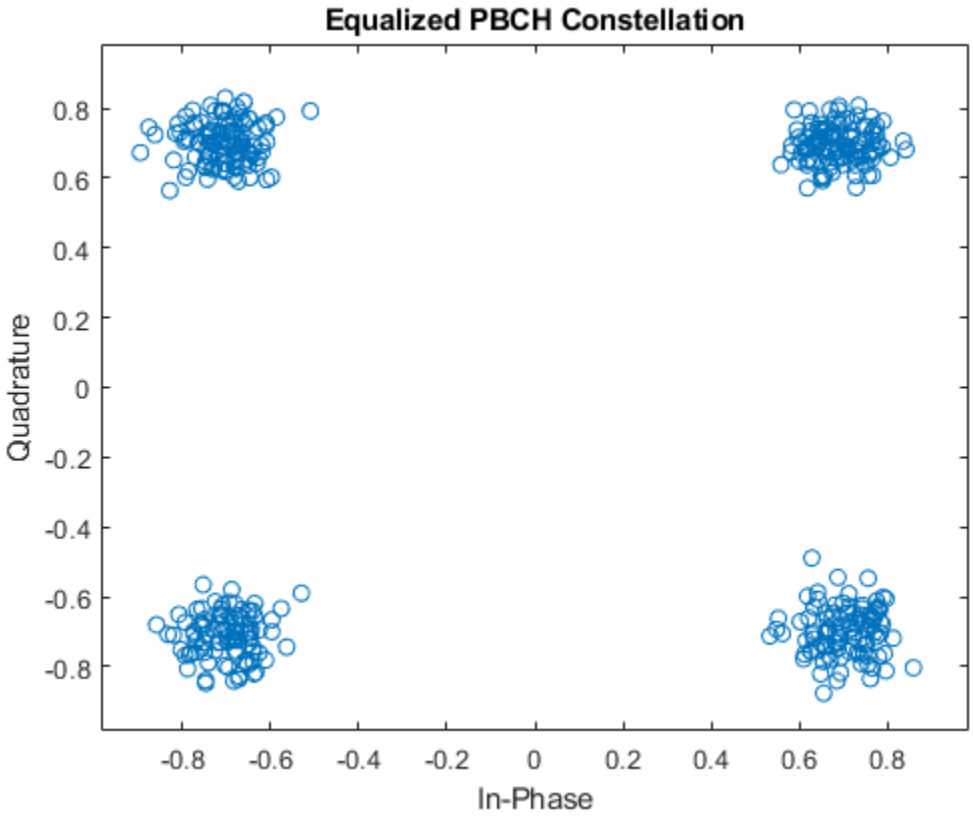
```
refGrid = zeros([nrbSSB*12 4]);
refGrid(dmrsIndices) = nrPBCHDMRS(ncellid,ibar_SSB);
refGrid(sssIndices) = nrSSS(ncellid);
[hest,nest,hestInfo] = nrChannelEstimate(rxGrid,refGrid,'AveragingWindow',[0 1]);
```

PBCH Demodulation

The receiver uses the cell identity to determine and extract the resource elements associated with the PBCH from the received grid. In addition, the receiver uses the channel and noise estimates to perform MMSE equalization. The equalized PBCH symbols are then demodulated and descrambled to give bit estimates for the coded BCH block.

```
disp(' -- PBCH demodulation and BCH decoding -- ')

% Extract the received PBCH symbols from the SS/PBCH block
[pbchIndices,pbchIndicesInfo] = nrPBCHIndices(ncellid);
pbchRx = nrExtractResources(pbchIndices,rxGrid);

% Configure 'v' for PBCH scrambling according to TS 38.211 Section 7.3.3.1
% 'v' is also the 2 LSBs of the SS/PBCH block index for L_max=4, or the 3
% LSBs for L_max=8 or 64.
if refBurst.L_max == 4
    v = mod(ibar_SSB,4);
else
    v = ibar_SSB;
end
ssbIndex = v;

% PBCH equalization and CSI calculation
pbchHest = nrExtractResources(pbchIndices,hest);
[pbchEq,csi] = nrEqualizeMMSE(pbchRx,pbchHest,nest);
Qm = pbchIndicesInfo.G / pbchIndicesInfo.Gd;
csi = repmat(csi.',Qm,1);
csi = reshape(csi,[],1);

% Plot received PBCH constellation after equalization
figure;
plot(pbchEq,'o');
xlabel('In-Phase'); ylabel('Quadrature')
title('Equalized PBCH Constellation');
m = max(abs([real(pbchEq(:)); imag(pbchEq(:))])) * 1.1;
axis([-m m -m m]);

% PBCH demodulation
pbchBits = nrPBCHDecode(pbchEq,ncellid,v,nest);

% Calculate RMS PBCH EVM
pbchRef = nrPBCH(pbchBits<0,ncellid,v);
evm = comm.EVM;
pbchEVMrms = evm(pbchRef,pbchEq);

% Display calculated EVM
disp([' PBCH RMS EVM: ' num2str(pbchEVMrms,'%0.3f') '%']);
```

```
 -- PBCH demodulation and BCH decoding --
 PBCH RMS EVM: 8.687%
```



BCH Decoding

The receiver weights BCH bit estimates with channel state information (CSI) from the MMSE equalizer and decodes the BCH. BCH decoding consists of rate recovery, polar decoding, CRC decoding, descrambling, and separating the 24 BCH transport block bits from the 8 additional timing-related payload bits.

```
% Apply CSI
pbchBits = pbchBits .* csi;

% Perform BCH decoding including rate recovery, polar decoding, and CRC
% decoding. PBCH descrambling and separation of the BCH transport block
% bits 'trblk' from 8 additional payload bits A...A+7 is also performed:
%   A ... A+3: 4 LSBs of system frame number
%         A+4: half frame number
% A+5 ... A+7: for L_max=64, 3 MSBs of the SS/PBCH block index
%              for L_max=4 or 8, A+5 is the MSB of subcarrier offset k_SSB
polarListLength = 8;
[~,crcBCH,trblk,sfn4lsb,nHalfFrame,msbidxoffset] = ...
    nrBCHDecode(pbchBits,polarListLength,refBurst.L_max,ncellid);

% Display the BCH CRC
disp([' BCH CRC: ' num2str(crcBCH)]);

% Stop processing MIB and SIB1 if BCH was received with errors
if crcBCH
    disp(' BCH CRC is not zero.');
```

```
    return
end

% Use 'msbidxoffset' value to set bits of 'k_SSB' or 'ssbIndex', depending
% on the number of SS/PBCH blocks in the burst
if (refBurst.L_max==64)
    ssbIndex = ssbIndex + (bit2int(msbidxoffset,3) * 8);
    k_SSB = 0;
else
    k_SSB = msbidxoffset * 16;
end

% Displaying the SSB index
disp([' SSB index: ' num2str(ssbIndex)]);
```

```
 BCH CRC: 0
 SSB index: 0
```

MIB and BCH Parsing

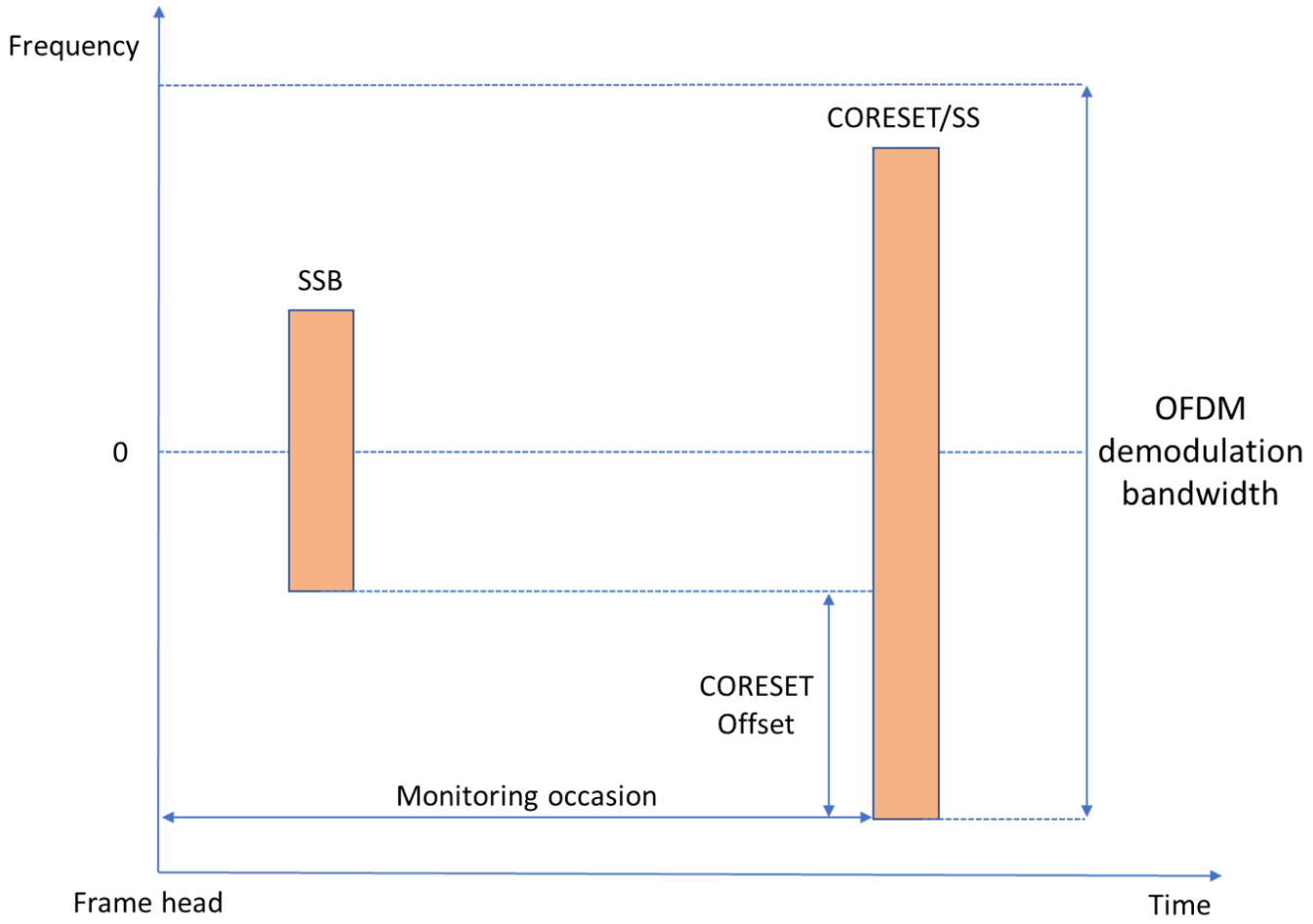
The example parses the 24 decoded BCH transport block bits into a MIB message and creates the `initialSystemInfo` structure with initial system information. This process includes reconstituting the 10-bit system frame number (SFN) `NFrame` from the 6 MSBs in the MIB and the 4 LSBs in the PBCH payload bits. It also includes incorporating the MSB of the subcarrier offset `k_SSB` from the PBCH payload bits in the case of L_max=4 or 8 SS/PBCH blocks per burst.

```
% Parse the last 23 decoded BCH transport block bits into a MIB message.
% The BCH transport block 'trblk' is the RRC message BCCH-BCH-Message,
% consisting of a leading 0 bit and 23 bits corresponding to the MIB. The
% leading bit signals the message type transmitted (MIB or empty sequence).

mib = fromBits(MIB,trblk(2:end)); % Do not parse leading bit

% Create a structure containing complete initial system information
initialSystemInfo = initSystemInfo(mib,sfn4lsb,k_SSB,refBurst.L_max);

% Display the MIB structure
disp(' BCH/MIB Content:')
disp(initialSystemInfo);

% Check if a CORESET for Type0-PDCCH common search space (CSS) is present,
% according to TS 38.213 Section 4.1
if ~isCORESET0Present(refBurst.BlockPattern,initialSystemInfo.k_SSB)
    fprintf('CORESET 0 is not present (k_SSB > k_SSB_max).\n');
    return
end
```

```
 BCH/MIB Content:
                    NFrame: 0
    SubcarrierSpacingCommon: 15
                      k_SSB: 0
          DMRSTypeAPosition: 3
            PDCCHConfigSIB1: [1x1 struct]
                 CellBarred: 0
       IntraFreqReselection: 0
```

OFDM Demodulation on Full Bandwidth

Once the MIB is recovered, the receiver uses common subcarrier spacing and a bandwidth supporting CORESET 0 to OFDM demodulate the frame containing the detected SS block. The receiver determines the CORESET 0 frequency resources in common numerology through an offset from the location of the SSB detected and a bandwidth specified in TS 38.213 Section 13 Tables 13-1 through 13-10 [5]. The frequency correction process aligned the center of the OFDM resource grid with the center frequency of the SS burst. However, these centers are not necessarily aligned with the center frequency of CORESET 0. This figure shows the relationship between the SSB, CORESET 0 frequency resources and associated PDCCH monitoring occasions.



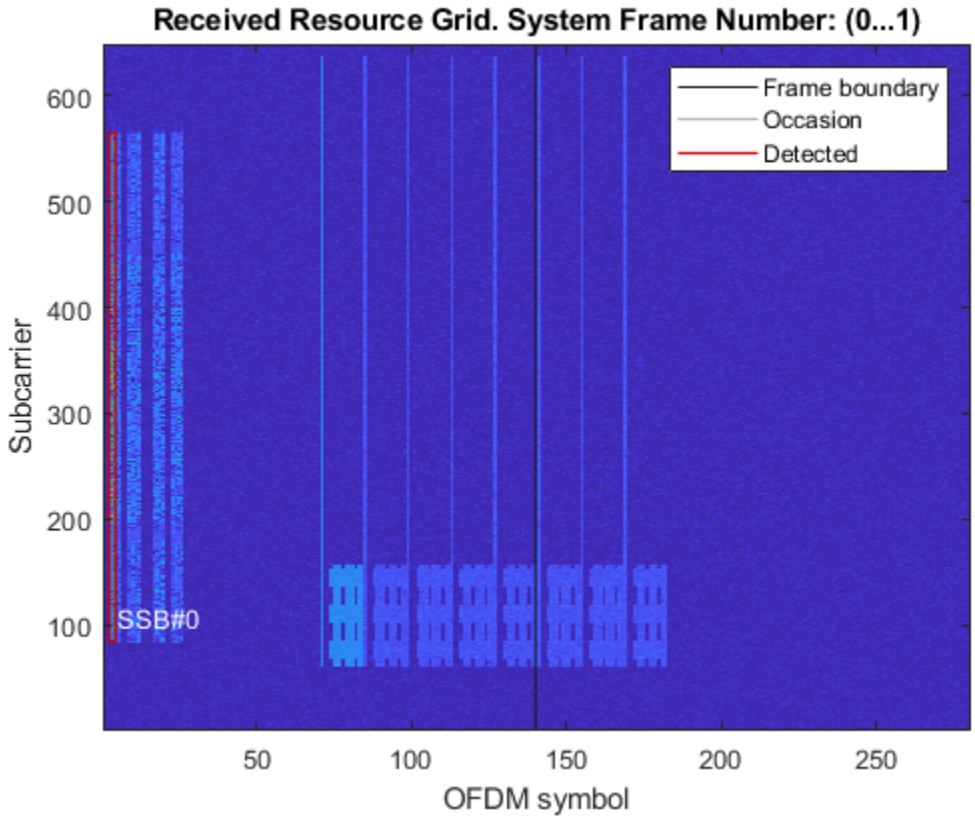
Unlike the SS burst, control and data channels must be aligned in frequency with their common resource block (CRB) raster. The value of KSSB indicates the frequency offset of the SSB from that CRB raster. As the frequency correction process centered the SSB in frequency, apply a frequency shift determined by `k_SSB` to align data and control channels with their CRB before OFDM demodulation

```
k_SSB = initialSystemInfo.k_SSB;
scsCommon = initialSystemInfo.SubcarrierSpacingCommon;
scsKSSB = kSSBSubcarrierSpacing(scsCommon);
kFreqShift = k_SSB*scsKSSB*1e3;
rxWaveform = rxWaveform.*exp(1i*2*pi*kFreqShift*(0:length(rxWaveform)-1)'/sampleRate);
```

```
% Adjust the symbol phase compensation frequency with the frequency shift
% introduced.
fPhaseComp = fPhaseComp - kFreqShift;
```

Adjust timing offset to the frame origin of the first frame that can schedule an SS/PBCH block. If the frame offset is negative, the first frame is incomplete. Add leading zeros to the waveform to align the wavefom to that frame.

```
[frameOffset,nLeadingFrames] = hTimingOffsetToFirstFrame(timingOffset,refBurst,ssbIndex,nHalfFra

% Add leading zeros
zeroPadding = zeros(-min(frameOffset,0),size(rxWaveform,2));
rxWaveform = [zeroPadding; rxWaveform(1+max(frameOffset,0):end,:)];

% Determine the number of resource blocks and subcarrier spacing for OFDM
% demodulation of CORESET 0.
nrb = hCORESET0DemodulationBandwidth(initialSystemInfo,scsSSB,minChannelBW);

if sampleRate < nrb*12*scsCommon*1e3
    disp(['SIB1 recovery cannot continue. CORESET 0 resources are beyond '...
          'the frequency limits of the received waveform for the sampling rate configured.']);
    return;
end

% OFDM demodulate received waveform with common subcarrier spacing
nSlot = 0;
rxGrid = nrOFDMDemodulate(rxWaveform, nrb, scsCommon, nSlot,...
                          'SampleRate',sampleRate,'CarrierFrequency',fPhaseComp);

% Display OFDM resource grid and highlight strongest SS block
plotResourceGrid(rxGrid,refBurst,initialSystemInfo,nLeadingFrames,ssbIndex,nHalfFrame)
```



Demodulation of PDCCH and Downlink Control Information Decoding

To blindly search for system information DCI messages in CORESET/SS, the receiver performs these steps:

- Determination of PDCCH monitoring occasions and extraction of the OFDM resource grid containing control information.
- Configuration of CORESET 0, Search spaces and PDCCH.
- Blind search for Format 1_0 DCI messages.

The receiver determines the PDCCH monitoring occasions through a slot and OFDM symbol offset from the location of the SS block detected, as described in TS 38.213 Tables 13-11 and 13-12 [5] . The receiver adjusts the system frame number to account for the new synchronization time to the first frame available in the waveform.

```
initialSystemInfo.NFrame = mod(initialSystemInfo.NFrame - nLeadingFrames,1024);
numRxSym = size(rxGrid,2);
[csetSubcarriers,monSlots,monSlotsSym,ssStartSym] = hPDCCH0MonitoringResources(initialSystemInfo

% Check if search space is beyond end of waveform
if isempty(monSlotsSym)
    disp('Search space slot is beyond end of waveform.');
    return;
end

% Extract slots containing strongest PDCCH from the received grid
rxMonSlotGrid = rxGrid(csetSubcarriers,monSlotsSym,:);
```

Configure CORESET, search space, and other PDCCH parameters. CORESET resources and search spaces are configured according to TS 38.213 Section 13 Tables 13-1 through 13-15 [5]. CCE-to-REG interleaved mapping parameters (REGBundleSize = 6, InterleaverSize = 2, and ShiftIndex = NCellID) are described in TS 38.211 Section 7.3.2.2 [3]. For CORESET 0, the BWP is the size of the CORESET as described in TS 38.212 Section 7.3.1.0 [4]. The PDCCH scrambling parameters are nRNTI = 0 and nID = NCellID as described in TS 38.211 Section 7.3.2.3 [3].

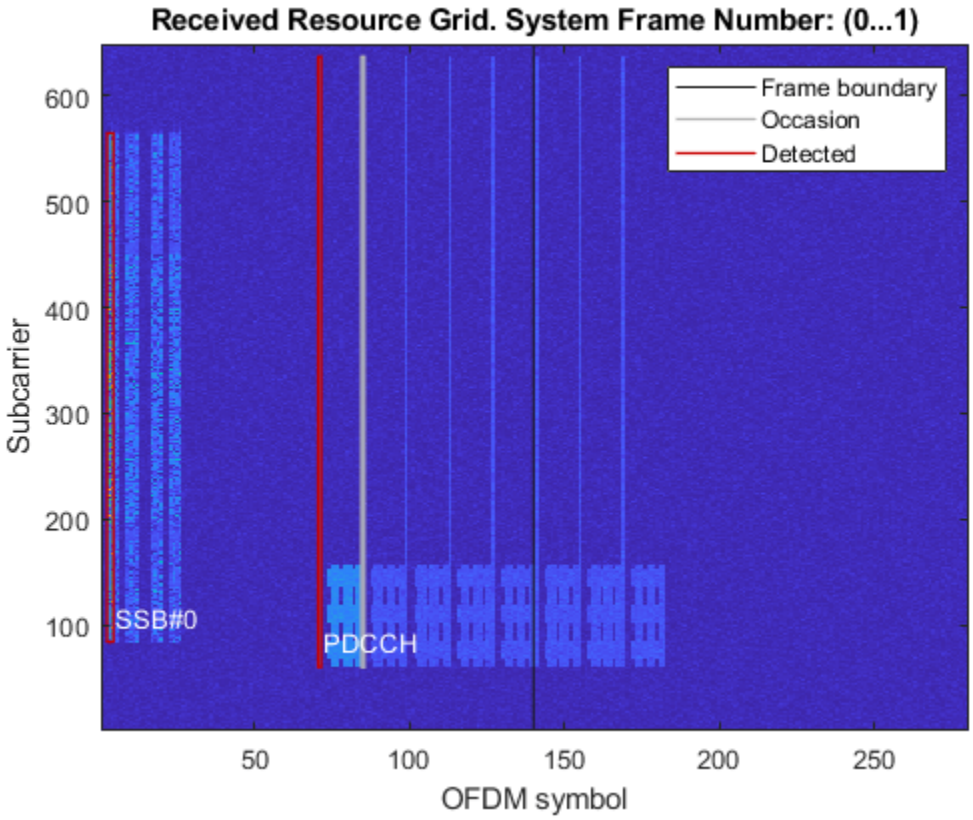
```
scsPair = [scsSSB scsCommon];
[pdcch,csetPattern] = hPDCCH0Configuration(ssbIndex,initialSystemInfo,scsPair,ncellid,minChannelB

% Configure the carrier to span the BWP (CORESET 0)
carrier = hCarrierConfigSIB1(ncellid,initialSystemInfo,pdcch);
```

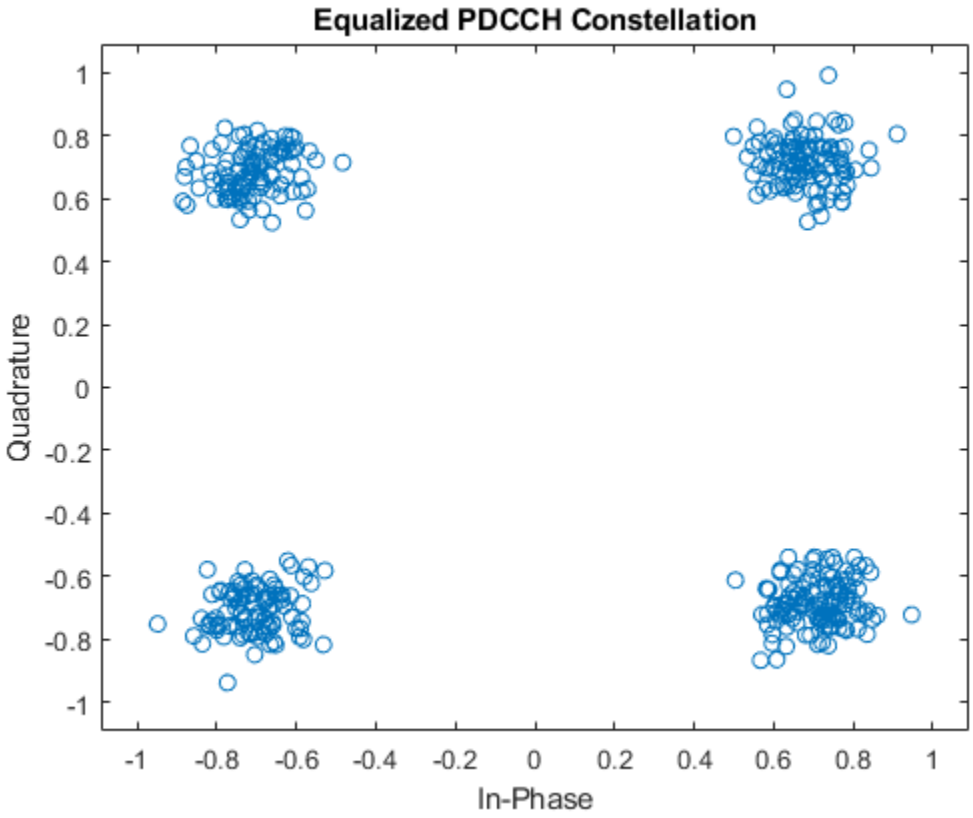
Search for DCI messages. UE decodes the received PDCCH symbols blindly by monitoring all PDCCH candidates for every aggregation level using the SI-RNTI to identify the right candidate (or instance).

```
% Specify DCI message with Format 1_0 scrambled with SI-RNTI (TS 38.212
% Section 7.3.1.2.1)
dci = DCIFormat1_0_SIRNTI(pdcch.NSizeBWP);

disp(' -- Downlink control information message search in PDCCH -- ');

symbolsPerSlot = 14;
siRNTI = 65535; % TS 38.321 Table 7.1-1
dciCRC = true;
mSlotIdx = 0;
% Loop over all monitoring slots
while (mSlotIdx < length(monSlots)) && dciCRC

    % Update slot number to next monitoring slot
    carrier.NSlot = monSlots(mSlotIdx+1);

    % Get PDCCH candidates according to TS 38.213 Section 10.1
    [pdcchInd,pdcchDmrsSym,pdcchDmrsInd] = nrPDCCHSpace(carrier,pdcch);

    % Extract resource grid for this monitoring slot and normalize
    rxSlotGrid = rxMonSlotGrid(:,(1:symbolsPerSlot) + symbolsPerSlot*mSlotIdx,:);
    rxSlotGrid = rxSlotGrid/max(abs(rxSlotGrid(:)));

    % Proceed to blind decoding only if the PDCCH REs are not zero.
    notZero = any(cellfun(@(x)any(rxSlotGrid(x),'all'),pdcchInd));

    % Loop over all supported aggregation levels
    aLevIdx = 1;
    while (aLevIdx <= 5) && dciCRC && notZero
        % Loop over all candidates at each aggregation level in SS
        cIdx = 1;
        numCandidatesAL = pdcch.SearchSpace.NumCandidates(aLevIdx);
        while (cIdx <= numCandidatesAL) && dciCRC
            % Channel estimation using PDCCH DM-RS
            [hest,nVar,pdcchHestInfo] = nrChannelEstimate(rxSlotGrid,pdcchDmrsInd{aLevIdx}(:,cIdx

            % Equalization and demodulation of PDCCH symbols
            [pdcchRxSym,pdcchHest] = nrExtractResources(pdcchInd{aLevIdx}(:,cIdx),rxSlotGrid,hest
            pdcchEqSym = nrEqualizeMMSE(pdcchRxSym,pdcchHest,nVar);
            dcicw = nrPDCCHDecode(pdcchEqSym,pdcch.DMRSScramblingID,pdcch.RNTI,nVar);

            % DCI message decoding
```

```
            polarListLength = 8;
            [dcibits,dciCRC] = nrDCIDecode(dcicw,dci.Width,polarListLength,siRNTI);

            if dciCRC == 0
                disp([' Decoded PDCCH candidate #' num2str(cIdx) ' at aggregation level ' num2st
            end
            cIdx = cIdx + 1;
        end
        aLevIdx = aLevIdx+1;
    end
    mSlotIdx = mSlotIdx+1;
end
mSlotIdx = mSlotIdx-1;
monSlotsSym = monSlotsSym(mSlotIdx*symbolsPerSlot + (1:symbolsPerSlot));

% Highlight CORESET 0/SS occasions in resource grid
highlightCORESET0SS(csetSubcarriers,monSlots,monSlots(mSlotIdx+1),pdcch,dciCRC)

if dciCRC
    disp(' DCI decoding failed.');
    return
end

% Calculate RMS PDCCH EVM
pdcchRef = nrPDCCH(double(dcicw<0),pdcch.DMRSScramblingID,pdcch.RNTI);
evm = comm.EVM;
pdcchEVMrms = evm(pdcchRef,pdcchEqSym);

% Display calculated EVM
disp([' PDCCH RMS EVM: ' num2str(pdcchEVMrms,'%0.3f') '%']);
disp([' PDCCH CRC: ' num2str(dciCRC)]);

% Plot received PDCCH constellation after equalization
figure;
plot(pdcchEqSym,'o');
xlabel('In-Phase'); ylabel('Quadrature')
title('Equalized PDCCH Constellation');
m = max(abs([real(pdcchEqSym(:)); imag(pdcchEqSym(:))])) * 1.1;
axis([-m m -m m]);
```

```
 -- Downlink control information message search in PDCCH --
 Decoded PDCCH candidate #1 at aggregation level 8
 PDCCH RMS EVM: 10.759%
 PDCCH CRC: 0
```


Received Resource Grid. System Frame Number: (0...1)
Frame boundary
Occasion
Detected
SSB#0
PDCCH
Subcarrier
OFDM symbol
600
500
400
300
200
100
50
100
150
200
250



Demodulation of PDSCH, Decoding of DL-SCH and SIB1 Extraction

To recover the first system information block, the receiver performs these steps:

- Determination of PDSCH configuration using cell ID, BCH information, and DCI
- Channel estimation, equalization and demodulation of PDSCH symbols
- Decoding of DL-SCH and SIB1 extraction

```
% Build DCI message structure
dci = fromBits(dci,dcibits);

% Get PDSCH configuration from cell ID, BCH information, and DCI
[pdsch,K0] = hSIB1PDSCHConfiguration(dci,pdcch.NSizeBWP,initialSystemInfo.DMRSTypeAPosition,csetI

% For CORESET pattern 2, the gNodeB can allocate PDSCH in the next slot,
% which is indicated by the slot offset K_0 signaled by DCI. For more
% information, see TS 38.214 Table 5.1.2.1.1-4.
carrier.NSlot = carrier.NSlot + K0;
monSlotsSym = monSlotsSym+symbolsPerSlot*K0;

if K0 > 0
    % Display the OFDM grid of the slot containing associated PDSCH
    figure;
    imagesc(abs(rxGrid(csetSubcarriers,monSlotsSym,1))); axis xy
    xlabel('OFDM symbol');
    ylabel('subcarrier');
    title('Slot Containing PDSCH (Slot Offset K_0 = 1)');
```

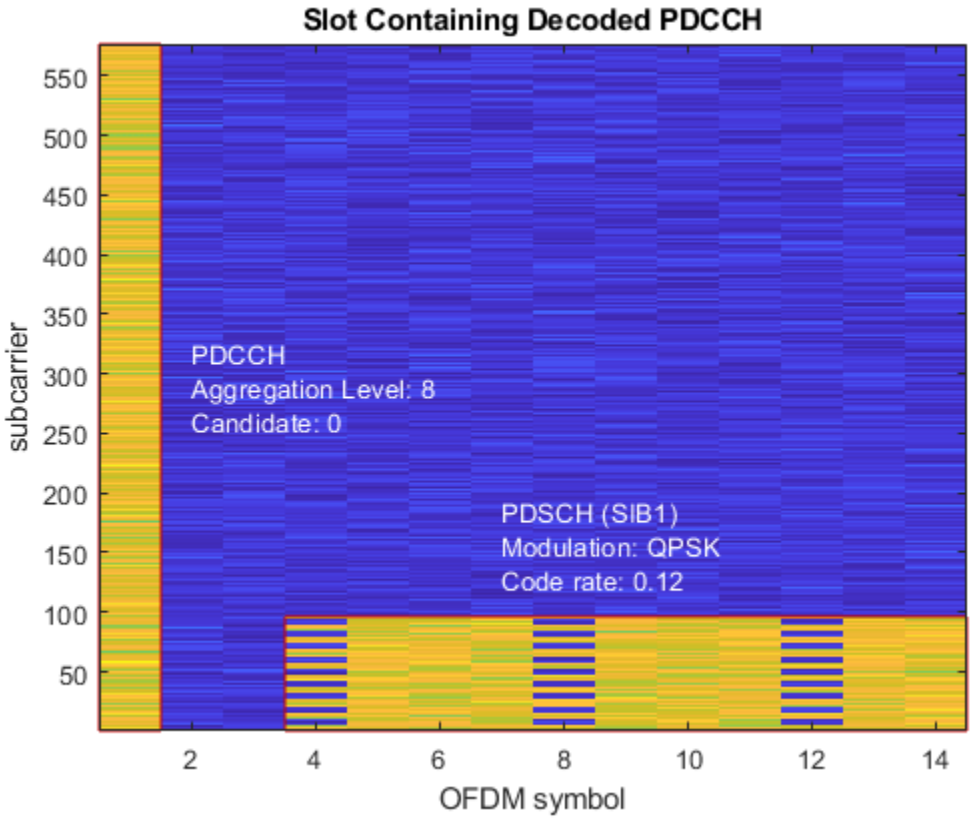
```
end

% PDSCH channel estimation and equalization using PDSCH DM-RS
pdschDmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
pdschDmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
```

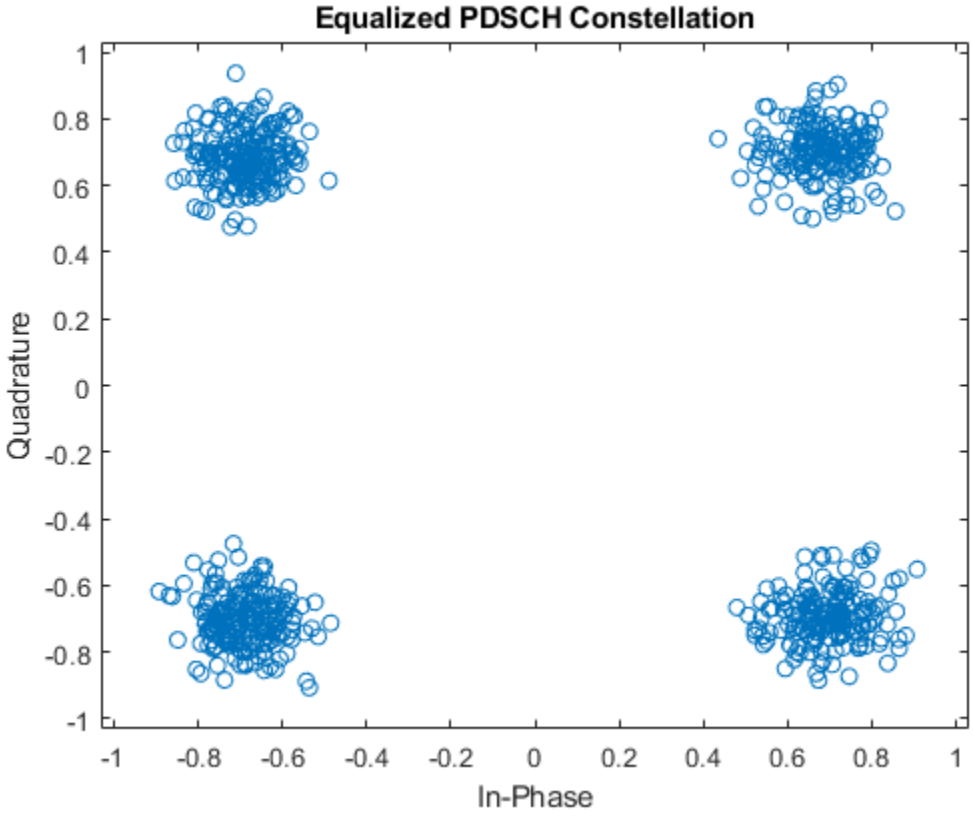
To compensate for the negative effects of a carrier frequency mismatch in symbol phase compensation and channel estimation, the receiver OFDM demodulates the waveform with a set of carrier frequencies over a search bandwidth around `fPhaseComp`. The search finishes when DL-SCH decoding succeeds or the last frequency has been reached. The minimum search bandwidths that produce equal symbol phase compensation are 1920, 3840, 7680, and 15360 kHz for common subcarrier spacings 15, 30, 60, and 120 kHz, respectively. Increase the search bandwidth up to these values when SIB1 decoding fails and the equalized PDSCH symbols result in a heavily distorted and rotated constellation.

```
disp(' -- PDSCH demodulation and DL-SCH decoding -- ')

mu = log2(scsCommon/15);
bw = 2^mu*100;   % Search bandwidth (kHz)
freqStep = 2^mu; % Frequency step (kHz)
freqSearch = -bw/2:freqStep:bw/2-freqStep;
[~,fSearchIdx] = sort(abs(freqSearch)); % Sort frequencies from center
freqSearch = freqSearch(fSearchIdx);

for fpc = fPhaseComp + 1e3*freqSearch

    % OFDM demodulate received waveform
    nSlot = 0;
    rxGrid = nrOFDMDemodulate(rxWaveform, nrb, scsCommon, nSlot,...
                              'SampleRate',sampleRate,'CarrierFrequency',fpc);

    % Extract monitoring slot from the received grid
    rxSlotGrid = rxGrid(csetSubcarriers,monSlotsSym,:);
    rxSlotGrid = rxSlotGrid/max(abs(rxSlotGrid(:))); % Normalization of received RE magnitude

    % Channel estimation and equalization of PDSCH symbols
    [hest,nVar,pdschHestInfo] = nrChannelEstimate(rxSlotGrid,pdschDmrsIndices,pdschDmrsSymbols);
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);
    [pdschRxSym,pdschHest] = nrExtractResources(pdschIndices,rxSlotGrid,hest);
    pdschEqSym = nrEqualizeMMSE(pdschRxSym,pdschHest,nVar);

    % PDSCH demodulation
    cw = nrPDSCHDecode(carrier,pdsch,pdschEqSym,nVar);

    % Create and configure DL-SCH decoder with target code rate and
    % transport block size
    decodeDLSCH = nrDLSCHDecoder;
    decodeDLSCH.LDPCDecodingAlgorithm = 'Normalized min-sum';
    Xoh_PDSCH = 0; % TS 38.214 Section 5.1.3.2
    tcr = hMCS(dci.ModulationCoding);
    NREPerPRB = pdschIndicesInfo.NREPerPRB;
    tbsLength = nrTBS(pdsch.Modulation,pdsch.NumLayers,length(pdsch.PRBSet),NREPerPRB,tcr,Xoh_PDS
    decodeDLSCH.TransportBlockLength = tbsLength;
    decodeDLSCH.TargetCodeRate = tcr;

    % Decode DL-SCH
    [sib1bits,sib1CRC] = decodeDLSCH(cw,pdsch.Modulation,pdsch.NumLayers,dci.RedundancyVersion);
```

```
    if sib1CRC == 0
        break;
    end

end

% Highlight PDSCH and PDSCH DM-RS in resource grid.
pdcch.AggregationLevel = 2^(aLevIdx-2);
pdcch.AllocatedCandidate = cIdx-1;
plotResourceGridSIB1(rxSlotGrid,carrier,pdcch,pdsch,tcr,K0);

% Plot received PDSCH constellation after equalization
figure;
plot(pdschEqSym,'o');
xlabel('In-Phase'); ylabel('Quadrature')
title('Equalized PDSCH Constellation');
m = max(abs([real(pdschEqSym(:)); imag(pdschEqSym(:))])) * 1.1;
axis([-m m -m m]);

% Calculate RMS PDSCH EVM, including normalization of PDSCH symbols for any
% offset between DM-RS and PDSCH power
pdschRef = nrPDSCH(carrier,pdsch,double(cw{1}<0));
evm = comm.EVM;
pdschEVMrms = evm(pdschRef,pdschEqSym/sqrt(var(pdschEqSym)));

% Display PDSCH EVM and DL-SCH CRC
disp([' PDSCH RMS EVM: ' num2str(pdschEVMrms,'%0.3f') '%']);
disp([' PDSCH CRC: ' num2str(sib1CRC)]);

if sib1CRC == 0
    disp(' SIB1 decoding succeeded.');
else
    disp(' SIB1 decoding failed.');
end
```

```
 -- PDSCH demodulation and DL-SCH decoding --
 PDSCH RMS EVM: 10.832%
 PDSCH CRC: 0
 SIB1 decoding succeeded.
```


Slot Containing Decoded PDCCH
subcarrier
OFDM symbol
PDCCH
Aggregation Level: 8
Candidate: 0
PDSCH (SIB1)
Modulation: QPSK
Code rate: 0.12
550
500
450
400
350
300
250
200
150
100
50
2
4
6
8
10
12
14



References

1. 3GPP TS 38.101-1. "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
2. 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
3. 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
4. 3GPP TS 38.212. "NR; Multiplexing and channel coding." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
5. 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
6. 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
7. 3GPP TS 38.321. "NR; Medium Access Control (MAC) protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local functions

```
function present = isCORESET0Present(ssbBlockPattern,kSSB)

    switch ssbBlockPattern
        case {'Case A','Case B','Case C'} % FR1
```

```
            kssb_max = 23;
        case {'Case D','Case E'} % FR2
            kssb_max = 11;
    end
    if (kSSB <= kssb_max)
        present = true;
    else
        present = false;
    end

end

function [timingOffset,nLeadingFrames] = hTimingOffsetToFirstFrame(offset,burst,ssbIdx,nHalfFrame

    % As the symbol lengths are measured in FFT samples, scale the symbol
    % lengths to account for the receiver sample rate. Non-integer delays
    % are approximated at the end of the process.
    scs = hSSBurstSubcarrierSpacing(burst.BlockPattern);
    ofdmInfo = nrOFDMInfo(1,scs,'SampleRate',sampleRate); % smallest FFT size for SCS-SR
    srRatio = sampleRate/(scs*1e3*ofdmInfo.Nfft);
    symbolLengths = ofdmInfo.SymbolLengths*srRatio;

    % Adjust timing offset to the start of the SS block. This step removes
    % the extra offset introduced in the reference grid during PSS search,
    % which contained the PSS in the second OFDM symbol.
    offset = offset + symbolLengths(1);

    % Timing offset is adjusted so that the received grid starts at the
    % frame head i.e. adjust the timing offset for the difference between
    % the first symbol of the strongest SSB, and the start of the frame
    burstStartSymbols = hSSBurstStartSymbols(burst.BlockPattern,burst.L_max); % Start symbols in
    ssbFirstSym = burstStartSymbols(ssbIdx+1); % 0-based

    % Adjust for whole subframes
    symbolsPerSubframe = length(symbolLengths);
    subframeOffset = floor(ssbFirstSym/symbolsPerSubframe);
    samplesPerSubframe = sum(symbolLengths);
    timingOffset = offset - (subframeOffset*samplesPerSubframe);

    % Adjust for remaining OFDM symbols
    symbolOffset = mod(ssbFirstSym,symbolsPerSubframe);
    timingOffset = timingOffset - sum(symbolLengths(1:symbolOffset));

    % The first OFDM symbol of the SSB is defined with respect to the
    % half-frame where it is transmitted. Adjust for half-frame offset
    timingOffset = timingOffset - nHalfFrame*5*samplesPerSubframe;

    % Adjust offset to the first frame in the waveform that is scheduled
    % for SSB transmission.
    repetitions = ceil(timingOffset/(20*samplesPerSubframe));
    timingOffset = round(timingOffset - repetitions*20*samplesPerSubframe);

    % Calculate the number of leading frames before the detected one
    nLeadingFrames = 2*repetitions;

end

function initSystemInfo = initSystemInfo(mib,sfn4lsb,k_SSB,L_max)
```

```
    % Create set of subcarrier spacings signaled by the 7th bit of the
    % decoded MIB, the set is different for FR1 (L_max=4 or 8) and FR2
    % (L_max=64)
    if (L_max==64)
        scsCommon = [60 120];
    else
        scsCommon = [15 30];
    end

    initSystemInfo = struct();
    initSystemInfo.NFrame = mib.systemFrameNumber*2^4 + bit2int(sfn4lsb,4);
    initSystemInfo.SubcarrierSpacingCommon = scsCommon(mib.subCarrierSpacingCommon + 1);
    initSystemInfo.k_SSB = k_SSB + mib.ssb_SubcarrierOffset;
    initSystemInfo.DMRSTypeAPosition = 2 + mib.dmrs_TypeA_Position;
    initSystemInfo.PDCCHConfigSIB1 = info(mib.pdcch_ConfigSIB1);
    initSystemInfo.CellBarred = mib.cellBarred;
    initSystemInfo.IntraFreqReselection = mib.intraFreqReselection;

end

function nrb = hCORESET0DemodulationBandwidth(sysInfo,scsSSB,minChannelBW)

    % Determine the OFDM demodulation bandwidth from CORESET 0 bandwidth
    cset0Idx = sysInfo.PDCCHConfigSIB1.controlResourceSetZero;
    scsCommon = sysInfo.SubcarrierSpacingCommon;
    scsPair = [scsSSB scsCommon];
    [csetNRB,~,csetFreqOffset] = hCORESET0Resources(cset0Idx,scsPair,minChannelBW,sysInfo.k_SSB)

    % Calculate a suitable bandwidth in RB that includes CORESET 0 in
    % received waveform.
    c0 = csetFreqOffset + 10*scsSSB/scsCommon;  % CORESET frequency offset from carrier center
    nrb = 2*max(c0,csetNRB-c0)+2;               % Number of RB to cover CORESET 0

end

function [k,slots,slotSymbols,ssStartSym] = hPDCCH0MonitoringResources(systemInfo,scsSSB,minChan

    cset0Idx = systemInfo.PDCCHConfigSIB1.controlResourceSetZero;
    scsCommon = systemInfo.SubcarrierSpacingCommon;
    scsPair = [scsSSB scsCommon];
    k_SSB = systemInfo.k_SSB;
    [c0NRB,c0Duration,c0FreqOffset,c0Pattern] = hCORESET0Resources(cset0Idx,scsPair,minChannelBW

    ssIdx = systemInfo.PDCCHConfigSIB1.searchSpaceZero;
    [ssSlot,ssStartSym,isOccasion] = hPDCCH0MonitoringOccasions(ssIdx,ssbIndex,scsPair,c0Pattern

    % PDCCH monitoring occasions associated to different SS blocks can be
    % in different frames. If there are no monitoring occasions in this
    % frame, there must be one in the next one. Adjust the slots associated
    % to the search space by one frame if needed.
    slotsPerFrame = 10*scsCommon/15;
    ssSlot = ssSlot + (~isOccasion)*slotsPerFrame;

    % For FR1, UE monitors PDCCH in the Type0-PDCCH CSS over two consecutive
    % slots for CORESET pattern 1
    monSlotsPerPeriod = 1 + (c0Pattern==1);
```

```
        % Calculate 1-based subscripts of the subcarriers and OFDM symbols for
        % the slots containing the PDCCH0 associated to the detected SS block
        % in this and subsequent 2-frame blocks
        nrb = hCORESET0DemodulationBandwidth(systemInfo,scsSSB,minChannelBW);
        k = 12*(nrb-20*scsSSB/scsCommon)/2 - c0FreqOffset*12 + (1:c0NRB*12);

        symbolsPerSlot = 14;
        numRxSlots = ceil(numRxSym/symbolsPerSlot);
        slots = ssSlot + (0:monSlotsPerPeriod-1)' + (0:2*slotsPerFrame:(numRxSlots-ssSlot-1));
        slots = slots(:)';
        slotSymbols = slots*symbolsPerSlot + (1:symbolsPerSlot)';
        slotSymbols = slotSymbols(:)';

        % Remove monitoring symbols exceeding waveform limits
        slotSymbols(slotSymbols>numRxSym) = [];

        % Calculate the monitoring slots after removing symbols
        slots = (slotSymbols(1:symbolsPerSlot:end)-1)/symbolsPerSlot;

    end

    function scsKSSB = kSSBSubcarrierSpacing(scsCommon)
    % Subcarrier spacing of k_SSB, as defined in TS 38.211 Section 7.4.3.1

        if scsCommon > 30  % FR2
            scsKSSB = scsCommon;
        else
            scsKSSB = 15;
        end

    end

    function c = hCarrierConfigSIB1(ncellid,initSystemInfo,pdcch)

        c = nrCarrierConfig;
        c.SubcarrierSpacing = initSystemInfo.SubcarrierSpacingCommon;
        c.NStartGrid = pdcch.NStartBWP;
        c.NSizeGrid = pdcch.NSizeBWP;
        c.NSlot = pdcch.SearchSpace.SlotPeriodAndOffset(2);
        c.NFrame = initSystemInfo.NFrame;
        c.NCellID = ncellid;

    end

    function plotResourceGrid(rxGrid,refBurst,systemInfo,nLeadingFrames,ssbIndex,nHalfFrame)

        % Extract SSB and common SCS from reference SS burst and initial system
        % information
        scsSSB = hSSBurstSubcarrierSpacing(refBurst.BlockPattern);
        scsCommon = systemInfo.SubcarrierSpacingCommon;
        scsRatio = scsSSB/scsCommon;

        % Number of subcarriers, symbols and frames.
        [K,L] = size(rxGrid);
        symbolsPerSubframe = 14*scsCommon/15;
        numFrames = ceil(L/(10*symbolsPerSubframe));

        % Define colors and auxiliary plotting function
```

```
basePlotProps = {'LineStyle','-','LineWidth',1};
occasionColor = 0.7*[1 1 1];
detectionColor = [200,0,0]/255;

frameBoundaryColor = 0.1*[1 1 1];
boundingBox = @(y,x,h,w,varargin)rectangle('Position',[x+0.5 y-0.5 w h],basePlotProps{:},vara

% Create figure and display resource grid
figure;
imagesc(abs(rxGrid(:,:,1))); axis xy; hold on;

% Add vertical frame lines
x = repmat((0:numFrames-1)*10*symbolsPerSubframe,3,1);
x(3,:) = NaN;
y = repmat([0;K;NaN],1,numFrames);
plot(x(:),y(:),'Color',frameBoundaryColor);

% Determine frequency origin of the SSB in common numerology
ssbCenter = K/2;
halfSSB = 10*12*scsRatio;
scsKSSB = kSSBSubcarrierSpacing(scsCommon);
kSSBFreqOff = systemInfo.k_SSB*scsKSSB/scsCommon;
ssbFreqOrig = ssbCenter - halfSSB + kSSBFreqOff + 1;

% Determine time origin of the SSB in common numerology
ssbStartSymbols = hSSBurstStartSymbols(refBurst.BlockPattern,refBurst.L_max);
ssbStartSymbols = ssbStartSymbols + 5*symbolsPerSubframe*nHalfFrame;
ssbHeadSymbol = ssbStartSymbols(ssbIndex+1)/scsRatio;
ssbTailSymbol = floor((ssbStartSymbols(ssbIndex+1)+4)/scsRatio)-1;

% Draw bounding boxes around all SS/PBCH block occasions
w = ssbTailSymbol - ssbHeadSymbol + 1;
for i = 1:ceil(numFrames/2)
    s = ssbHeadSymbol + (i-1)*2*10*symbolsPerSubframe + 5*symbolsPerSubframe*nHalfFrame;
    if s <= (L - w)
        boundingBox(ssbFreqOrig,s,240*scsRatio,w,'EdgeColor',occasionColor);
    end
end

% Draw bounding box for detected SS/PBCH block
s = ssbHeadSymbol + nLeadingFrames*10*symbolsPerSubframe + 5*symbolsPerSubframe*nHalfFrame;
boundingBox(ssbFreqOrig,s,240*scsRatio,w,basePlotProps{:},'EdgeColor',detectionColor)

% Add text next to detected SS/PBCH block
str = sprintf('SSB#%d',ssbIndex);
text(s+w+1,ssbFreqOrig+24,0,str,'FontSize',10,'Color','w')

% Create legend. Since rectangles don't show up in legend, create a
% placeholder for bounding boxes.
plot(NaN,NaN,basePlotProps{:},'Color',occasionColor);
plot(NaN,NaN,basePlotProps{:},'Color',detectionColor);
legend('Frame boundary','Occasion','Detected')
xlabel('OFDM symbol'); ylabel('Subcarrier');

% Add title including frame numbers
firstNFrame = systemInfo.NFrame - nLeadingFrames;
nframes = mod(firstNFrame + (0:numFrames-1),1024);
sfns = sprintf('(%d...%d)',nframes(1),nframes(end));
```

```
    title(['Received Resource Grid. System Frame Number: ' sfns]);

end

function highlightCORESET0SS(csetSubcarriers,monSlots,detSlot,pdcch,dciCRC)

    ssFirstSym = pdcch.SearchSpace.StartSymbolWithinSlot;
    csetDuration = pdcch.CORESET.Duration;

    % Define colors and plotting function
    basePlotProps = {'LineStyle','-','LineWidth',1};
    occasionColor = 0.7*[1 1 1];
    detectionColor = [200,0,0]/255;
    boundingBox = @(y,x,h,w,varargin)rectangle('Position',[x+0.5 y-0.5 w h],basePlotProps{:},vara

    % Highlight all CORESET 0/SS occasions related to the detected SSB
    k0 = csetSubcarriers(1);
    K = length(csetSubcarriers);
    ssSym = 14*monSlots + ssFirstSym ;
    for i = 1:length(ssSym)
        boundingBox(k0,ssSym(i),K,csetDuration,'EdgeColor',occasionColor);
    end

    if dciCRC == 0
        % Highlight decoded PDCCH
        ssSym = 14*detSlot + ssFirstSym;
        boundingBox(k0,ssSym,K,csetDuration,'EdgeColor',detectionColor);

        % Add text next to decoded PDCCH
        text(ssSym+csetDuration+1,k0+24,0,'PDCCH','FontSize',10,'Color','w')
    end

end

function plotResourceGridSIB1(slotGrid,carrier,pdcch,pdsch,tcr,K0)

    % Display the OFDM grid of the slot containing decoded PDCCH
    figure;
    imagesc(abs(slotGrid(:,:,1))); axis xy
    xlabel('OFDM symbol');
    ylabel('subcarrier');
    title('Slot Containing Decoded PDCCH');

    aggregationLevelIndex = log2(pdcch.AggregationLevel)+1;
    candidate = pdcch.AllocatedCandidate;

    % Define auxiliary plotting function
    color = [200,0,0]/255;
    boundingBox = @(y,x,h,w,varargin)rectangle('Position',[x+0.5 y-0.5 w h],'EdgeColor',color,va

    % Highlight PDCCH in resource grid
    carrier.NSlot = carrier.NSlot - K0; % Substract slot offset K0 for subscripts calculations
    subsPdcch = nrPDCCHSpace(carrier,pdcch,'IndexStyle','Subs');
    subsPdcch = double(subsPdcch{aggregationLevelIndex}(:,:,candidate));
    x = min(subsPdcch(:,2))-1; X = max(subsPdcch(:,2))-x;
    y = min(subsPdcch(:,1)); Y = max(subsPdcch(:,1))-y+1;
    boundingBox(y,x,Y,X);
    str = sprintf('PDCCH \nAggregation Level: %d\nCandidate: %d',2.^(aggregationLevelIndex-1),ca
```

```
    text(x+X+1,y+Y/2,0,str,'FontSize',10,'Color','w')

    % Highlight PDSCH and PDSCH DM-RS in resource grid
    carrier.NSlot = carrier.NSlot + K0; % Add back slot offset K0 for subscripts calculations
    subsPdschSym = double(nrPDSCHIndices(carrier,pdsch,'IndexStyle','subscript'));
    subsPdschDmrs = double(nrPDSCHDMRSIndices(carrier,pdsch,'IndexStyle','subscript'));
    subsPdsch = [subsPdschSym;subsPdschDmrs];
    x = min(subsPdsch(:,2))-1; X = max(subsPdsch(:,2))-x;
    y = min(subsPdsch(:,1)); Y = max(subsPdsch(:,1))-y+1;
    boundingBox(y,x,Y,X);
    str = sprintf('PDSCH (SIB1) \nModulation: %s\nCode rate: %.2f',pdsch.Modulation,tcr);
    text(x+4,y+Y+60,0, str,'FontSize',10,'Color','w')

end
```

See Also

More About

- “Synchronization Signal Blocks and Bursts”
- “NR SSB Beam Sweeping” on page 1-98
- “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113

NR PDSCH Throughput

This reference simulation shows how to measure the physical downlink shared channel (PDSCH) throughput of a 5G New Radio (NR) link, as defined by the 3GPP NR standard. The example implements the PDSCH and downlink shared channel (DL-SCH). The transmitter model includes PDSCH demodulation reference signals (DM-RS) and PDSCH phase tracking reference signals (PT-RS). The example supports both clustered delay line (CDL) and tapped delay line (TDL) propagation channels. You can perform perfect or practical synchronization and channel estimation. To reduce the total simulation time, you can execute the SNR points in the SNR loop in parallel by using the Parallel Computing Toolbox™.

Introduction

This example measures the PDSCH throughput of a 5G link, as defined by the 3GPP NR standard [1], [2], [3], [4].

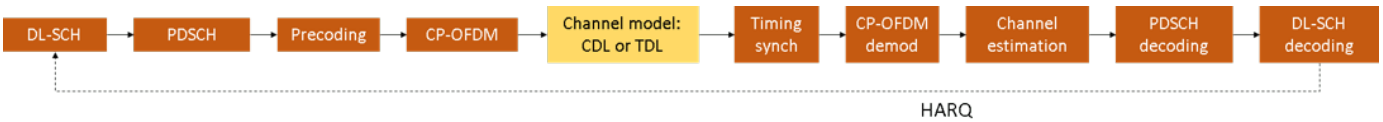
The example models these 5G NR features:

- DL-SCH transport channel coding
- Multiple codewords, dependent on the number of layers
- PDSCH, PDSCH DM-RS, and PDSCH PT-RS generation
- Variable subcarrier spacing and frame numerologies (2^n * 15 kHz)
- Normal and extended cyclic prefix
- TDL and CDL propagation channel models

Other features of the simulation are:

- PDSCH subband precoding using SVD
- CP-OFDM modulation
- Slot wise and non slot wise PDSCH and DM-RS mapping
- Perfect or practical synchronization and channel estimation
- HARQ operation with 16 processes
- The example uses a single bandwidth part across the whole carrier

The figure shows the implemented processing chain. For clarity, the DM-RS and PT-RS generation are omitted.



For a more detailed explanation of the steps implemented in this example, see “Model 5G NR Communication Links” and “DL-SCH and PDSCH Transmit and Receive Processing Chain”.

This example supports both wideband and subband precoding. The precoding matrix is determined using SVD by averaging the channel estimate across all PDSCH PRBs in the allocation (wideband case) or in the subband.

To reduce the total simulation time, you can use the Parallel Computing Toolbox to execute the SNR points of the SNR loop in parallel.

Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10ms frames. A large number of NFrames should be used to produce meaningful throughput results. Set the SNR points to simulate. The SNR for each layer is defined per RE, and it includes the effect of signal and noise across all antennas. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
simParameters = struct();       % Clear simParameters variable to contain all key simulation para
simParameters.NFrames = 2;      % Number of 10 ms frames
simParameters.SNRIn = [-5 0 5]; % SNR range (dB)
```

Channel Estimator Configuration

The logical variable `PerfectChannelEstimator` controls channel estimation and synchronization behavior. When set to `true`, perfect channel estimation and synchronization is used. Otherwise, practical channel estimation and synchronization is used, based on the values of the received PDSCH DM-RS.

```
simParameters.PerfectChannelEstimator = true;
```

Simulation Diagnostics

The variable `DisplaySimulationInformation` controls the display of simulation information such as the HARQ process ID used for each subframe. In case of CRC error, the value of the index to the RV sequence is also displayed.

```
simParameters.DisplaySimulationInformation = true;
```

The `DisplayDiagnostics` flag enables the plotting of the EVM per layer. This plot monitors the quality of the received signal after equalization. The EVM per layer figure shows:

- The EVM per layer per slot, which shows the EVM evolving with time.
- The EVM per layer per resource block, which shows the EVM in frequency.

This figure evolves with the simulation and is updated with each slot. Typically, low SNR or channel fades can result in decreased signal quality (high EVM). The channel affects each layer differently, therefore, the EVM values may differ across layers.

In some cases, some layers can have a much higher EVM than others. These low-quality layers can result in CRC errors. This behavior may be caused by low SNR or by using too many layers for the channel conditions. You can avoid this situation by a combination of higher SNR, lower number of layers, higher number of antennas, and more robust transmission (lower modulation scheme and target code rate).

```
simParameters.DisplayDiagnostics = false;
```

Carrier and PDSCH Configuration

Set the key parameters of the simulation. These include:

- The bandwidth in resource blocks (12 subcarriers per resource block).

- Subcarrier spacing: 15, 30, 60, 120 (kHz)
- Cyclic prefix length: normal or extended
- Cell ID
- Number of transmit and receive antennas

A substructure containing the DL-SCH and PDSCH parameters is also specified. This includes:

- Target code rate
- Allocated resource blocks (PRBSet)
- Modulation scheme: 'QPSK', '16QAM', '64QAM', '256QAM'
- Number of layers
- PDSCH mapping type
- DM-RS configuration parameters
- PT-RS configuration parameters

Other simulation wide parameters are:

- Propagation channel model delay profile (TDL or CDL)

```
% Set waveform type and PDSCH numerology (SCS and CP type)
simParameters.Carrier = nrCarrierConfig;         % Carrier resource grid configuration
simParameters.Carrier.NSizeGrid = 51;            % Bandwidth in number of resource blocks (51 RBs
simParameters.Carrier.SubcarrierSpacing = 30;    % 15, 30, 60, 120 (kHz)
simParameters.Carrier.CyclicPrefix = 'Normal';   % 'Normal' or 'Extended' (Extended CP is relevan
simParameters.Carrier.NCellID = 1;               % Cell identity

% PDSCH/DL-SCH parameters
simParameters.PDSCH = nrPDSCHConfig;      % This PDSCH definition is the basis for all PDSCH tran
simParameters.PDSCHExtension = struct();  % This structure is to hold additional simulation param

% Define PDSCH time-frequency resource allocation per slot to be full grid (single full grid BWP)
simParameters.PDSCH.PRBSet = 0:simParameters.Carrier.NSizeGrid-1;                % PDSCH PRB all
simParameters.PDSCH.SymbolAllocation = [0,simParameters.Carrier.SymbolsPerSlot];  % Starting symb
simParameters.PDSCH.MappingType = 'A';     % PDSCH mapping type ('A'(slot-wise),'B'(non slot-wise

% Scrambling identifiers
simParameters.PDSCH.NID = simParameters.Carrier.NCellID;
simParameters.PDSCH.RNTI = 1;

% PDSCH resource block mapping (TS 38.211 Section 7.3.1.6)
simParameters.PDSCH.VRBToPRBInterleaving = 0; % Disable interleaved resource mapping
simParameters.PDSCH.VRBBundleSize = 4;

% Define the number of transmission layers to be used
simParameters.PDSCH.NumLayers = 2;            % Number of PDSCH transmission layers

% Define codeword modulation and target coding rate
% The number of codewords is directly dependent on the number of layers so ensure that
% layers are set first before getting the codeword number
if simParameters.PDSCH.NumCodewords > 1                                % Multicodeword transmission
    simParameters.PDSCH.Modulation = {'16QAM','16QAM'};                % 'QPSK', '16QAM', '64QAM',
    simParameters.PDSCHExtension.TargetCodeRate = [490 490]/1024;      % Code rate used to calculate
else
    simParameters.PDSCH.Modulation = '16QAM';                          % 'QPSK', '16QAM', '64QAM',
```

```
    simParameters.PDSCHExtension.TargetCodeRate = 490/1024;         % Code rate used to calculate
end

% DM-RS and antenna port configuration (TS 38.211 Section 7.4.1.1)
simParameters.PDSCH.DMRS.DMRSPortSet = 0:simParameters.PDSCH.NumLayers-1; % DM-RS ports to use fo
simParameters.PDSCH.DMRS.DMRSTypeAPosition = 2;      % Mapping type A only. First DM-RS symbol po
simParameters.PDSCH.DMRS.DMRSLength = 1;             % Number of front-loaded DM-RS symbols (1(si
simParameters.PDSCH.DMRS.DMRSAdditionalPosition = 2; % Additional DM-RS symbol positions (max ran
simParameters.PDSCH.DMRS.DMRSConfigurationType = 2;  % DM-RS configuration type (1,2)
simParameters.PDSCH.DMRS.NumCDMGroupsWithoutData = 1;% Number of CDM groups without data
simParameters.PDSCH.DMRS.NIDNSCID = 1;               % Scrambling identity (0...65535)
simParameters.PDSCH.DMRS.NSCID = 0;                  % Scrambling initialization (0,1)

% PT-RS configuration (TS 38.211 Section 7.4.1.2)
simParameters.PDSCH.EnablePTRS = 0;                  % Enable or disable PT-RS (1 or 0)
simParameters.PDSCH.PTRS.TimeDensity = 1;            % PT-RS time density (L_PT-RS) (1, 2, 4)
simParameters.PDSCH.PTRS.FrequencyDensity = 2;       % PT-RS frequency density (K_PT-RS) (2 or 4)
simParameters.PDSCH.PTRS.REOffset = '00';            % PT-RS resource element offset ('00', '01'
simParameters.PDSCH.PTRS.PTRSPortSet = [];           % PT-RS antenna port, subset of DM-RS port s

% Reserved PRB patterns, if required (for CORESETs, forward compatibility etc)
simParameters.PDSCH.ReservedPRB{1}.SymbolSet = [];   % Reserved PDSCH symbols
simParameters.PDSCH.ReservedPRB{1}.PRBSet = [];      % Reserved PDSCH PRBs
simParameters.PDSCH.ReservedPRB{1}.Period = [];      % Periodicity of reserved resources

% Additional simulation and DL-SCH related parameters
%
% PDSCH PRB bundling (TS 38.214 Section 5.1.2.3)
simParameters.PDSCHExtension.PRGBundleSize = [];     % 2, 4, or [] to signify "wideband"
%
% HARQ process and rate matching/TBS parameters
simParameters.PDSCHExtension.XOverhead = 6*simParameters.PDSCH.EnablePTRS; % Set PDSCH rate match
simParameters.PDSCHExtension.NHARQProcesses = 16;    % Number of parallel HARQ processes to use
simParameters.PDSCHExtension.EnableHARQ = true;      % Enable retransmissions for each process, u

% LDPC decoder parameters
% Available algorithms: 'Belief propagation', 'Layered belief propagation', 'Normalized min-sum'
simParameters.PDSCHExtension.LDPCDecodingAlgorithm = 'Normalized min-sum';
simParameters.PDSCHExtension.MaximumLDPCIterationCount = 6;

% Define the overall transmission antenna geometry at end-points
% If using a CDL propagation channel then the integer number of antenna elements is
% turned into an antenna panel configured when the channel model object is created
simParameters.NTxAnts = 8;                           % Number of PDSCH transmission antennas (1,2,4
if simParameters.PDSCH.NumCodewords > 1              % Multi-codeword transmission
    simParameters.NRxAnts = 8;                       % Number of UE receive antennas (even number >=
else
    simParameters.NRxAnts = 2;                       % Number of UE receive antennas (1 or even numb
end

% Define the general CDL/TDL propagation channel parameters
simParameters.DelayProfile = 'CDL-C';   % Use CDL-C model (Urban macrocell model)
simParameters.DelaySpread = 300e-9;
simParameters.MaximumDopplerShift = 5;

% Cross-check the PDSCH layering against the channel geometry
validateNumLayers(simParameters);
```

The simulation relies on various pieces of information about the baseband waveform, such as sample rate.

```
waveformInfo = nrOFDMInfo(simParameters.Carrier); % Get information about the baseband waveform a
```

Propagation Channel Model Construction

Create the channel model object for the simulation. Both CDL and TDL channel models are supported [5].

```
% Constructed the CDL or TDL channel model object
if contains(simParameters.DelayProfile,'CDL','IgnoreCase',true)

    channel = nrCDLChannel; % CDL channel object

    % Turn the number of antennas into antenna panel array layouts. If
    % NTxAnts is not one of (1,2,4,8,16,32,64,128,256,512,1024), its value
    % is rounded up to the nearest value in the set. If NRxAnts is not 1 or
    % even, its value is rounded up to the nearest even number.
    channel = hArrayGeometry(channel,simParameters.NTxAnts,simParameters.NRxAnts);
    simParameters.NTxAnts = prod(channel.TransmitAntennaArray.Size);
    simParameters.NRxAnts = prod(channel.ReceiveAntennaArray.Size);
else
    channel = nrTDLChannel; % TDL channel object

    % Set the channel geometry
    channel.NumTransmitAntennas = simParameters.NTxAnts;
    channel.NumReceiveAntennas = simParameters.NRxAnts;
end

% Assign simulation channel parameters and waveform sample rate to the object
channel.DelayProfile = simParameters.DelayProfile;
channel.DelaySpread = simParameters.DelaySpread;
channel.MaximumDopplerShift = simParameters.MaximumDopplerShift;
channel.SampleRate = waveformInfo.SampleRate;
```

Get the maximum number of delayed samples by a channel multipath component. This is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This is required later to flush the channel filter to obtain the received signal.

```
chInfo = info(channel);
maxChDelay = chInfo.MaximumChannelDelay;
```

Processing Loop

To determine the throughput at each SNR point, analyze the PDSCH data per transmission instance using the following steps:

- *Update current HARQ process.* Check the transmission status for the given HARQ process to determine whether a retransmission is required. If that is not the case then generate new data.
- *Resource grid generation.* Perform channel coding by calling the `nrDLSCH` System object. The object operates on the input transport block and keeps an internal copy of the transport block in case a retransmission is required. Modulate the coded bits on the PDSCH by using the `nrPDSCH` function. Then apply precoding to the resulting signal.
- *Waveform generation.* OFDM modulate the generated grid.

- *Noisy channel modeling.* Pass the waveform through a CDL or TDL fading channel. Add AWGN. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.
- *Perform synchronization and OFDM demodulation.* For perfect synchronization, reconstruct the channel impulse response to synchronize the received waveform. For practical synchronization, correlate the received waveform with the PDSCH DM-RS. Then OFDM demodulate the synchronized signal.
- *Perform channel estimation.* For perfect channel estimation, reconstruct the channel impulse response and perform OFDM demodulation. For practical channel estimation, use the PDSCH DM-RS.
- *Perform equalization and CPE compensation.* MMSE equalize the estimated channel. Estimate the common phase error (CPE) by using the PT-RS symbols, then correct the error in each OFDM symbol within the range of reference PT-RS OFDM symbols.
- *Precoding matrix calculation.* Generate the precoding matrix W for the next transmission by using singular value decomposition (SVD).
- *Decode the PDSCH.* To obtain an estimate of the received codewords, demodulate and descramble the recovered PDSCH symbols for all transmit and receive antenna pairs, along with a noise estimate, by using the `nrPDSCHDecode` function.
- *Decode the downlink shared channel (DL-SCH) and update HARQ process with the block CRC error.* Pass the vector of decoded soft bits to the `nrDLSCHDecoder` System object. The object decodes the codeword and returns the block CRC error used to determine the throughput of the system.

```
% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(simParameters.SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(simParameters.SNRIn),1);

% Set up redundancy version (RV) sequence for all HARQ processes
if simParameters.PDSCHExtension.EnableHARQ
    % In the final report of RAN WG1 meeting #91 (R1-1719301), it was
    % observed in R1-1717405 that if performance is the priority, [0 2 3 1]
    % should be used. If self-decodability is the priority, it should be
    % taken into account that the upper limit of the code rate at which
    % each RV is self-decodable is in the following order: 0>3>2>1
    rvSeq = [0 2 3 1];
else
    % HARQ disabled - single transmission with RV=0, no retransmissions
    rvSeq = 0;
end

% Create DL-SCH encoder system object to perform transport channel encoding
encodeDLSCH = nrDLSCH;
encodeDLSCH.MultipleHARQProcesses = true;
encodeDLSCH.TargetCodeRate = simParameters.PDSCHExtension.TargetCodeRate;

% Create DL-SCH decoder system object to perform transport channel decoding
% Use layered belief propagation for LDPC decoding, with half the number of
% iterations as compared to the default for belief propagation decoding
decodeDLSCH = nrDLSCHDecoder;
decodeDLSCH.MultipleHARQProcesses = true;
decodeDLSCH.TargetCodeRate = simParameters.PDSCHExtension.TargetCodeRate;
decodeDLSCH.LDPCDecodingAlgorithm = simParameters.PDSCHExtension.LDPCDecodingAlgorithm;
decodeDLSCH.MaximumLDPCIterationCount = simParameters.PDSCHExtension.MaximumLDPCIterationCount;
```

```
for snrIdx = 1:numel(simParameters.SNRIn)      % comment out for parallel computing
% parfor snrIdx = 1:numel(simParameters.SNRIn) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'
% statement and uncomment the 'parfor' statement. If the Parallel Computing
% Toolbox is not installed, 'parfor' defaults to normal 'for' statement.
% Because parfor-loop iterations are executed in parallel in a
% nondeterministic order, the simulation information displayed for each SNR
% point can be intertwined. To switch off simulation information display,
% set the 'displaySimulationInformation' variable above to false

    % Reset the random number generator so that each SNR point will
    % experience the same noise realization
    rng('default');

    % Take full copies of the simulation-level parameter structures so that they are not
    % PCT broadcast variables when using parfor
    simLocal = simParameters;
    waveinfoLocal = waveformInfo;

    % Take copies of channel-level parameters to simplify subsequent parameter referencing
    carrier = simLocal.Carrier;
    pdsch = simLocal.PDSCH;
    pdschextra = simLocal.PDSCHExtension;
    decodeDLSCHLocal = decodeDLSCH;  % Copy of the decoder handle to help PCT classification of v
    decodeDLSCHLocal.reset();        % Reset decoder at the start of each SNR point
    pathFilters = [];

    % Prepare simulation for new SNR point
    SNRdB = simLocal.SNRIn(snrIdx);
    fprintf('\nSimulating transmission scheme 1 (%dx%d) and SCS=%dkHz with %s channel at %gdB SNR
        simLocal.NTxAnts,simLocal.NRxAnts,carrier.SubcarrierSpacing, ...
        simLocal.DelayProfile,SNRdB,simLocal.NFrames);

    % Specify the fixed order in which we cycle through the HARQ process IDs
    harqSequence = 0:pdschextra.NHARQProcesses-1;

    % Initialize the state of all HARQ processes
    harqEntity = HARQEntity(harqSequence,rvSeq,pdsch.NumCodewords);

    % Reset the channel so that each SNR point will experience the same
    % channel realization
    reset(channel);

    % Total number of slots in the simulation period
    NSlots = simLocal.NFrames * carrier.SlotsPerFrame;

    % Obtain a precoding matrix (wtx) to be used in the transmission of the
    % first transport block
    estChannelGrid = getInitialChannelEstimate(carrier,simLocal.NTxAnts,channel);
    newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGrid,pdschextra.PRGBundleSize);

    % Timing offset, updated in every slot for perfect synchronization and
    % when the correlation is strong for practical synchronization
    offset = 0;

    % Loop over the entire waveform length
```

```
for nslot = 0:NSlots-1

    % Update the carrier slot numbers for new slot
    carrier.NSlot = nslot;

    % Calculate the transport block sizes for the transmission in the slot
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);
    trBlkSizes = nrTBS(pdsch.Modulation,pdsch.NumLayers,numel(pdsch.PRBSet),pdschIndicesInfo

    % HARQ processing
    for cwIdx = 1:pdsch.NumCodewords
        % If new data for current process and codeword then create a new DL-SCH transport blo
        if harqEntity.NewData(cwIdx)
            trBlk = randi([0 1],trBlkSizes(cwIdx),1);
            setTransportBlock(encodeDLSCH,trBlk,cwIdx-1,harqEntity.HARQProcessID);
            % If new data because of previous RV sequence time out then flush decoder soft bu
            if harqEntity.SequenceTimeout(cwIdx)
                resetSoftBuffer(decodeDLSCHLocal,cwIdx-1,harqEntity.HARQProcessID);
            end
        end
    end

    % Encode the DL-SCH transport blocks
    codedTrBlocks = encodeDLSCH(pdsch.Modulation,pdsch.NumLayers, ...
        pdschIndicesInfo.G,harqEntity.RedundancyVersion,harqEntity.HARQProcessID);

    % Get precoding matrix (wtx) calculated in previous slot
    wtx = newWtx;

    % Create resource grid for a slot
    pdschGrid = nrResourceGrid(carrier,simLocal.NTxAnts);

    % PDSCH modulation and precoding
    pdschSymbols = nrPDSCH(carrier,pdsch,codedTrBlocks);
    [pdschAntSymbols,pdschAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGrid,pdschS

    % PDSCH mapping in grid associated with PDSCH transmission period
    pdschGrid(pdschAntIndices) = pdschAntSymbols;

    % PDSCH DM-RS precoding and mapping
    dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
    dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
    [dmrsAntSymbols,dmrsAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGrid,dmrsSymb
    pdschGrid(dmrsAntIndices) = dmrsAntSymbols;

    % PDSCH PT-RS precoding and mapping
    ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);
    ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
    [ptrsAntSymbols,ptrsAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGrid,ptrsSymb
    pdschGrid(ptrsAntIndices) = ptrsAntSymbols;

    % OFDM modulation
    txWaveform = nrOFDMModulate(carrier,pdschGrid);

    % Pass data through channel model. Append zeros at the end of the
    % transmitted waveform to flush channel content. These zeros take
    % into account any delay introduced in the channel. This is a mix
    % of multipath delay and implementation delay. This value may
```

```
% change depending on the sampling rate, delay profile and delay
% spread
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))]; %#ok<AGROW>
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

% Add AWGN to the received time domain waveform
% Normalize noise power by the IFFT size used in OFDM modulation,
% as the OFDM modulator applies this normalization to the
% transmitted waveform. Also normalize by the number of receive
% antennas, as the channel model applies this normalization to the
% received waveform, by default
SNR = 10^(SNRdB/10);
N0 = 1/sqrt(2.0*simLocal.NRxAnts*double(waveinfoLocal.Nfft)*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

if (simLocal.PerfectChannelEstimator)
    % Perfect synchronization. Use information provided by the
    % channel to find the strongest multipath component
    pathFilters = getPathFilters(channel);
    [offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
else
    % Practical synchronization. Correlate the received waveform
    % with the PDSCH DM-RS to give timing offset estimate 't' and
    % correlation magnitude 'mag'. The function
    % hSkipWeakTimingOffset is used to update the receiver timing
    % offset. If the correlation peak in 'mag' is weak, the current
    % timing estimate 't' is ignored and the previous estimate
    % 'offset' is used
    [t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols);
    offset = hSkipWeakTimingOffset(offset,t,mag);
    % Display a warning if the estimated timing offset exceeds the
    % maximum channel delay
    if offset > maxChDelay
        warning(['Estimated timing offset (%d) is greater than the maximum channel delay
            ' This will result in a decoding failure. This may be caused by low SNR,' ..
            ' or not enough DM-RS symbols to synchronize successfully.'],offset,maxChDela
    end
end
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to recreate the
% resource grid, including padding in the event that practical
% synchronization results in an incomplete slot being demodulated
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end

if (simLocal.PerfectChannelEstimator)
    % Perfect channel estimation, using the value of the path gains
    % provided by the channel. This channel estimate does not
    % include the effect of transmitter precoding
    estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sample

    % Get perfect noise estimate (from the noise realization)
    noiseGrid = nrOFDMDemodulate(carrier,noise(1+offset:end ,:));
```

```
        noiseEst = var(noiseGrid(:));

        % Get precoding matrix for next slot
        newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGrid,pdschextra.PRGBundleSize);

        % Get PDSCH resource elements from the received grid and
        % channel estimate
        [pdschRx,pdschHest,~,pdschHestIndices] = nrExtractResources(pdschIndices,rxGrid,estCh

        % Apply precoding to channel estimate
        pdschHest = hPRGPrecode(size(estChannelGrid),carrier.NStartGrid,pdschHest,pdschHestIn
    else
        % Practical channel estimation between the received grid and
        % each transmission layer, using the PDSCH DM-RS for each
        % layer. This channel estimate includes the effect of
        % transmitter precoding
        [estChannelGrid,noiseEst] = hSubbandChannelEstimate(carrier,rxGrid,dmrsIndices,dmrsSy

        % Average noise estimate across PRGs and layers
        noiseEst = mean(noiseEst,'all');

        % Get PDSCH resource elements from the received grid and
        % channel estimate
        [pdschRx,pdschHest] = nrExtractResources(pdschIndices,rxGrid,estChannelGrid);

        % Remove precoding from estChannelGrid prior to precoding
        % matrix calculation
        estChannelGridPorts = precodeChannelEstimate(carrier,estChannelGrid,conj(wtx));

        % Get precoding matrix for next slot
        newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGridPorts,pdschextra.PRGBundleSiz
    end

    % Equalization
    [pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

    % Common phase error (CPE) compensation
    if ~isempty(ptrsIndices)
        % Initialize temporary grid to store equalized symbols
        tempGrid = nrResourceGrid(carrier,pdsch.NumLayers);

        % Extract PT-RS symbols from received grid and estimated
        % channel grid
        [ptrsRx,ptrsHest,~,~,ptrsHestIndices,ptrsLayerIndices] = nrExtractResources(ptrsIndic

        if (simLocal.PerfectChannelEstimator)
            % Apply precoding to channel estimate
            ptrsHest = hPRGPrecode(size(estChannelGrid),carrier.NStartGrid,ptrsHest,ptrsHestI
        end

        % Equalize PT-RS symbols and map them to tempGrid
        ptrsEq = nrEqualizeMMSE(ptrsRx,ptrsHest,noiseEst);
        tempGrid(ptrsLayerIndices) = ptrsEq;

        % Estimate the residual channel at the PT-RS locations in
        % tempGrid
        cpe = nrChannelEstimate(tempGrid,ptrsIndices,ptrsSymbols);
```

```
                % Sum estimates across subcarriers, receive antennas, and
                % layers. Then, get the CPE by taking the angle of the
                % resultant sum
                cpe = angle(sum(cpe,[1 3 4]));

                % Map the equalized PDSCH symbols to tempGrid
                tempGrid(pdschIndices) = pdschEq;

                % Correct CPE in each OFDM symbol within the range of reference
                % PT-RS OFDM symbols
                symLoc = pdschIndicesInfo.PTRSSymbolSet(1)+1:pdschIndicesInfo.PTRSSymbolSet(end)+1;
                tempGrid(:,symLoc,:) = tempGrid(:,symLoc,:).*exp(-1i*cpe(symLoc));

                % Extract PDSCH symbols
                pdschEq = tempGrid(pdschIndices);
            end

            % Decode PDSCH physical channel
            [dlschLLRs,rxSymbols] = nrPDSCHDecode(carrier,pdsch,pdschEq,noiseEst);

            % Display EVM per layer, per slot and per RB
            if (simLocal.DisplayDiagnostics)
                plotLayerEVM(NSlots,nslot,pdsch,size(pdschGrid),pdschIndices,pdschSymbols,pdschEq);
            end

            % Scale LLRs by CSI
            csi = nrLayerDemap(csi); % CSI layer demapping
            for cwIdx = 1:pdsch.NumCodewords
                Qm = length(dlschLLRs{cwIdx})/length(rxSymbols{cwIdx}); % bits per symbol
                csi{cwIdx} = repmat(csi{cwIdx}.',Qm,1);                 % expand by each bit per symb
                dlschLLRs{cwIdx} = dlschLLRs{cwIdx} .* csi{cwIdx}(:);   % scale by CSI
            end

            % Decode the DL-SCH transport channel
            decodeDLSCHLocal.TransportBlockLength = trBlkSizes;
            [decbits,blkerr] = decodeDLSCHLocal(dlschLLRs,pdsch.Modulation,pdsch.NumLayers,harqEntity

            % Store values to calculate throughput
            simThroughput(snrIdx) = simThroughput(snrIdx) + sum(~blkerr .* trBlkSizes);
            maxThroughput(snrIdx) = maxThroughput(snrIdx) + sum(trBlkSizes);

            % Update current process with CRC error and advance to next process
            procstatus = updateAndAdvance(harqEntity,blkerr,trBlkSizes,pdschIndicesInfo.G);
            if (simLocal.DisplaySimulationInformation)
                fprintf('\n(%3.2f%%) NSlot=%d, %s',100*(nslot+1)/NSlots,nslot,procstatus);
            end

        end

        % Display the results dynamically in the command window
        if (simLocal.DisplaySimulationInformation)
            fprintf('\n');
        end
        fprintf('\nThroughput(Mbps) for %d frame(s) = %.4f\n',simLocal.NFrames,1e-6*simThroughput(sn
        fprintf('Throughput(%%) for %d frame(s) = %.4f\n',simLocal.NFrames,simThroughput(snrIdx)*100/

    end
```

```
Simulating transmission scheme 1 (8x2) and SCS=30kHz with CDL-C channel at -5dB SNR for 2 10ms fr

(2.50%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(5.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(7.50%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(10.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(12.50%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(15.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(17.50%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(22.50%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(25.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
(27.50%) NSlot=10, HARQ Proc 10: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=11, HARQ Proc 11: CW0: Initial transmission failed (RV=0,CR=0.474736).
(32.50%) NSlot=12, HARQ Proc 12: CW0: Initial transmission failed (RV=0,CR=0.474736).
(35.00%) NSlot=13, HARQ Proc 13: CW0: Initial transmission failed (RV=0,CR=0.474736).
(37.50%) NSlot=14, HARQ Proc 14: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=15, HARQ Proc 15: CW0: Initial transmission failed (RV=0,CR=0.474736).
(42.50%) NSlot=16, HARQ Proc 0: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(45.00%) NSlot=17, HARQ Proc 1: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(47.50%) NSlot=18, HARQ Proc 2: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(50.00%) NSlot=19, HARQ Proc 3: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(52.50%) NSlot=20, HARQ Proc 4: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(55.00%) NSlot=21, HARQ Proc 5: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(57.50%) NSlot=22, HARQ Proc 6: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(60.00%) NSlot=23, HARQ Proc 7: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(62.50%) NSlot=24, HARQ Proc 8: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(65.00%) NSlot=25, HARQ Proc 9: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(67.50%) NSlot=26, HARQ Proc 10: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(70.00%) NSlot=27, HARQ Proc 11: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(72.50%) NSlot=28, HARQ Proc 12: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(75.00%) NSlot=29, HARQ Proc 13: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(77.50%) NSlot=30, HARQ Proc 14: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(80.00%) NSlot=31, HARQ Proc 15: CW0: Retransmission #1 passed (RV=2,CR=0.474736).
(82.50%) NSlot=32, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(85.00%) NSlot=33, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(87.50%) NSlot=34, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=35, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(92.50%) NSlot=36, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(95.00%) NSlot=37, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(97.50%) NSlot=38, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=39, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).

Throughput(Mbps) for 2 frame(s) = 24.1728
Throughput(%) for 2 frame(s) = 40.0000

Simulating transmission scheme 1 (8x2) and SCS=30kHz with CDL-C channel at 0dB SNR for 2 10ms fra

(2.50%) NSlot=0, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(5.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(7.50%) NSlot=2, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(10.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(12.50%) NSlot=4, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
(15.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(17.50%) NSlot=6, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(20.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).
(22.50%) NSlot=8, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.474736).
```

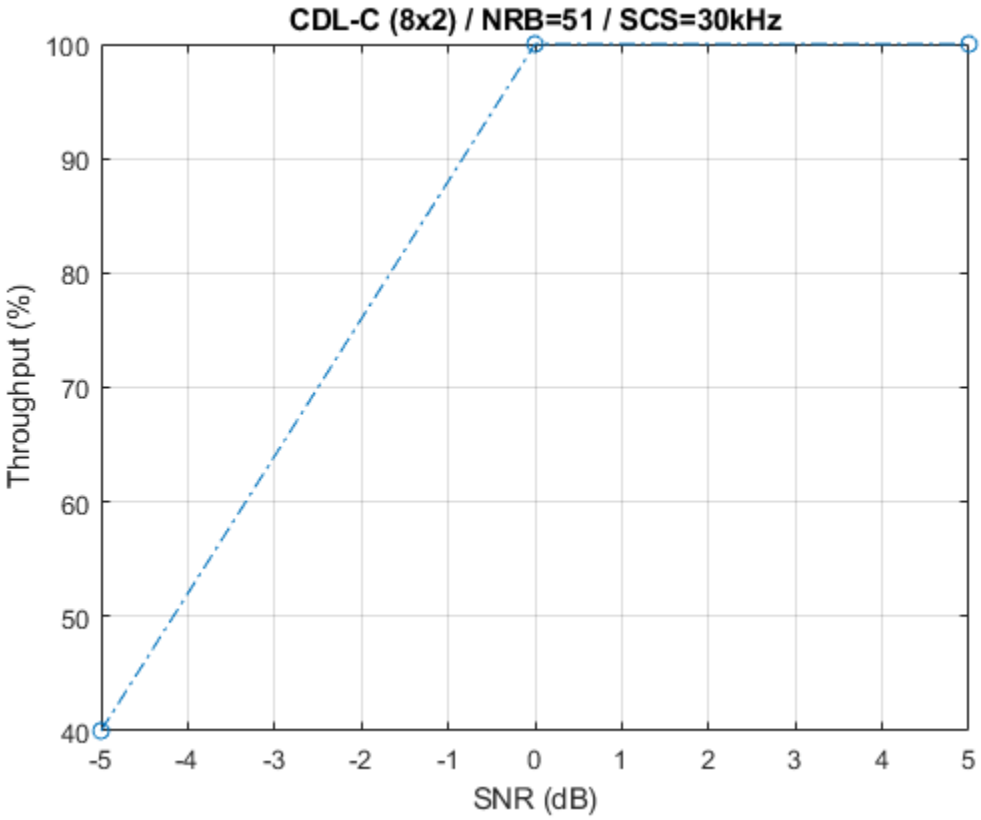
```
(25.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.474736).
(27.50%) NSlot=10, HARQ Proc 10: CW0: Initial transmission passed (RV=0,CR=0.474736).
(30.00%) NSlot=11, HARQ Proc 11: CW0: Initial transmission passed (RV=0,CR=0.474736).
(32.50%) NSlot=12, HARQ Proc 12: CW0: Initial transmission passed (RV=0,CR=0.474736).
(35.00%) NSlot=13, HARQ Proc 13: CW0: Initial transmission passed (RV=0,CR=0.474736).
(37.50%) NSlot=14, HARQ Proc 14: CW0: Initial transmission passed (RV=0,CR=0.474736).
(40.00%) NSlot=15, HARQ Proc 15: CW0: Initial transmission passed (RV=0,CR=0.474736).
(42.50%) NSlot=16, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(45.00%) NSlot=17, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(47.50%) NSlot=18, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(50.00%) NSlot=19, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(52.50%) NSlot=20, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
(55.00%) NSlot=21, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(57.50%) NSlot=22, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(60.00%) NSlot=23, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).
(62.50%) NSlot=24, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.474736).
(65.00%) NSlot=25, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.474736).
(67.50%) NSlot=26, HARQ Proc 10: CW0: Initial transmission passed (RV=0,CR=0.474736).
(70.00%) NSlot=27, HARQ Proc 11: CW0: Initial transmission passed (RV=0,CR=0.474736).
(72.50%) NSlot=28, HARQ Proc 12: CW0: Initial transmission passed (RV=0,CR=0.474736).
(75.00%) NSlot=29, HARQ Proc 13: CW0: Initial transmission passed (RV=0,CR=0.474736).
(77.50%) NSlot=30, HARQ Proc 14: CW0: Initial transmission passed (RV=0,CR=0.474736).
(80.00%) NSlot=31, HARQ Proc 15: CW0: Initial transmission passed (RV=0,CR=0.474736).
(82.50%) NSlot=32, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(85.00%) NSlot=33, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(87.50%) NSlot=34, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(90.00%) NSlot=35, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(92.50%) NSlot=36, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
(95.00%) NSlot=37, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(97.50%) NSlot=38, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(100.00%) NSlot=39, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).

Throughput(Mbps) for 2 frame(s) = 60.4320
Throughput(%) for 2 frame(s) = 100.0000

Simulating transmission scheme 1 (8x2) and SCS=30kHz with CDL-C channel at 5dB SNR for 2 10ms fra

(2.50%) NSlot=0, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(5.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(7.50%) NSlot=2, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(10.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(12.50%) NSlot=4, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
(15.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(17.50%) NSlot=6, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(20.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).
(22.50%) NSlot=8, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.474736).
(25.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.474736).
(27.50%) NSlot=10, HARQ Proc 10: CW0: Initial transmission passed (RV=0,CR=0.474736).
(30.00%) NSlot=11, HARQ Proc 11: CW0: Initial transmission passed (RV=0,CR=0.474736).
(32.50%) NSlot=12, HARQ Proc 12: CW0: Initial transmission passed (RV=0,CR=0.474736).
(35.00%) NSlot=13, HARQ Proc 13: CW0: Initial transmission passed (RV=0,CR=0.474736).
(37.50%) NSlot=14, HARQ Proc 14: CW0: Initial transmission passed (RV=0,CR=0.474736).
(40.00%) NSlot=15, HARQ Proc 15: CW0: Initial transmission passed (RV=0,CR=0.474736).
(42.50%) NSlot=16, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(45.00%) NSlot=17, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(47.50%) NSlot=18, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(50.00%) NSlot=19, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(52.50%) NSlot=20, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
```

```
(55.00%) NSlot=21, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(57.50%) NSlot=22, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(60.00%) NSlot=23, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).
(62.50%) NSlot=24, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.474736).
(65.00%) NSlot=25, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.474736).
(67.50%) NSlot=26, HARQ Proc 10: CW0: Initial transmission passed (RV=0,CR=0.474736).
(70.00%) NSlot=27, HARQ Proc 11: CW0: Initial transmission passed (RV=0,CR=0.474736).
(72.50%) NSlot=28, HARQ Proc 12: CW0: Initial transmission passed (RV=0,CR=0.474736).
(75.00%) NSlot=29, HARQ Proc 13: CW0: Initial transmission passed (RV=0,CR=0.474736).
(77.50%) NSlot=30, HARQ Proc 14: CW0: Initial transmission passed (RV=0,CR=0.474736).
(80.00%) NSlot=31, HARQ Proc 15: CW0: Initial transmission passed (RV=0,CR=0.474736).
(82.50%) NSlot=32, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(85.00%) NSlot=33, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(87.50%) NSlot=34, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(90.00%) NSlot=35, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(92.50%) NSlot=36, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
(95.00%) NSlot=37, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(97.50%) NSlot=38, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(100.00%) NSlot=39, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).

Throughput(Mbps) for 2 frame(s) = 60.4320
Throughput(%) for 2 frame(s) = 100.0000
```

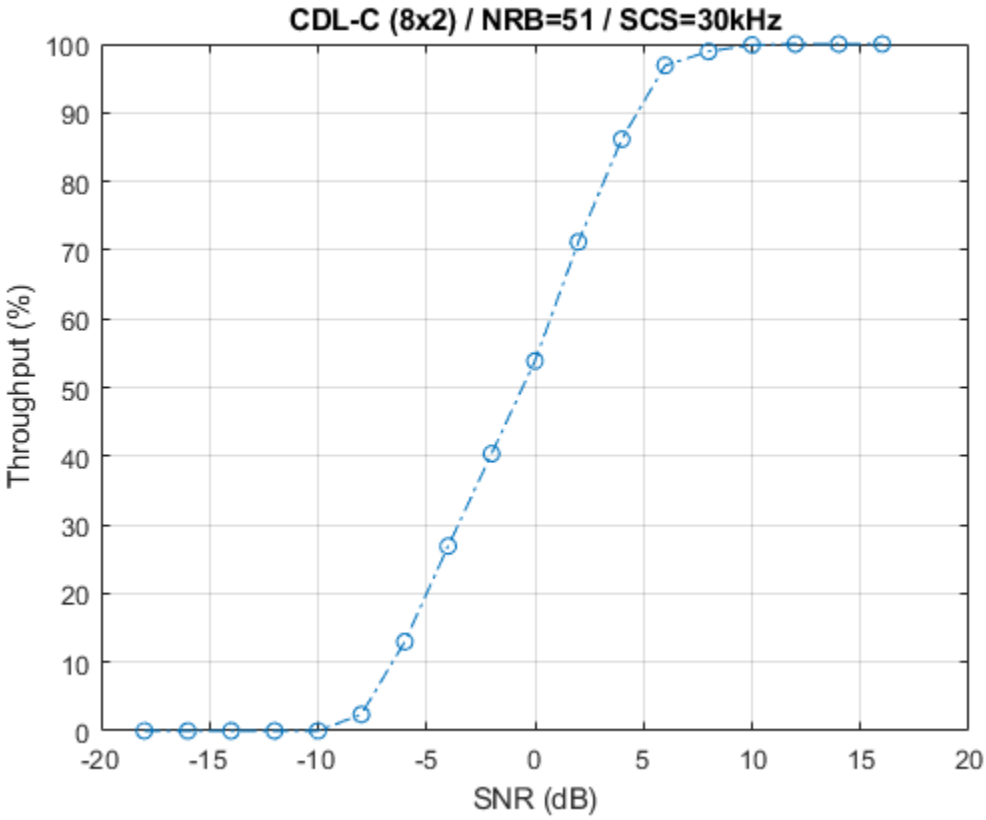
Results

Display the measured throughput. This is calculated as the percentage of the maximum possible throughput of the link given the available resources for data transmission.

```
figure;
plot(simParameters.SNRIn,simThroughput*100./maxThroughput,'o-.')
xlabel('SNR (dB)'); ylabel('Throughput (%)'); grid on;
title(sprintf('%s (%dx%d) / NRB=%d / SCS=%dkHz', ...
              simParameters.DelayProfile,simParameters.NTxAnts,simParameters.NRxAnts, ...
              simParameters.Carrier.NSizeGrid,simParameters.Carrier.SubcarrierSpacing));

% Bundle key parameters and results into a combined structure for recording
simResults.simParameters = simParameters;
simResults.simThroughput = simThroughput;
simResults.maxThroughput = maxThroughput;
```



The figure below shows throughput results obtained simulating 10000 subframes (`NFrames = 1000`, `SNRIn = -18:2:16`).



Selected Bibliography

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.
3. 3GPP TS 38.213. "NR; Physical layer procedures for control." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
4. 3GPP TS 38.214. "NR; Physical layer procedures for data." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
5. 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.

Local Functions

```
function validateNumLayers(simParameters)
% Validate the number of layers, relative to the antenna geometry

    numlayers = simParameters.PDSCH.NumLayers;
    ntxants = simParameters.NTxAnts;
    nrxants = simParameters.NRxAnts;
    antennaDescription = sprintf('min(NTxAnts,NRxAnts) = min(%d,%d) = %d',ntxants,nrxants,min(nt
    if numlayers > min(ntxants,nrxants)
        error('The number of layers (%d) must satisfy NumLayers <= %s', ...
            numlayers,antennaDescription);
```

```
        end

        % Display a warning if the maximum possible rank of the channel equals
        % the number of layers
        if (numlayers > 2) && (numlayers == min(ntxants,nrxants))
            warning(['The maximum possible rank of the channel, given by %s, is equal to NumLayers (%
                ' This may result in a decoding failure under some channel conditions.' ...
                ' Try decreasing the number of layers or increasing the channel rank' ...
                ' (use more transmit or receive antennas).'],antennaDescription,numlayers); %#ok<SPWH
        end

    end

    function estChannelGrid = getInitialChannelEstimate(carrier,nTxAnts,propchannel)
    % Obtain channel estimate before first transmission. This can be used to
    % obtain a precoding matrix for the first slot.

        ofdmInfo = nrOFDMInfo(carrier);

        chInfo = info(propchannel);
        maxChDelay = chInfo.MaximumChannelDelay;

        % Temporary waveform (only needed for the sizes)
        tmpWaveform = zeros((ofdmInfo.SampleRate/1000/carrier.SlotsPerSubframe)+maxChDelay,nTxAnts);

        % Filter through channel
        [~,pathGains,sampleTimes] = propchannel(tmpWaveform);

        % Perfect timing synch
        pathFilters = getPathFilters(propchannel);
        offset = nrPerfectTimingEstimate(pathGains,pathFilters);

        % Perfect channel estimate
        estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sampleTimes);

    end

    function wtx = getPrecodingMatrix(carrier,pdsch,hestGrid,prgbundlesize)
    % Calculate precoding matrices for all PRGs in the carrier that overlap
    % with the PDSCH allocation

        % Maximum CRB addressed by carrier grid
        maxCRB = carrier.NStartGrid + carrier.NSizeGrid - 1;

        % PRG size
        if nargin==4 && ~isempty(prgbundlesize)
            Pd_BWP = prgbundlesize;
        else
            Pd_BWP = maxCRB + 1;
        end

        % PRG numbers (1-based) for each RB in the carrier grid
        NPRG = ceil((maxCRB + 1) / Pd_BWP);
        prgset = repmat((1:NPRG),Pd_BWP,1);
        prgset = prgset(carrier.NStartGrid + (1:carrier.NSizeGrid).');

        [~,~,R,P] = size(hestGrid);
        wtx = zeros([pdsch.NumLayers P NPRG]);
```

```
    for i = 1:NPRG

        % Subcarrier indices within current PRG and within the PDSCH
        % allocation
        thisPRG = find(prgset==i) - 1;
        thisPRG = intersect(thisPRG,pdsch.PRBSet(:),'rows');
        prgSc = (1:12)' + 12*thisPRG';
        prgSc = prgSc(:);

        if (~isempty(prgSc))

            % Average channel estimate in PRG
            estAllocGrid = hestGrid(prgSc,:,:,:);
            Hest = permute(mean(reshape(estAllocGrid,[],R,P)),[2 3 1]);

            % SVD decomposition
            [~,~,V] = svd(Hest);
            wtx(:,:,i) = V(:,1:pdsch.NumLayers).';

        end

    end

    wtx = wtx / sqrt(pdsch.NumLayers); % Normalize by NumLayers

end

function estChannelGrid = precodeChannelEstimate(carrier,estChannelGrid,W)
% Apply precoding matrix W to the last dimension of the channel estimate

    [K,L,R,P] = size(estChannelGrid);
    estChannelGrid = reshape(estChannelGrid,[K*L R P]);
    estChannelGrid = hPRGPrecode([K L R P],carrier.NStartGrid,estChannelGrid,reshape(1:numel(estC
    estChannelGrid = reshape(estChannelGrid,K,L,R,[]);

end

function plotLayerEVM(NSlots,nslot,pdsch,siz,pdschIndices,pdschSymbols,pdschEq)
% Plot EVM information

    persistent slotEVM;
    persistent rbEVM
    persistent evmPerSlot;

    if (nslot==0)
        slotEVM = comm.EVM;
        rbEVM = comm.EVM;
        evmPerSlot = NaN(NSlots,pdsch.NumLayers);
        figure;
    end
    evmPerSlot(nslot+1,:) = slotEVM(pdschSymbols,pdschEq);
    subplot(2,1,1);
    plot(0:(NSlots-1),evmPerSlot,'o-');
    xlabel('Slot number');
    ylabel('EVM (%)');
    legend("layer " + (1:pdsch.NumLayers),'Location','EastOutside');
    title('EVM per layer per slot');
```

```
    subplot(2,1,2);
    [k,~,p] = ind2sub(siz,pdschIndices);
    rbsubs = floor((k-1) / 12);
    NRB = siz(1) / 12;
    evmPerRB = NaN(NRB,pdsch.NumLayers);
    for nu = 1:pdsch.NumLayers
        for rb = unique(rbsubs).'
            this = (rbsubs==rb & p==nu);
            evmPerRB(rb+1,nu) = rbEVM(pdschSymbols(this),pdschEq(this));
        end
    end
    plot(0:(NRB-1),evmPerRB,'x-');
    xlabel('Resource block');
    ylabel('EVM (%)');
    legend("layer " + (1:pdsch.NumLayers),'Location','EastOutside');
    title(['EVM per layer per resource block, slot #' num2str(nslot)]);

    drawnow;

end
```

See Also

Objects
nrTDLChannel | nrCDLChannel

Functions
parfor | nrPDSCH | nrPDSCHDecode | nrDLSCHInfo

More About

- “SNR Definition Used in Link Simulations” on page 5-86

Downlink Control Processing and Procedures

This example describes the blind search decoding of the physical downlink control channel (PDCCH) for 5G New Radio communications system. Building on the tutorial “Modeling Downlink Control Information”, this example introduces the concepts of control resource set (CORESET) and search spaces, their generic specification and shows how a PDCCH instance is mapped to one of several candidates within a search space. To recover the transmitted control information at the receiver, the example performs a blind search over the set of candidates.

System Parameters

Set system parameters corresponding to the carrier, CORESET, search space set, and PDCCH instance respectively.

```
rng(111);                                % Set RNG state for repeatability

% Carrier configuration
carrier = nrCarrierConfig;
carrier.NCellID = 2;                     % Cell identity
carrier.SubcarrierSpacing = 30;          % Carrier/BWP Subcarrier spacing
carrier.CyclicPrefix = 'normal';         % Cyclic prefix
carrier.NSlot      = 0;                  % Slot counter
carrier.NFrame     = 0;                  % Frame counter
carrier.NStartGrid = 10;                 % Carrier offset
carrier.NSizeGrid  = 48;                 % Size of carrier in RB

% CORESET configuration
coreset = nrCORESETConfig;
coreset.CORESETID = 1;                   % CORESET ID (0...11)
coreset.FrequencyResources = ones(1,4); % 6 RB sized
coreset.Duration = 1;                    % CORESET symbol duration (1,2,3)
coreset.CCEREGMapping = 'interleaved';   % CORESET Mapping
coreset.REGBundleSize = 2;               % L (2,6) or (3,6)
coreset.InterleaverSize = 2;             % R (2,3,6)
coreset.ShiftIndex = carrier.NCellID;    % default to NCellID

% Search space configuration
ss = nrSearchSpaceConfig;
ss.CORESETID = 1;                        % Associated CORESET ID (0...11)
ss.SearchSpaceType = 'ue';               % 'ue', 'common'
ss.StartSymbolWithinSlot = 0;            % Starting symbol in slot
ss.SlotPeriodAndOffset = [1 0];          % Search space period and offset
ss.Duration = 1;                         % Search space duration in slots
ss.NumCandidates = [4 2 1 0 0];          % For (1,2,4,8,16) levels respectively

% PDCCH configuration
pdcch = nrPDCCHConfig;
pdcch.NStartBWP  = 10;                   % BWP offset wrt CRB 0
pdcch.NSizeBWP   = 48;                   % Size of BWP in resource blocks
pdcch.CORESET = coreset;                 % Associated CORESET
pdcch.SearchSpace = ss;                  % Associated SearchSpace
pdcch.RNTI = 1;                          % C-RNTI
pdcch.DMRSScramblingID = [];             % Use carrier.NCellID instead
pdcch.AggregationLevel = 4;              % Number of CCEs in PDCCH (1,2,4,8,16)
pdcch.AllocatedCandidate = 1;            % 1-based scalar
```

This example assumes single slot processing, using a single bandwidth part with a single PDCCH transmission for an associated CORESET and search space set.

For more information on waveform generation with multiple physical channels, see the “5G NR Downlink Vector Waveform Generation” on page 1-2 example.

PDCCH Bit Capacity

The bit capacity for a PDCCH instance is determined based on the number of control-channel elements (CCE) configured for the PDCCH. A CCE consists of six resource-element groups (REGs) where a REG equals one resource block (RB) during one OFDM symbol.

```
% Number of bits for PDCCH resources and actual indices
[ind,dmrs,dmrsInd] = nrPDCCHResources(carrier,pdcch);
E = 2*numel(ind);
```

DCI Encoding

The `nrDCIEncode` function encodes the DCI message bits based on a downlink format. DCI encoding includes the stages of CRC attachment, polar encoding and rate matching the codeword to the PDCCH bit capacity `E`.

```
K = 64;                             % Number of DCI message bits
dciBits = randi([0 1],K,1,'int8');

dciCW = nrDCIEncode(dciBits,pdcch.RNTI,E);
```

PDCCH Symbol Generation and Mapping

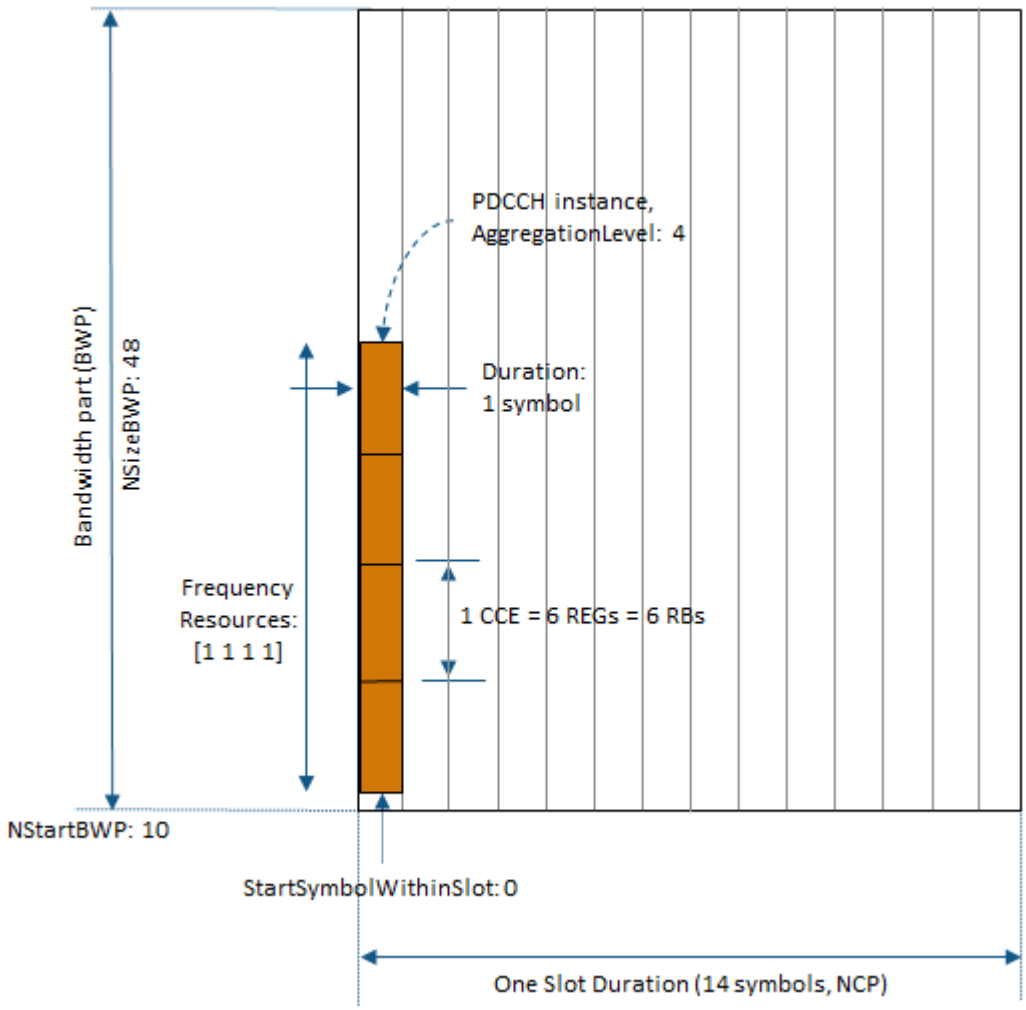
The `nrPDCCH` function maps the encoded DCI bits onto the physical downlink control channel (PDCCH). The function returns the scrambled, QPSK-modulated symbols. The scrambling accounts for the user-specific parameters.

```
if isempty(pdcch.DMRSScramblingID)
    nID = carrier.NCellID;
else
    nID = pdcch.DMRSScramblingID;
end
sym = nrPDCCH(dciCW,nID,pdcch.RNTI);
```

The PDCCH symbols are then mapped to the resource elements corresponding to the allocated candidate within an OFDM grid. The resource grid also contains PDSCH and PBCH symbols, and other reference signal elements. For simplicity, this example only additionally maps the PDCCH DM-RS symbols to the grid.

```
carrierGrid = nrResourceGrid(carrier);
carrierGrid(ind) = sym;                 % PDCCH symbols
carrierGrid(dmrsInd) = dmrs;            % PDCCH DM-RS
```

For a resource grid spanning the whole bandwidth part and a single slot, this figure shows some of the CORESET, search space set and PDCCH instance parameters for the selected example configuration.



OFDM Modulation

OFDM modulate the carrier grid. Specify no windowing for the slot-based processing.

```
[wave,winfo] = nrOFDMModulate(carrier,carrierGrid,'Windowing',0);
```

Fading Channel

Transmit the generated waveform over a TDL fading channel with delay profile A and a delay spread of 30 ns.

```
channel = nrTDLChannel;
channel.DelayProfile = 'TDL-A';
channel.DelaySpread = 30e-9;
channel.NumTransmitAntennas = 1;
channel.NumReceiveAntennas = 1;
channel.SampleRate = winfo.SampleRate;

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

```
txWave = [wave; zeros(maxChDelay, size(wave,2))];

rxWave = channel(txWave);
```

Noise Addition

Add white Gaussian noise with the specified level to the received signal, taking into account the coding rate, QPSK modulation, and sampling rate.

```
EbNo = 6;                               % in dB
bps = 2;                                % bits per symbol, 2 for QPSK
EsNo = EbNo + 10*log10(bps);
snrdB = EsNo + 10*log10(K/E);
noiseVar = 10.^(-snrdB/10);             % assumes unit signal power
N0 = sqrt(noiseVar)/sqrt(2*winfo.Nfft);
noise = N0 * complex(randn(size(rxWave)),randn(size(rxWave)));
rxWaveN = rxWave + noise;
```

Blind PDCCH and DCI Decoding

The UE does not have information about the detailed control channel structure. Therefore, the UE decodes the received PDCCH symbols blindly by monitoring a set of PDCCH candidates for each slot using the UE's RNTI to identify the right candidate (or instance).

Monitoring a candidate implies attempting to decode a set of resource elements corresponding to the candidate by checking if the returned checksum is zero for the known RNTI (UE). Use the `nrPDCCHSpace` function to determine all candidates specified by the search space set in terms of the PDCCH resource element indices, corresponding DM-RS symbols and indices.

For each candidate, the front-end recovery includes

- timing offset estimation based on the DM-RS symbols using the function `nrTimingEstimate`,
- OFDM demodulation using the function `nrOFDMDemodulate`,
- channel estimation based on the DM-RS symbols using the function `nrChannelEstimate`, and
- MMSE equalization using the function `nrEqualizeMMSE`

to yield the equalized candidate PDCCH symbols.

The equalized symbols per candidate are demodulated with known user-specific parameters and channel noise variance using the `nrPDCCHDecode` function.

For an instance of the received PDCCH codeword, the `nrDCIDecode` function includes the stages of rate recovery, polar decoding, and CRC decoding. If the output mask value is zero, the PDCCH is decoded successfully and the UE can process the DCI message.

In this example, the receiver assumes knowledge of the DCI format and the DCI payload size `K`. In practice, even these would be searched for in an outer loop over all supported formats with respective bit lengths per format.

```
listLen = 8;                            % polar decoding list length

% Get all possible candidates
[allInd,allDMRS,allDMRSInd] = nrPDCCHSpace(carrier,pdcch);

% Loop over all supported aggregation levels
decoded = false;
```

```
for alIdx = 1:5

    % Loop over all candidates at each aggregation level
    for cIdx = 1:pdcch.SearchSpace.NumCandidates(alIdx)

        % Get candidate
        cSymIdx = allInd{alIdx}(:,cIdx);
        cDMRS = allDMRS{alIdx}(:,cIdx);
        cDMRSInd = allDMRSInd{alIdx}(:,cIdx);

        % Timing estimate
        offset = nrTimingEstimate(carrier,rxWaveN,cDMRSInd,cDMRS);
        if offset > maxChDelay
            offset = 0;
        end
        rxWaveS = rxWaveN(1+offset:end,:);

        % OFDM demodulate the carrier
        rxCarrGrid = nrOFDMDemodulate(carrier,rxWaveS);

        % Channel estimate
        [hest,nVar] = nrChannelEstimate(carrier,rxCarrGrid,cDMRSInd,cDMRS);
        [rxSym,pdcchHest] = nrExtractResources(cSymIdx,rxCarrGrid,hest);

        % Equalization
        [pdcchEq,csi] = nrEqualizeMMSE(rxSym,pdcchHest,nVar);

        % Demodulate
        rxCW = nrPDCCHDecode(pdcchEq,nID,pdcch.RNTI,nVar);

        % Apply CSI
        csiRep = repmat(csi.',2,1);
        scalRxCW = rxCW.*csiRep(:);

        % Decode
        [decDCIBits,errFlag] = nrDCIDecode(scalRxCW,K,listLen,pdcch.RNTI);

        if isequal(errFlag,0)
            disp(['Decoded candidate #' num2str(cIdx)  ...
                ' at aggregation level ' num2str(2^(alIdx-1)) ...
                ' in slot'])
            decoded = true;
            if isequal(decDCIBits,dciBits)
                disp('   Recovered DCI bits with no errors');
            else
                disp('   Recovered DCI bits with errors');
            end
            break;
        end
    end
    % Dont loop over other aggregation levels if RNTI matched
    if decoded
        break;
    end
end

Decoded candidate #1 at aggregation level 4 in slot
   Recovered DCI bits with no errors
```

For the chosen system parameters, the decoded information matches the transmitted information bits.

The example searched over all candidates within a single search space set as specified by the `ss` configuration parameter. Search over multiple search space sets would require another external loop over all the sets defined.

Selected 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.
3. 3GPP TS 38.213. "NR; Physical layer procedures for control" 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

See Also

Functions

`nrDCIDecode` | `nrDCIEncode` | `nrPDCCHDecode` | `nrPDCCH` | `nrPDCCHSpace` | `nrPDCCHResources`

NR Channel Estimation Using CSI-RS

This example shows how to generate channel state information reference signal (CSI-RS) symbols and indices for a given carrier and CSI-RS resource configuration, as defined in TS 38.211 Section 7.4.1.5. The example shows how to map the generated symbols to the carrier resource grid, performs channel estimation at the receiver side, and compares the estimated channel against the actual channel.

Introduction

CSI-RS is a downlink-specific (DL) reference signal. The NR standard defines zero-power (ZP) and non-zero-power (NZP) CSI-RSs.

The user equipment (UE) processes utilize NZP-CSI-RSs:

- L1-Reference signal received power (RSRP) measurements for mobility and beam management
- DL CSI acquisition
- Interference measurement
- Time and frequency tracking

ZP-CSI-RS is used for DL CSI acquisition and interference measurement. It also masks certain resource elements (REs) to make them unavailable for PDSCH transmission. As the name ZP indicates, nothing is transmitted in those REs.

This example shows how to use CSI-RS to perform channel estimation which forms the basis of CSI acquisition.

Initialize Configuration Objects

Create a carrier configuration object representing a 5 MHz carrier with subcarrier spacing of 15 kHz.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 25;
carrier.SubcarrierSpacing = 15;
carrier.NSlot = 1;
carrier.NFrame = 0
```

```
carrier =
  nrCarrierConfig with properties:

             NCellID: 1
    SubcarrierSpacing: 15
         CyclicPrefix: 'normal'
            NSizeGrid: 25
          NStartGrid: 0
               NSlot: 1
              NFrame: 0

   Read-only properties:
       SymbolsPerSlot: 14
     SlotsPerSubframe: 1
        SlotsPerFrame: 10
```

Create a CSI-RS configuration object representing two CSI-RS resources, NZP with row number 3 and ZP with row number 5.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','zp'};
csirs.CSIRSPeriod = {[5 1],[5 1]};
csirs.Density = {'one','one'};
csirs.RowNumber = [3 5];
csirs.SymbolLocations = {1,6};
csirs.SubcarrierLocations = {6,4};
csirs.NumRB = 25
```

```
csirs =
  nrCSIRSConfig with properties:

             CSIRSType: {'nzp'  'zp'}
            CSIRSPeriod: {[5 1]  [5 1]}
              RowNumber: [3 5]
                Density: {'one'  'one'}
        SymbolLocations: {[1]  [6]}
    SubcarrierLocations: {[6]  [4]}
                  NumRB: 25
               RBOffset: 0
                    NID: 0

   Read-only properties:
         NumCSIRSPorts: [2 4]
               CDMType: {'FD-CDM2'  'FD-CDM2'}
```

Consider the power scaling of CSI-RS in dB.

```
powerCSIRS = 0;
disp(['CSI-RS power scaling: ' num2str(powerCSIRS) ' dB']);
```

```
CSI-RS power scaling: 0 dB
```

Generate CSI-RS Symbols and Indices

Generate CSI-RS symbols for the specified carrier and CSI-RS configuration parameters. Apply power scaling.

```
sym = nrCSIRS(carrier,csirs);
csirsSym = sym*db2mag(powerCSIRS);
```

The variable `csirsSym` is a column vector containing CSI-RS symbols.

Generate CSI-RS indices for the specified carrier and CSI-RS configuration parameters.

```
csirsInd = nrCSIRSIndices(carrier,csirs);
```

The variable `csirsInd` is also a column vector of same size as that of `csirsSym`.

When you configure both ZP and NZP resources, the generation of ZP signals takes priority over the generation of NZP signals.

Initialize Carrier Grid

Initialize the carrier resource grid for one slot.

```
ports = max(csirs.NumCSIRSPorts);   % Number of antenna ports
txGrid = nrResourceGrid(carrier,ports);
```

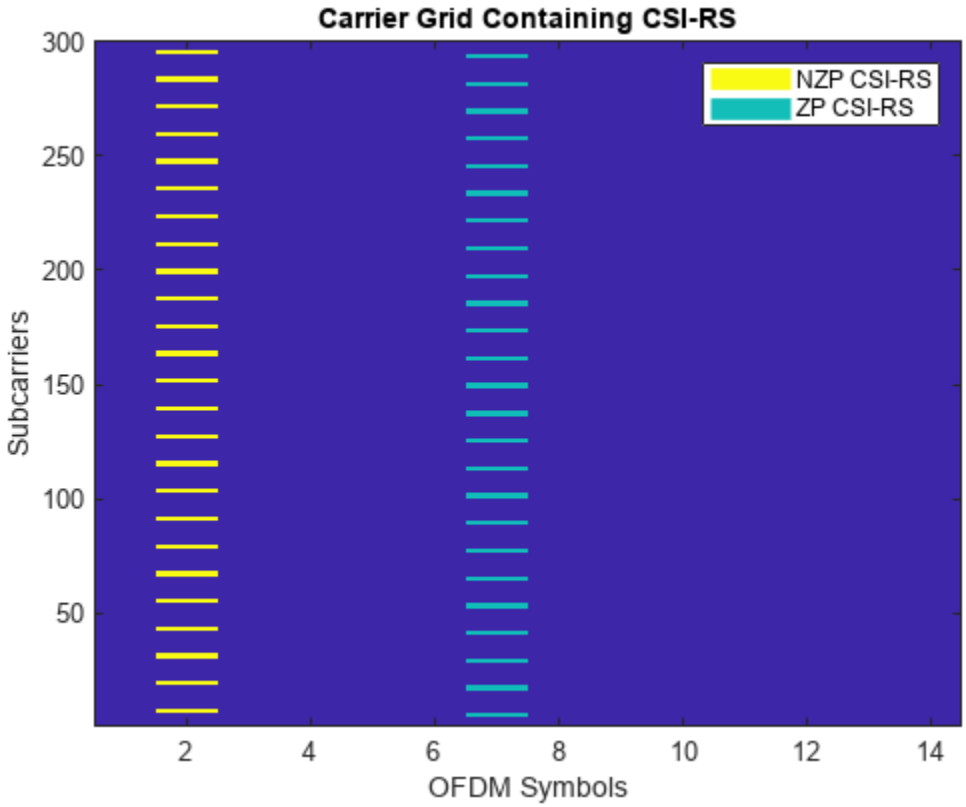
Map CSI-RS Symbols Onto Carrier Grid

Perform resource element mapping.

```
txGrid(csirsInd) = csirsSym;
```

Plot the locations of the CSI-RS (both ZP and NZP) in the grid.

```
plotGrid(size(txGrid),csirsInd,csirsSym);
```



Perform OFDM Modulation

Perform OFDM modulation to generate the time-domain waveform.

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

Pass Time-Domain Waveform Through Channel and Add AWGN Noise

Configure the number of receiving antennas.

```
R = 4;
```

Configure the channel.

```
channel = nrTDLChannel;
channel.NumTransmitAntennas = ports;
```

```
channel.NumReceiveAntennas = R;
channel.DelayProfile = 'TDL-C';
channel.MaximumDopplerShift = 10;
channel.DelaySpread = 1e-8

channel =
  nrTDLChannel with properties:

                DelayProfile: 'TDL-C'
                 DelaySpread: 1.0000e-08
          MaximumDopplerShift: 10
                   SampleRate: 30720000
              MIMOCorrelation: 'Low'
                 Polarization: 'Co-Polar'
        TransmissionDirection: 'Downlink'
          NumTransmitAntennas: 4
           NumReceiveAntennas: 4
           NormalizePathGains: true
                  InitialTime: 0
                 NumSinusoids: 48
                 RandomStream: 'mt19937ar with seed'
                         Seed: 73
      NormalizeChannelOutputs: true
             ChannelFiltering: true
    TransmitAndReceiveSwapped: false
```

Based on the configured channel, append zeros to the transmitted waveform to account for the channel delay.

```
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))];
```

Pass waveform through channel.

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

To produce the actual propagation channel `H_actual`, perform perfect channel estimation.

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

Add AWGN noise to the waveform. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
SNRdB = 50;           % in dB
SNR = 10^(SNRdB/10);  % Linear value
N0 = 1/sqrt(2.0*R*double(ofdmInfo.Nfft)*SNR); % Noise variance
rng(0);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;
```

Perform timing synchronization using NZP-CSI-RS. To estimate timing offset, use `nrTimingEstimate` and consider the NZP-CSI-RS as a reference.

```
% Disable ZP-CSI-RS resource, not going to be used for timing and channel
% estimation
```

```
csirs.CSIRSPeriod = {[5 1],'off'};
% Generate reference symbols and apply power scaling
refSym = db2mag(powerCSIRS)*nrCSIRS(carrier,csirs);
% Generate reference indices
refInd = nrCSIRSIndices(carrier,csirs);
offset = nrTimingEstimate(carrier,rxWaveform,refInd,refSym)
```

```
offset = 7
```

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

OFDM demodulate the received time-domain waveform.

```
rxGrid = nrOFDMDemodulate(carrier,rxWaveform); % Of size K-by-L-by-R
```

Compare Estimated Channel Against Actual Channel

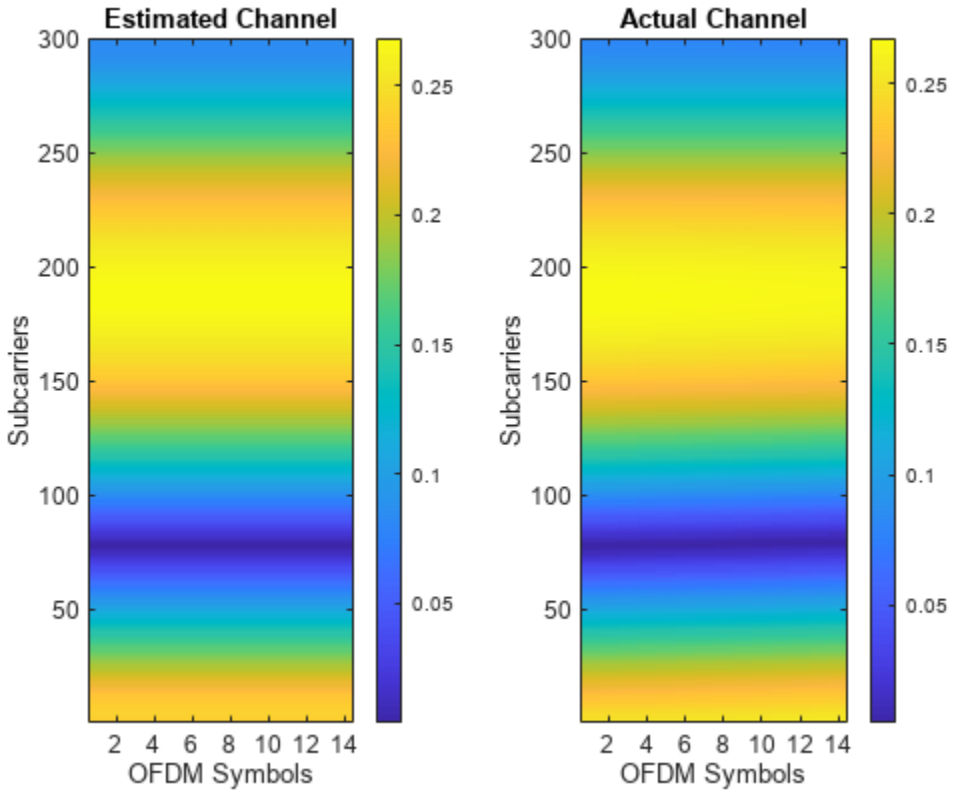
Perform practical channel estimation using NZP-CSI-RS. Make sure the CSI-RS symbols `csirsSym` are of the same CDM type.

```
cdmLen = [2 1]; % Corresponds to CDMType = 'FD-CDM2'
[H_est,nVar] = nrChannelEstimate(carrier,rxGrid,refInd,refSym,'CDMLengths',cdmLen);
estSNR = -10*log10(nVar);
disp(['estimated SNR = ' num2str(estSNR) ' dB'])
```

```
estimated SNR = 27.4577 dB
```

Plot the estimated channel and actual channel between first transmitting antenna and first receiving antenna.

```
figure;

% Plot the estimated channel
subplot(1,2,1)
imagesc(abs(H_est(:,:,1,1)));
colorbar;
title('Estimated Channel')
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');

% Plot the actual channel
subplot(1,2,2)
imagesc(abs(H_actual(:,:,1,1)));
colorbar;
title('Actual Channel')
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
```



Calculate the channel estimation error.

```
H_err = (H_est - H_actual(:,:,:,1:size(H_est,4)));
[minErr,maxErr] = bounds(abs(H_err),'all');
disp(['Absolute value of the channel estimation error is in the range of [' num2str(minErr) ', '
```

```
Absolute value of the channel estimation error is in the range of [4.9184e-06, 0.035047]
```

Local Functions

```
function plotGrid(gridSize,csirsInd,csirsSym)
%    plotGrid(GRIDSIZE,CSIRSIND,CSIRSSYM) plots the carrier grid of size GRIDSIZE
%    by populating the grid with CSI-RS symbols using CSIRSIND and CSIRSSYM.

    figure()
    cmap = colormap(gcf);
    chpval = {20,2};
    chpscale = 0.25*length(cmap); % Scaling factor

    tempSym = csirsSym;
    tempSym(tempSym ~= 0) = chpval{1}; % Replacing non-zero-power symbols
    tempSym(tempSym == 0) = chpval{2}; % Replacing zero-power symbols
    tempGrid = complex(zeros(gridSize));
    tempGrid(csirsInd) = tempSym;

    image(chpscale*tempGrid(:,:,1)); % Multiplied with scaling factor for better visualization
    axis xy;
    names = {'NZP CSI-RS','ZP CSI-RS'};
```

```
    clevels = chpscale*[chpval{:}];
    N = length(clevels);
    L = line(ones(N),ones(N),'LineWidth',8); % Generate lines
    % Index the color map and associate the selected colors with the lines
    set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3)); % Set the colo
    % Create legend
    legend(names{:});

    title('Carrier Grid Containing CSI-RS')
    xlabel('OFDM Symbols');
    ylabel('Subcarriers');
end
```

References

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

See Also

Functions
`nrCSIRS` | `nrCSIRSIndices`

Objects
`nrCSIRSConfig` | `nrCarrierConfig`

More About

- “SNR Definition Used in Link Simulations” on page 5-86

5G NR CSI-RS Measurements

This example shows the measurement procedure of CSI-RSRP, CSI-RSSI, and CSI-RSRQ for the test environment, as described in TS 38.133 Annex A.4.6.4.3, by using the channel state information reference signal from 5G Toolbox™.

Introduction

In NR 5G, the three types of CSI-RS based reference signal measurements, as defined in TS 38.215 Sections 5.1.2 and 5.1.4, includes:

- **CSI-RSRP (CSI reference signal received power):** CSI-RSRP is defined as the linear average over the power contributions of the resource elements of the antenna ports, which carry CSI-RS configured for RSRP measurements. This measurement is performed across N number of resource blocks (measurement bandwidth). For this measurement, CSI-RS transmitted on antenna port(s) 3000 or 3000 and 3001 is used.
- **CSI-RSSI (CSI received signal strength indicator):** CSI-RSSI is defined as the linear average of the total received power observed only in the OFDM symbols, in which CSI-RS is present. This measurement is also performed across *N* number of resource blocks (measurement bandwidth). CSI-RSSI includes the power from sources, such as co-channel serving and non-serving cells, adjacent channel interference, and thermal noise. For this measurement, CSI-RS transmitted on antenna port 3000 is used.
- **CSI-RSRQ (CSI reference signal received quality):** CSI-RSRQ is defined as, $N * \frac{\text{CSI\_RSRP}}{\text{CSI\_RSSI}}$.

The purposes of these measurements include:

- Cell selection and reselection
- Mobility and handover management
- Beam management (beam adjustment and beam recovery)

This example configures only the CSI-RS from the test environment.

Initialize Configuration Objects

Carrier configuration

Create carrier configuration object occupying 10 MHz bandwidth with 15 kHz subcarrier spacing as per configuration 1 in TS 38.133 Table A.4.6.4.3.1-1.

```
carrier = nrCarrierConfig;
carrier.NSlot = 1;
carrier.NSizeGrid = 52;
```

CSI-RS configuration

As per the test environment TS 38.133 Table A.4.6.4.3.2-1, user equipment (UE) is configured with one CSI-RS resource set (CSI-RS 1.2 FDD), consisting of 2 CSI-RS resources.

```
csirs = nrCSIRSConfig;
% CSI-RS resource             #0       #1
csirs.CSIRSType           = {'nzp',   'nzp'};
csirs.CSIRSPeriod         = {[10 1],  [10 1]};
```

```
csirs.RowNumber           = [1        1]; % Single port (3000) CSI-RS resources
csirs.Density             = {'three', 'three'};
csirs.SymbolLocations     = {6,       10};
csirs.SubcarrierLocations = {0,       0};
csirs.NumRB               = [52,      52]; % Measurement bandwidth in terms of number of resource
```

Generate CSI-RS Symbols and Indices

Generate CSI-RS symbols and indices for the specified carrier and CSI-RS configuration parameters with the output resource format as 'cell'. This output resource format provides a way to identify the outputs uniquely for each CSI-RS resource in a resource set. You can also apply different power levels to each CSI-RS resource.

```
ind = nrCSIRSIndices(carrier,csirs,'OutputResourceFormat','cell');
sym = nrCSIRS(carrier,csirs,'OutputResourceFormat','cell');
```

Signal and Noise Power Setup

Set up the signal and noise powers as described in TS 38.133 Table A.4.6.4.3.2-2. As per note 2 in TS 38.133 Table A.4.6.4.3.2-2, interference from other cells and noise from other sources is modeled as additive white Gaussian noise (AWGN) of appropriate power `Noc`.

```
SINRdB0 = 0; % For CSI-RS #0
SINRdB1 = 3; % For CSI-RS #1
NocdBm = -94.65;
NocdB = NocdBm - 30;
Noc = 10^(NocdB/10);
```

Calculate the power scaling of CSI-RS resources by using SINR values.

```
% Power scaling of CSI-RS resource #0
SINR0 = 10^(SINRdB0/10);     % linear Es/Noc
Es0 = SINR0*Noc;

% Power scaling of CSI-RS resource #1
SINR1 = 10^(SINRdB1/10);     % linear Es/Noc
Es1 = SINR1*Noc;
```

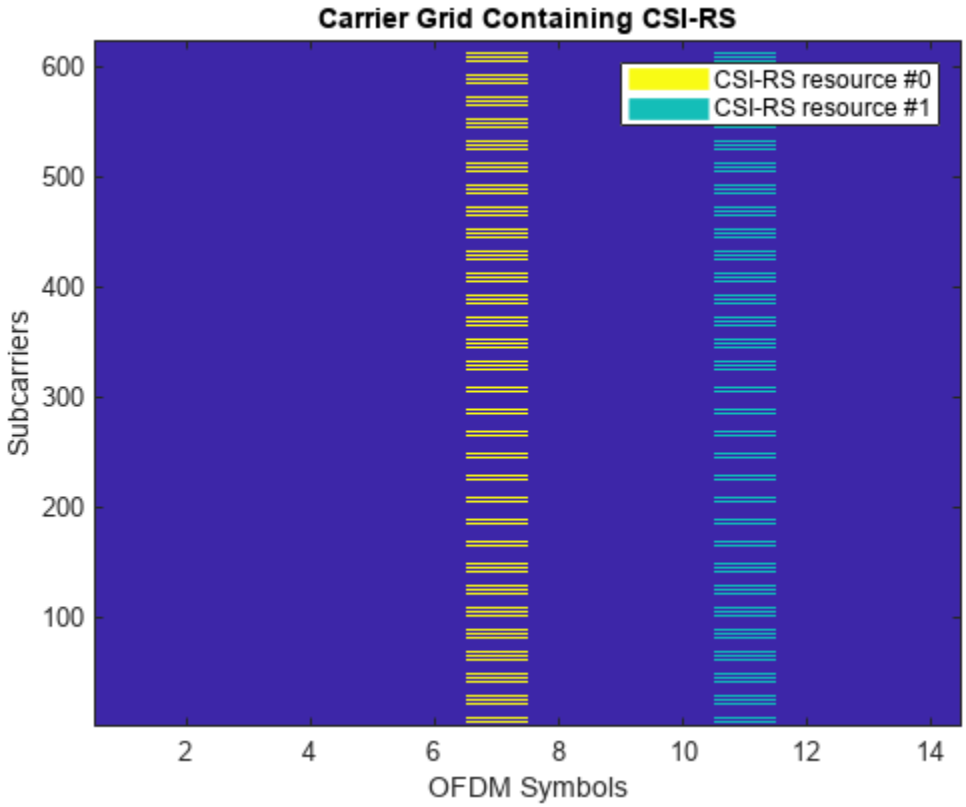
Initialize the Carrier Resource Grid and Map CSI-RS Symbols to the Grid

Initialize the carrier resource grid for one slot.

```
ports = max(csirs.NumCSIRSPorts); % Number of antenna ports
txGrid = nrResourceGrid(carrier,ports);
```

Apply the power scaling values to the CSI-RS resources and map them onto the grid.

```
txGrid(ind{1}) = sqrt(Es0)*sym{1};
txGrid(ind{2}) = sqrt(Es1)*sym{2};

% Plot the carrier grid for two CSI-RS resources
plotGrid(size(txGrid),ind)
```



Perform OFDM modulation

Perform OFDM modulation to generate the time-domain waveform.

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

Add AWGN to the Transmitted Waveform and Perform OFDM Demodulation

Consider the propagation condition as AWGN, as specified in TS 38.133 Table A.4.6.4.3.2-1.

```
% Generate the noise
rng('default');  % Set RNG state for repeatability
N0 = sqrt(Noc/(2*double(ofdmInfo.Nfft)));
noise = N0*complex(randn(size(txWaveform)),randn(size(txWaveform)));

% Add AWGN to the transmitted waveform
rxWaveform = txWaveform + noise;
```

Perform OFDM demodulation on the received time-domain waveform to get the received resource element array.

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

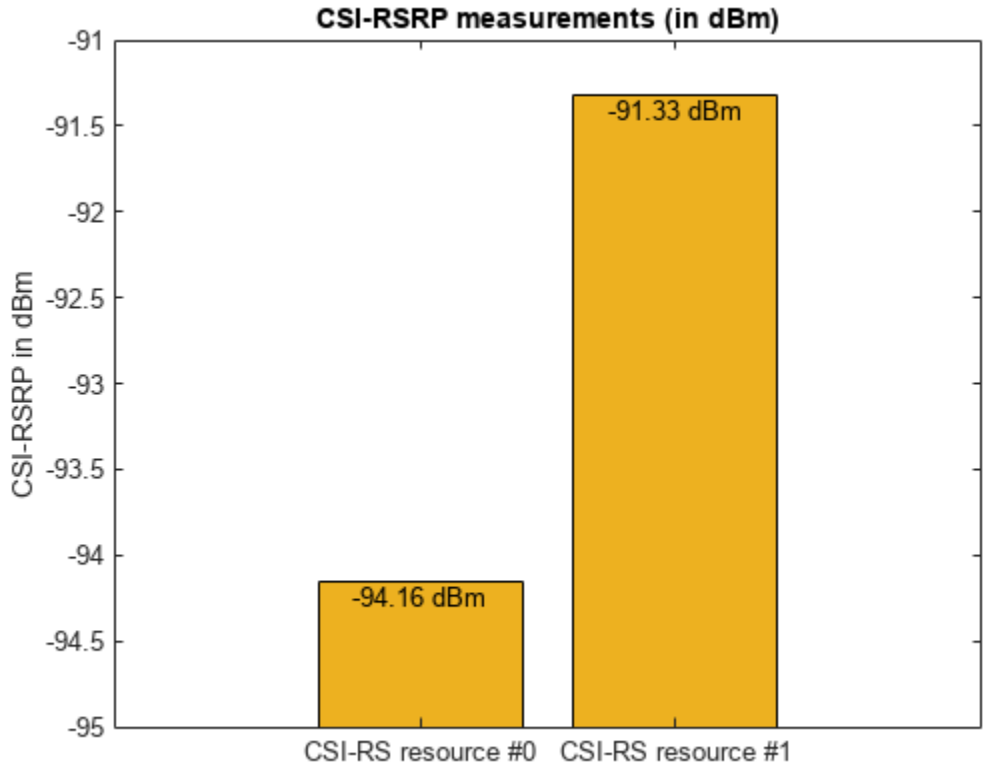
Perform CSI-RSRP, CSI-RSSI, and CSI-RSRQ Measurements

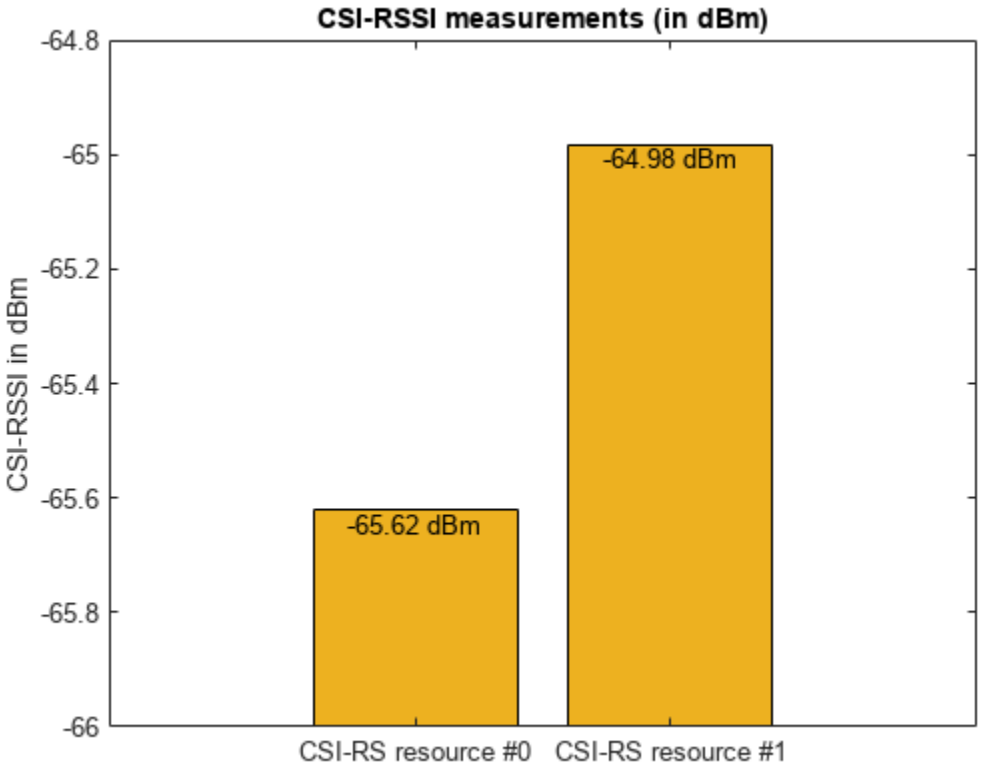
Finally, perform the CSI-RSRP, CSI-RSSI, and CSI-RSRQ measurements on the CSI-RS resources present in the received grid, by using the `nrCSIRSMeasurements` function.

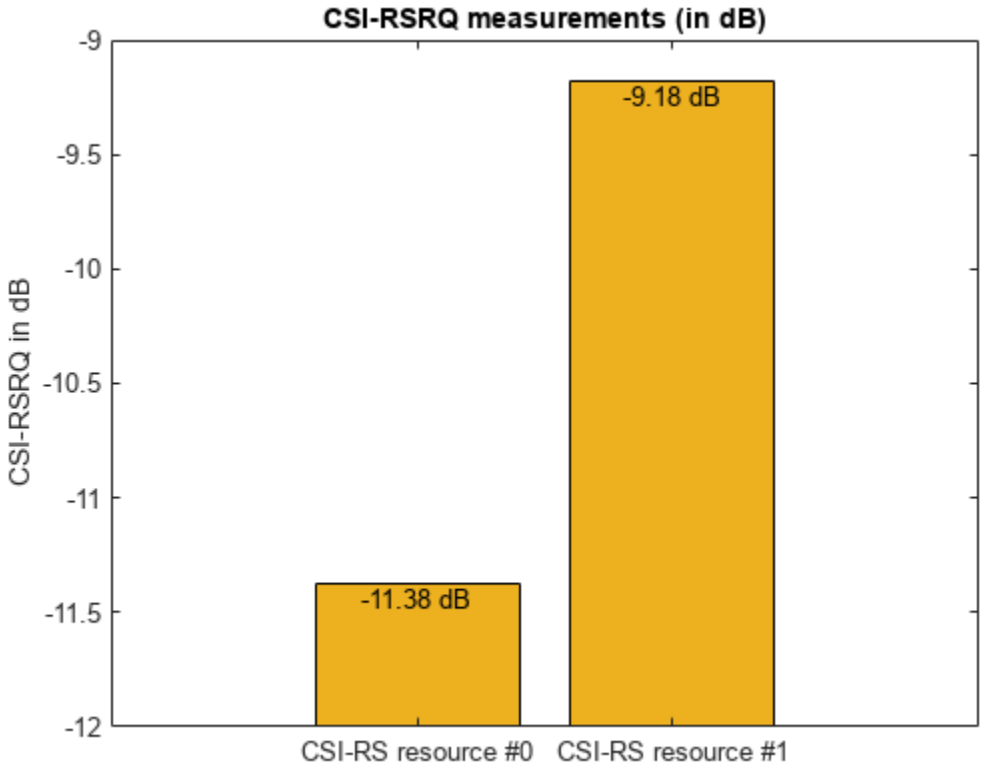
```
meas = nrCSIRSMeasurements(carrier,csirs,rxGrid)
```

```
meas = struct with fields:
    RSRPPerAntenna: [-94.1599 -91.3258]
    RSSIPerAntenna: [-65.6220 -64.9823]
    RSRQPerAntenna: [-11.3779 -9.1834]
```

```
% Plot RSRPdBm, RSSIdBm and RSRQdB measurements for all CSI-RS resources
plotCSIRSMeasurements(meas)
```




CSI-RSSI measurements (in dBm)
-64.8
-65
-65.2
-65.4
-65.6
-65.8
-66
CSI-RSSI in dBm
-64.98 dBm
-65.62 dBm
CSI-RS resource #0
CSI-RS resource #1



You can compare the measured CSI-RSRP values of two CSI-RS resources represented by the output field `RSRPdBm` to the standard specified values given in TS 38.133 Table A.4.6.4.3.2-2.

Local Functions

```
function plotGrid(gridSize,csirsInd)
%    plotGrid(GRIDSIZE,CSIRSIND) plots the carrier grid of size GRIDSIZE
%    by populating the grid with CSI-RS symbols of multiple resources
%    indicated by a cell array of CSI-RS indices CSIRSIND.

    figure()
    cmap = colormap(gcf);

    % Considering the following values for two CSI-RS resources and they need
    % to be updated based on the number of CSI-RS resources
    names = {'CSI-RS resource #0','CSI-RS resource #1'};
    chpval = {20,2};
    chpscale = 0.25*length(cmap); % Scaling factor
    tempGrid = zeros(gridSize);
    tempGrid(csirsInd{1}) = chpval{1};
    tempGrid(csirsInd{2}) = chpval{2};

    image(chpscale*tempGrid(:,:,1)); % Multiplied with scaling factor for better visualization
    axis xy;
    clevels = chpscale*[chpval{:}];
    N = length(clevels);
    L = line(ones(N),ones(N),'LineWidth',8); % Generate lines
```

```
    % Index the color map and associate the selected colors with the lines
    set(L,{'color'},mat2cell(cmap( min(1+clevels,length(cmap) ),:),ones(1,N),3)); % Set the colo
    % Create legend
    legend(names{:});

    title('Carrier Grid Containing CSI-RS')
    xlabel('OFDM Symbols');
    ylabel('Subcarriers');
end

function plotCSIRSMeasurements(meas)
%   plotCSIRSMeasurements(MEAS) plots CSI-RS based RSRP/RSSI/RSRQ measurements
    meas = structfun(@(s)max(s,[],1),meas,'UniformOutput',false);
    numRes = numel(meas.RSRPPerAntenna);
    xTickLabels = {};
    for idx = 1:numRes
        xTickLabels = [xTickLabels {['CSI-RS resource #' num2str(idx-1)]}]; %#ok<AGROW>
    end

    measType = {'CSI-RSRP','CSI-RSSI','CSI-RSRQ'};
    measVal = {meas.RSRPPerAntenna, meas.RSSIPerAntenna, meas.RSRQPerAntenna};
    measUnits = {'dBm','dBm','dB'};

    for measIdx = 1:3
        figure()
        values = measVal{measIdx};
        baseVal = 0;
        if ~any(values > 0)
            baseVal = floor(min(values));
        end
        b = bar(values,'FaceColor','#EDB120','BaseValue',baseVal);
        xticklabels(xTickLabels);

        xtips = b.XEndPoints;
        ytips = b.YEndPoints;
        for i = 1:numel(xtips)
            text(xtips(i), ytips(i), sprintf(['%0.2f ' measUnits{measIdx}],values(i)), ...
                'HorizontalAlignment', 'Center', 'VerticalAlignment', 'Top');
        end
        ylabel([measType{measIdx} ' in ' measUnits{measIdx}]);
        title([measType{measIdx} ' measurements (in ' measUnits{measIdx} ')'])
    end
end
```

References

[1] 3GPP TS 38.133. “NR; Requirements for support of radio resource management.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.215. “NR; Physical layer measurements.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions

`nrCSIRS` | `nrCSIRSIndices`

Objects
`nrCSIRSConfig` | `nrCarrierConfig`

Related Examples

- “NR Channel Estimation Using CSI-RS” on page 1-83

NR SSB Beam Sweeping

This example shows how to employ beam sweeping at both the transmitter (gNB) and receiver (UE) ends of a 5G NR system. Using synchronization signal blocks (SSB), this example illustrates some of the beam management procedures used during initial access. To accomplish beam sweeping, the example uses several components from Phased Array System Toolbox™.

Introduction

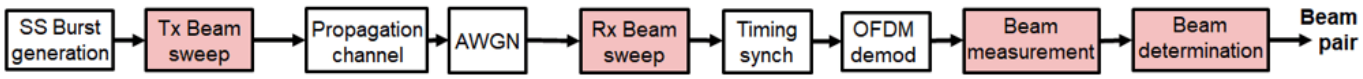
The support of millimeter wave (mmWave) frequencies requires directional links, which led to the specification of beam management procedures for initial access in NR. Beam management is a set of Layer 1 (physical) and Layer 2 (medium access control) procedures to acquire and maintain a set of beam pair links (a beam used at gNB paired with a beam used at UE). Beam management procedures are applied for both downlink and uplink transmission and reception [1], [2]. These procedures include:

- Beam sweeping
- Beam measurement
- Beam determination
- Beam reporting
- Beam recovery

This example focuses on initial access procedures for idle users when a connection is established between the user equipment (UE) and access network node (gNB). At the physical layer, using synchronization signal blocks (SSB) transmitted as a burst in the downlink direction (gNB to UE), the example highlights both transmit/receive point (TRP) beam sweeping and UE beam sweeping to establish a beam pair link. Among the multiple beam management procedures, TR 38.802 defines this dual-end sweep as procedure P-1 [1].

Once connected, the same beam pair link can be used for subsequent transmissions. If necessary, the beams are further refined using CSI-RS (for downlink) and SRS (for uplink). In case of beam failure, these pair links can be reestablished. For an example of beam pair refinement, see “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113.

This example generates an NR synchronization signal burst, beamforms each of the SSBs within the burst to sweep over both azimuth and elevation directions, transmits this beamformed signal over a spatial scattering channel, and processes this received signal over the multiple receive-end beams. The example measures the reference signal received power (RSRP) for each of the transmit-receive beam pairs (in a dual loop) and determines the beam pair link with the maximum RSRP. This beam pair link thus signifies the best beam-pair at transmit and receive ends for the simulated spatial scenario. This figure shows the main processing steps with the beam management ones highlighted in color.




```
rng(211);                           % Set RNG state for repeatability
```

Simulation Parameters

Define system parameters for the example. Modify these parameters to explore their impact on the system.

```
prm.NCellID = 1;                    % Cell ID
prm.FreqRange = 'FR1';              % Frequency range: 'FR1' or 'FR2'
prm.CenterFreq = 3.5e9;             % Hz
prm.SSBlockPattern = 'Case B';      % Case A/B/C/D/E
prm.SSBTransmitted = [ones(1,8) zeros(1,0)];   % 4/8 or 64 in length

prm.TxArraySize = [8 8];            % Transmit array size, [rows cols]
prm.TxAZlim = [-60 60];             % Transmit azimuthal sweep limits
prm.TxELlim = [-90 0];              % Transmit elevation sweep limits

prm.RxArraySize = [2 2];            % Receive array size, [rows cols]
prm.RxAZlim = [-180 180];           % Receive azimuthal sweep limits
prm.RxELlim = [0 90];               % Receive elevation sweep limits

prm.ElevationSweep = false;         % Enable/disable elevation sweep
prm.SNRdB = 30;                     % SNR, dB
prm.RSRPMode = 'SSSwDMRS';          % {'SSSwDMRS', 'SSSonly'}
```

The example uses these parameters:

- Cell ID for a single-cell scenario with a single BS and UE
- Frequency range, as a string to designate FR1 or FR2 operation
- Center frequency, in Hz, dependent on the frequency range
- Synchronization signal block pattern as one of Case A/B/C for FR1 and Case D/E for FR2. This also selects the subcarrier spacing.
- Transmitted SSBs in the pattern, as a binary vector of length 4 or 8 for FR1 and length 64 for FR2. The number of SSBs transmitted sets the number of beams at both transmit and receive ends.
- Transmit array size, as a two-element row vector specifying the number of antenna elements in the rows and columns of the transmit array, respectively. A uniform rectangular array (URA) is used when both values are greater than one.
- Transmit azimuthal sweep limits in degrees to specify the starting and ending azimuth angles for the sweep
- Transmit elevation sweep limits in degrees to specify the starting and ending elevation angles for the sweep
- Receive array size, as a two-element row vector specifying the number of antenna elements in the rows and columns of the receive array, respectively. A uniform rectangular array (URA) is used when both values are greater than one.
- Receive azimuthal sweep limits in degrees to specify the starting and ending azimuth angles for the sweep
- Receive elevation sweep limits in degrees to specify the starting and ending elevation angles for the sweep
- Enable or disable elevation sweep for both transmit and receive ends. Enable elevation sweep for FR2 and/or URAs
- Signal-to-noise ratio in dB
- Measurement mode for SSB to specify the use of only secondary synchronization signals ('SSSonly') or use of PBCH DM-RS along with secondary synchronization signals ('SSSwDMRS')

```
prm = validateParams(prm);
```

Synchronization Signal Burst Configuration

Set up the synchronization signal burst parameters by using the specified system parameters. For initial access, set the SSB periodicity to 20 ms.

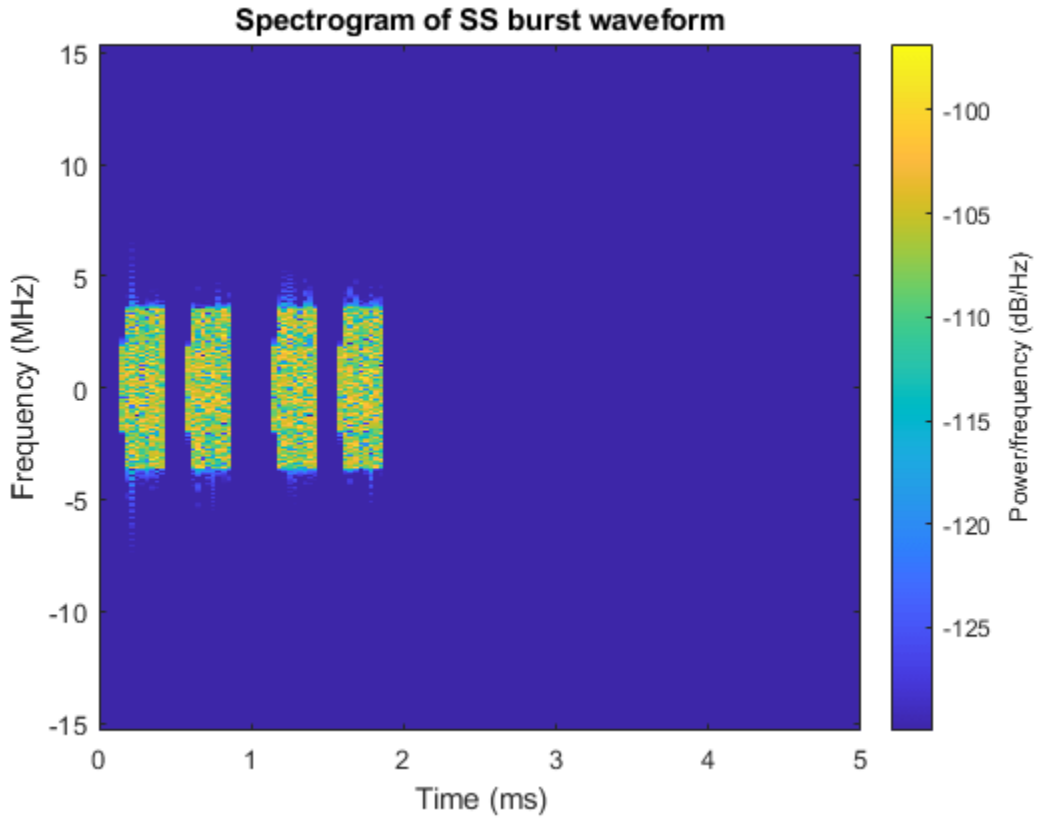
```
txBurst = nrWavegenSSBurstConfig;
txBurst.BlockPattern = prm.SSBlockPattern;
txBurst.TransmittedBlocks = prm.SSBTransmitted;
txBurst.Period = 20;
txBurst.SubcarrierSpacingCommon = prm.SubcarrierSpacingCommon;

% Configure an nrDLCarrierConfig object to use the synchronization signal
% burst parameters and to disable other channels. This object will be used
% by nrWaveformGenerator to generate the SS burst waveform.

cfgDL = configureWaveformGenerator(prm,txBurst);
```

Refer to the tutorial “Synchronization Signal Blocks and Bursts” for more details on synchronization signal blocks and bursts.

Burst Generation

Create the SS burst waveform [3] by calling the `nrWaveformGenerator` function. The generated waveform is not yet beamformed.

```
burstWaveform = nrWaveformGenerator(cfgDL);

% Display spectrogram of SS burst waveform
figure;
ofdmInfo = nrOFDMInfo(cfgDL.SCSCarriers{1}.NSizeGrid,prm.SCS);
nfft = ofdmInfo.Nfft;
spectrogram(burstWaveform,ones(nfft,1),0,nfft,'centered',ofdmInfo.SampleRate,'yaxis','MinThresho
title('Spectrogram of SS burst waveform')
```



Channel Configuration

Configure a spatial scattering MIMO channel `channel`. This channel model applies free space path loss and, optionally, other atmospheric attenuations to the input. Specify the locations for the BS and UE as `[x,y,z]` coordinates in a Cartesian system. Depending on the array sizes specified, employ either uniform linear arrays (ULA) or uniform rectangular arrays (URA). Use isotropic antenna elements for the arrays.

```
c = physconst('LightSpeed');   % Propagation speed
lambda = c/prm.CenterFreq;     % Wavelength

prm.posTx = [0;0;0];           % Transmit array position, [x;y;z], meters
prm.posRx = [100;50;0];        % Receive array position, [x;y;z], meters

toRxRange = rangeangle(prm.posTx,prm.posRx);
spLoss = fspl(toRxRange,lambda);    % Free space path loss

% Transmit array
if prm.IsTxURA
    % Uniform rectangular array
    arrayTx = phased.URA(prm.TxArraySize,0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement('BackBaffled',true));
else
    % Uniform linear array
    arrayTx = phased.ULA(prm.NumTx, ...
        'ElementSpacing',0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement('BackBaffled',true));
```

```
end

% Receive array
if prm.IsRxURA
    % Uniform rectangular array
    arrayRx = phased.URA(prm.RxArraySize,0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement);
else
    % Uniform linear array
    arrayRx = phased.ULA(prm.NumRx, ...
        'ElementSpacing',0.5*lambda, ...
        'Element',phased.IsotropicAntennaElement);
end

% Scatterer locations
prm.FixedScatMode = true;
if prm.FixedScatMode
    % Fixed single scatterer location
    prm.ScatPos = [50; 80; 0];
else
    % Generate scatterers at random positions
    Nscat = 10;        % Number of scatterers
    azRange = -180:180;
    elRange = -90:90;
    randAzOrder = randperm(length(azRange));
    randElOrder = randperm(length(elRange));
    azAngInSph = azRange(randAzOrder(1:Nscat));
    elAngInSph = elRange(randElOrder(1:Nscat));
    r = 20;            % radius
    [x,y,z] = sph2cart(deg2rad(azAngInSph),deg2rad(elAngInSph),r);
    prm.ScatPos = [x;y;z] + (prm.posTx + prm.posRx)/2;
end

% Configure channel
channel = phased.ScatteringMIMOChannel;
channel.PropagationSpeed = c;
channel.CarrierFrequency = prm.CenterFreq;
channel.SampleRate = ofdmInfo.SampleRate;
channel.SimulateDirectPath = false;
channel.ChannelResponseOutputPort = true;
channel.Polarization = 'None';
channel.TransmitArray = arrayTx;
channel.TransmitArrayPosition = prm.posTx;
channel.ReceiveArray = arrayRx;
channel.ReceiveArrayPosition = prm.posRx;
channel.ScattererSpecificationSource = 'Property';
channel.ScattererPosition = prm.ScatPos;
channel.ScattererCoefficient = ones(1,size(prm.ScatPos,2));

% Get maximum channel delay
[~,~,tau] = channel(complex(randn(ofdmInfo.SampleRate*1e-3,prm.NumTx), ...
    randn(ofdmInfo.SampleRate*1e-3,prm.NumTx)));
maxChDelay = ceil(max(tau)*ofdmInfo.SampleRate);
```

Transmit-End Beam Sweeping

To achieve TRP beam sweeping, beamform each of the SS blocks in the generated burst using analog beamforming. Based on the number of SS blocks in the burst and the sweep ranges specified,

determine both the azimuth and elevation directions for the different beams. Then beamform the individual blocks within the burst to each of these directions.

```
% Number of beams at both transmit and receive ends
numBeams = sum(txBurst.TransmittedBlocks);

% Transmit beam angles in azimuth and elevation, equi-spaced
azBW = beamwidth(arrayTx,prm.CenterFreq,'Cut','Azimuth');
elBW = beamwidth(arrayTx,prm.CenterFreq,'Cut','Elevation');
txBeamAng = hGetBeamSweepAngles(numBeams,prm.TxAZlim,prm.TxELlim, ...
    azBW,elBW,prm.ElevationSweep);

% For evaluating transmit-side steering weights
SteerVecTx = phased.SteeringVector('SensorArray',arrayTx, ...
    'PropagationSpeed',c);

% Get the set of OFDM symbols occupied by each SSB
numBlocks = length(txBurst.TransmittedBlocks);
burstStartSymbols = ssBurstStartSymbols(txBurst.BlockPattern,numBlocks);
burstStartSymbols = burstStartSymbols(txBurst.TransmittedBlocks==1);
burstOccupiedSymbols = burstStartSymbols.' + (1:4);

% Apply steering per OFDM symbol for each SSB
gridSymLengths = repmat(ofdmInfo.SymbolLengths,1,cfgDL.NumSubframes);
%   repeat burst over numTx to prepare for steering
strTxWaveform = repmat(burstWaveform,1,prm.NumTx)./sqrt(prm.NumTx);
for ssb = 1:numBeams

    % Extract SSB waveform from burst
    blockSymbols = burstOccupiedSymbols(ssb,:);
    startSSBInd = sum(gridSymLengths(1:blockSymbols(1)-1))+1;
    endSSBInd = sum(gridSymLengths(1:blockSymbols(4)));
    ssbWaveform = strTxWaveform(startSSBInd:endSSBInd,1);

    % Generate weights for steered direction
    wT = SteerVecTx(prm.CenterFreq,txBeamAng(:,ssb));

    % Apply weights per transmit element to SSB
    strTxWaveform(startSSBInd:endSSBInd,:) = ssbWaveform.*(wT');

end
```

The beamformed burst waveform is then transmitted over the spatially-aware scattering channel.

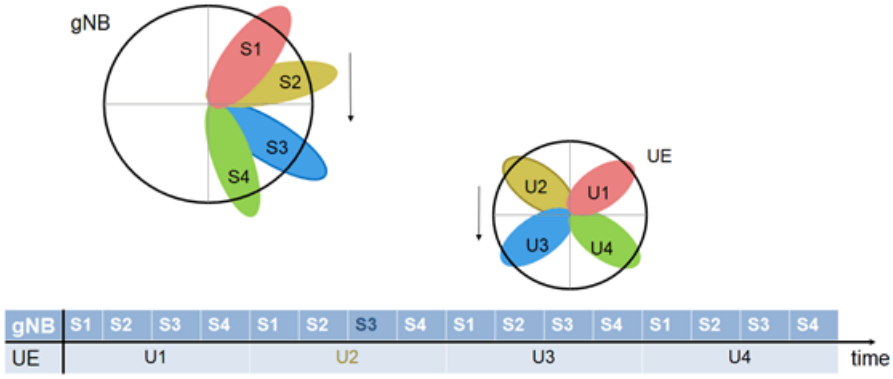
Receive-End Beam Sweeping and Measurement

For receive-end beam sweeping, the transmitted beamformed burst waveform is received successively over each receive beam. For `N` transmit beams and `M` receive beams in procedure P-1, each of the `N` beams is transmitted `M` times from gNB so that each transmit beam is received over the `M` receive beams.

The example assumes both `N` and `M` to be equal to the number of SSBs in the burst. For simplicity, the example generates only one burst, but to mimic the burst reception over the air `M` times, the receiver processes this single burst `M` times.

This figure shows a beam-based diagram for the sweeps at both gNB and UE for `N = M = 4`, in the azimuthal plane. The diagram shows the time taken for the dual sweep, where each interval at gNB

corresponds to an SSB and each interval at the UE corresponds to the SS burst. For the depicted scenario, beams `S3` and `U2` are highlighted as the selected beam-pair link notionally. The example implements the dual-sweep over a time duration of `N*M` time instants.



The receive processing of the transmitted burst includes

- Application of the spatially-aware fading channel
- Receive gain to compensate for the induced path loss and AWGN
- Receive-end beamforming
- Timing correction
- OFDM demodulation
- Extracting the known SSB grid
- Measuring the RSRP based on the specified measurement mode

The processing repeats these steps for each of the receive beams, then selects the best beam-pair based on the complete set of measurements made.

To highlight beam sweeping, the example assumes known SSB information at the receiver. For more details on recovery processing see “NR Cell Search and MIB and SIB1 Recovery” on page 1-30.

For the idle mode SS-RSRP measurement, use either only the secondary synchronization signals (SSS) or the physical broadcast channel (PBCH) demodulation reference signals (DM-RS) in addition to the SSS (Section 5.1.1. of [4]). Specify this by the `RSRPMode` parameter of the example. For FR2, the RSRP measurement is based on the combined signal from antenna elements, while the measurement is per antenna element for FR1.

```
% Receive beam angles in azimuth and elevation, equi-spaced
azBW = beamwidth(arrayRx,prm.CenterFreq,'Cut','Azimuth');
elBW = beamwidth(arrayRx,prm.CenterFreq,'Cut','Elevation');
rxBeamAng = hGetBeamSweepAngles(numBeams,prm.RxAZlim,prm.RxELlim, ...
    azBW,elBW,prm.ElevationSweep);

% For evaluating receive-side steering weights
SteerVecRx = phased.SteeringVector('SensorArray',arrayRx, ...
    'PropagationSpeed',c);

% AWGN level
SNR = 10^(prm.SNRdB/20);                          % Convert to linear gain
N0 = 1/(sqrt(2.0*prm.NumRx*double(ofdmInfo.Nfft))*SNR); % Noise Std. Dev.
```

```
% Receive gain in linear terms, to compensate for the path loss
rxGain = 10^(spLoss/20);

% Generate a reference grid for timing correction
%   assumes an SSB in first slot
carrier = nrCarrierConfig('NCellID',prm.NCellID);
carrier.NSizeGrid = cfgDL.SCSCarriers{1}.NSizeGrid;
carrier.SubcarrierSpacing = prm.SCS;
pssRef = nrPSS(carrier.NCellID);
pssInd = nrPSSIndices;
ibar_SSB = 0;
pbchdmrsRef = nrPBCHDMRS(carrier.NCellID,ibar_SSB);
pbchDMRSInd = nrPBCHDMRSIndices(carrier.NCellID);
pssGrid = zeros([240 4]);
pssGrid(pssInd) = pssRef;
pssGrid(pbchDMRSInd) = pbchdmrsRef;
refGrid = zeros([12*carrier.NSizeGrid ofdmInfo.SymbolsPerSlot]);
burstOccupiedSubcarriers = carrier.NSizeGrid*6 + (-119:120).';
refGrid(burstOccupiedSubcarriers, ...
    burstOccupiedSymbols(1,:)) = pssGrid;

% Loop over all receive beams
rsrp = zeros(numBeams,numBeams);
for rIdx = 1:numBeams

    % Fading channel, with path loss
    txWave = [strTxWaveform; zeros(maxChDelay,size(strTxWaveform,2))];
    fadWave = channel(txWave);

    % Receive gain, to compensate for the path loss
    fadWaveG = fadWave*rxGain;

    % Add WGN
    noise = N0*complex(randn(size(fadWaveG)),randn(size(fadWaveG)));
    rxWaveform = fadWaveG + noise;

    % Generate weights for steered direction
    wR = SteerVecRx(prm.CenterFreq,rxBeamAng(:,rIdx));

    % Apply weights per receive element
    if strcmp(prm.FreqRange, 'FR1')
        strRxWaveform = rxWaveform.*(wR');
    else  % for FR2, combine signal from antenna elements
        strRxWaveform = rxWaveform*conj(wR);
    end

    % Correct timing
    offset = nrTimingEstimate(carrier, ...
        strRxWaveform(1:ofdmInfo.SampleRate*1e-3,:),refGrid*wR(1)');
    if offset > maxChDelay
        offset = 0;
    end
    strRxWaveformS = strRxWaveform(1+offset:end,:);

    % OFDM Demodulate
    rxGrid = nrOFDMDemodulate(carrier,strRxWaveformS);

    % Loop over all SSBs in rxGrid (transmit end)
```

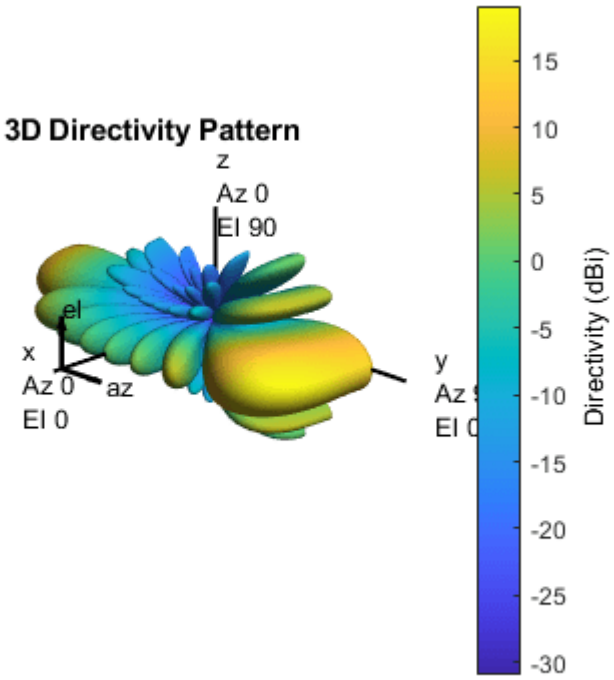
```
    for tIdx = 1:numBeams
        % Get each SSB grid
        rxSSBGrid = rxGrid(burstOccupiedSubcarriers, ...
            burstOccupiedSymbols(tIdx,:),:);

        if strcmpi(prm.RSRPMode,'SSSwDMRS')
            meas = nrSSBMeasurements(rxSSBGrid,carrier.NCellID,mod(tIdx-1,8));
        else
            meas = nrSSBMeasurements(rxSSBGrid,carrier.NCellID);
        end
        rsrp(rIdx,tIdx) = max(meas.RSRPPerAntenna);
    end
end
```

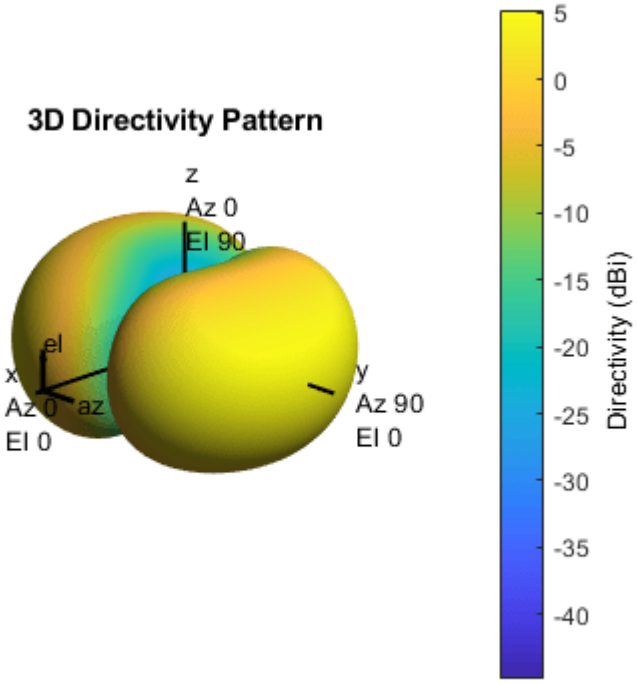
Beam Determination

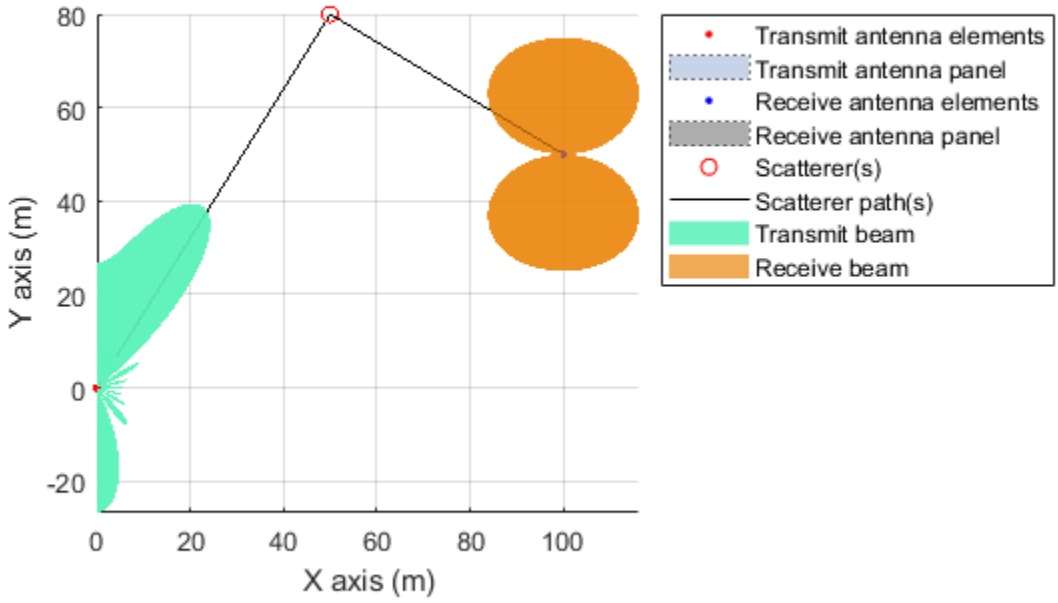
After the dual-end sweep and measurements are complete, determine the best beam-pair link based on the RSRP measurement.

```
[m,i] = max(rsrp,[],'all','linear');    % First occurrence is output
% i is column-down first (for receive), then across columns (for transmit)
[rxBeamID,txBeamID] = ind2sub([numBeams numBeams],i(1));

% Display the selected beam pair
disp(['Selected Beam pair with RSRP: ' num2str(rsrp(rxBeamID, ...
    txBeamID)) ' dBm', 13 '  Transmit #' num2str(txBeamID) ...
    ' (Azimuth: ' num2str(txBeamAng(1,txBeamID)) ', Elevation: ' ...
    num2str(txBeamAng(2,txBeamID)) ')' 13 '  Receive #' num2str(rxBeamID) ...
    ' (Azimuth: ' num2str(rxBeamAng(1,rxBeamID)) ', Elevation: ' ...
    num2str(rxBeamAng(2,rxBeamID)) ')' ]);

% Display final beam pair patterns
h = figure('Position',figposition([32 55 32 40]),'MenuBar','none');
h.Name = 'Selected Transmit Array Response Pattern';
wT = SteerVecTx(prm.CenterFreq,txBeamAng(:,txBeamID));
pattern(arrayTx,prm.CenterFreq,'PropagationSpeed',c,'Weights',wT);

h = figure('Position',figposition([32 55 32 40]),'MenuBar','none');
h.Name = 'Selected Receive Array Response Pattern';
wR = SteerVecRx(prm.CenterFreq,rxBeamAng(:,rxBeamID));
pattern(arrayRx,prm.CenterFreq,'PropagationSpeed',c,'Weights',wR);

% Plot MIMO scenario with tx, rx, scatterers, and determined beams. Beam
% patterns in this figure resemble the power patterns in linear scale.
prmScene = struct();
prmScene.TxArray = arrayTx;
prmScene.RxArray = arrayRx;
prmScene.TxArrayPos = prm.posTx;
prmScene.RxArrayPos = prm.posRx;
prmScene.ScatterersPos = prm.ScatPos;
prmScene.Lambda = lambda;
prmScene.ArrayScaling = 1;     % To enlarge antenna arrays in the plot
prmScene.MaxTxBeamLength = 45; % Maximum length of transmit beams in the plot
prmScene.MaxRxBeamLength = 25; % Maximum length of receive beam in the plot
hPlotSpatialMIMOScene(prmScene,wT,wR);
if ~prm.ElevationSweep
    view(2);
end
```

```
Selected Beam pair with RSRP: 45.0787 dBm  Transmit #8 (Azimuth: 60, Elevation: 0)  Receive #6 (A
```




3D Directivity Pattern
z
Az 0
El 90
el
x
Az 0
az
El 0
y
Az 90
El 0
5
0
-5
-10
-15
-20
-25
-30
-35
-40
Directivity (dBi)



These plots highlight the transmit directivity pattern, receive directivity pattern, and the spatial scene, respectively. The results are dependent on the individual beam directions used for the sweeps. The spatial scene offers a combined view of the transmit and receive arrays and the respective determined beams, along with the scatterers.

Summary and Further Exploration

This example highlights the P-1 beam management procedure by using synchronization signal blocks for transmit-end and receive-end beam sweeping. By measuring the reference signal received power for SSBs, you can identify the best beam pair link for a selected spatial environment.

The example allows variation on frequency range, SSB block pattern, number of SSBs, transmit and receive array sizes, transmit and receive sweep ranges, and the measuring mode. To see the impact of parameters on the beam selection, experiment with different values. The receive processing is simplified to highlight the beamforming aspects for the example.

For an example of the P-2 procedures of transmit-end beam sweeping using CSI-RS signals for the downlink, see “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113. You can use these procedures for beam refinement and adjustment in the connected mode, once the initial beam pair links are established [5], [6].

References

1 3GPP TR 38.802. "Study on New Radio access technology physical layer aspects." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

2 Giordani, M., M. Polese, A. Roy, D. Castor, and M. Zorzi. "A tutorial on beam management for 3GPP NR at mmWave frequencies." *IEEE Comm. Surveys & Tutorials*, vol. 21, No. 1, Q1 2019.

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

4 3GPP TS 38.215. "NR; Physical layer measurements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

5 Giordani, M., M. Polese, A. Roy, D. Castor, and M. Zorzi. "Standalone and non-standalone beam management for 3GPP NR at mmWaves." *IEEE Comm. Mag.*, April 2019, pp. 123-129.

6 Onggosanusi, E., S. Md. Rahman, et al. "Modular and high-resolution channel state information and beam management for 5G NR." *IEEE Comm. Mag.*, March 2018, pp. 48-55.

Local Functions

```
function prm = validateParams(prm)
% Validate user specified parameters and return updated parameters
%
% Only cross-dependent checks are made for parameter consistency.

    if strcmpi(prm.FreqRange,'FR1')
        if prm.CenterFreq > 7.125e9 || prm.CenterFreq < 410e6
            error(['Specified center frequency is outside the FR1 ', ...
                   'frequency range (410 MHz - 7.125 GHz).']);
        end
        if strcmpi(prm.SSBlockPattern,'Case D') ||  ...
           strcmpi(prm.SSBlockPattern,'Case E')
            error(['Invalid SSBlockPattern for selected FR1 frequency ' ...
                'range. SSBlockPattern must be one of ''Case A'' or ' ...
                '''Case B'' or ''Case C'' for FR1.']);
        end
        if ~((length(prm.SSBTransmitted)==4) || ...
             (length(prm.SSBTransmitted)==8))
            error(['SSBTransmitted must be a vector of length 4 or 8', ...
                   'for FR1 frequency range.']);
        end
        if (prm.CenterFreq <= 3e9) && (length(prm.SSBTransmitted)~=4)
            error(['SSBTransmitted must be a vector of length 4 for ' ...
                   'center frequency less than or equal to 3GHz.']);
        end
        if (prm.CenterFreq > 3e9) && (length(prm.SSBTransmitted)~=8)
            error(['SSBTransmitted must be a vector of length 8 for ', ...
                   'center frequency greater than 3GHz and less than ', ...
                   'or equal to 7.125GHz.']);
        end
    else % 'FR2'
        if prm.CenterFreq > 52.6e9 || prm.CenterFreq < 24.25e9
            error(['Specified center frequency is outside the FR2 ', ...
                   'frequency range (24.25 GHz - 52.6 GHz).']);
        end
        if ~(strcmpi(prm.SSBlockPattern,'Case D') || ...
                strcmpi(prm.SSBlockPattern,'Case E'))
            error(['Invalid SSBlockPattern for selected FR2 frequency ' ...
                'range. SSBlockPattern must be either ''Case D'' or ' ...
                '''Case E'' for FR2.']);
        end
        if length(prm.SSBTransmitted)~=64
            error(['SSBTransmitted must be a vector of length 64 for ', ...
```

```
                    'FR2 frequency range.']);
        end
    end

    prm.NumTx = prod(prm.TxArraySize);
    prm.NumRx = prod(prm.RxArraySize);
    if prm.NumTx==1 || prm.NumRx==1
        error(['Number of transmit or receive antenna elements must be', ...
               ' greater than 1.']);
    end
    prm.IsTxURA = (prm.TxArraySize(1)>1) && (prm.TxArraySize(2)>1);
    prm.IsRxURA = (prm.RxArraySize(1)>1) && (prm.RxArraySize(2)>1);

    if ~( strcmpi(prm.RSRPMode,'SSSonly') || ...
          strcmpi(prm.RSRPMode,'SSSwDMRS') )
        error(['Invalid RSRP measuring mode. Specify either ', ...
               '''SSSonly'' or ''SSSwDMRS'' as the mode.']);
    end

    % Select SCS based on SSBlockPattern
    switch lower(prm.SSBlockPattern)
        case 'case a'
            scs = 15;
            cbw = 10;
            scsCommon = 15;
        case {'case b', 'case c'}
            scs = 30;
            cbw = 25;
            scsCommon = 30;
        case 'case d'
            scs = 120;
            cbw = 100;
            scsCommon = 120;
        case 'case e'
            scs = 240;
            cbw = 200;
            scsCommon = 120;
    end
    prm.SCS = scs;
    prm.ChannelBandwidth = cbw;
    prm.SubcarrierSpacingCommon = scsCommon;

end

function ssbStartSymbols = ssBurstStartSymbols(ssbBlockPattern,Lmax)
% Starting OFDM symbols of SS burst.

    % 'alln' gives the overall set of SS block indices 'n' described in
    % TS 38.213 Section 4.1, from which a subset is used for each Case A-E
    alln = [0; 1; 2; 3; 5; 6; 7; 8; 10; 11; 12; 13; 15; 16; 17; 18];

    cases = {'Case A' 'Case B' 'Case C' 'Case D' 'Case E'};
    m = [14 28 14 28 56];
    i = {[2 8] [4 8 16 20] [2 8] [4 8 16 20] [8 12 16 20 32 36 40 44]};
    nn = [2 1 2 16 8];

    caseIdx = find(strcmpi(ssbBlockPattern,cases));
    if (any(caseIdx==[1 2 3]))
```

```
        if (Lmax==4)
            nn = nn(caseIdx);
        elseif (Lmax==8)
            nn = nn(caseIdx) * 2;
        end
    else
        nn = nn(caseIdx);
    end

    n = alln(1:nn);
    ssbStartSymbols = (i{caseIdx} + m(caseIdx)*n).';
    ssbStartSymbols = ssbStartSymbols(:).';

end

function cfgDL = configureWaveformGenerator(prm,txBurst)
% Configure an nrDLCarrierConfig object to be used by nrWaveformGenerator
% to generate the SS burst waveform.

    cfgDL = nrDLCarrierConfig;
    cfgDL.SCSCarriers{1}.SubcarrierSpacing = prm.SCS;
    if (prm.SCS==240)
        cfgDL.SCSCarriers = [cfgDL.SCSCarriers cfgDL.SCSCarriers];
        cfgDL.SCSCarriers{2}.SubcarrierSpacing = prm.SubcarrierSpacingCommon;
        cfgDL.BandwidthParts{1}.SubcarrierSpacing = prm.SubcarrierSpacingCommon;
    else
        cfgDL.BandwidthParts{1}.SubcarrierSpacing = prm.SCS;
    end
    cfgDL.PDSCH{1}.Enable = false;
    cfgDL.PDCCH{1}.Enable = false;
    cfgDL.ChannelBandwidth = prm.ChannelBandwidth;
    cfgDL.FrequencyRange = prm.FreqRange;
    cfgDL.NCellID = prm.NCellID;
    cfgDL.NumSubframes = 5;
    cfgDL.WindowingPercent = 0;
    cfgDL.SSBurst = txBurst;

end
```

See Also

Objects

`phased.ScatteringMIMOChannel` | `phased.SteeringVector`

Related Examples

- “NR Cell Search and MIB and SIB1 Recovery” on page 1-30
- “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113

NR Downlink Transmit-End Beam Refinement Using CSI-RS

This example demonstrates the downlink transmit-end beam refinement procedure using the channel state information reference signal (CSI-RS) from 5G Toolbox™. The example shows how to transmit multiple CSI-RS resources in different directions in a scattering environment and how to select the optimal transmit beam based on reference signal received power (RSRP) measurements.

Introduction

In NR 5G, frequency range 2 (FR2) operates at millimeter wave (mmWave) frequencies (24.25 GHz to 52.6 GHz). As the frequency increases, the transmitted signal is prone to high path loss and penetration loss, which affects the link budget. To improve the gain and directionality of the transmission and reception of the signals at higher frequencies, beamforming is essential. Beam management is a set of Layer 1 (physical layer) and Layer 2 (medium access control) procedures to establish and retain an optimal beam pair (transmit beam and a corresponding receive beam) for good connectivity. TR 38.802 Section 6.1.6.1 [1 on page 1-124] defines beam management as three procedures:

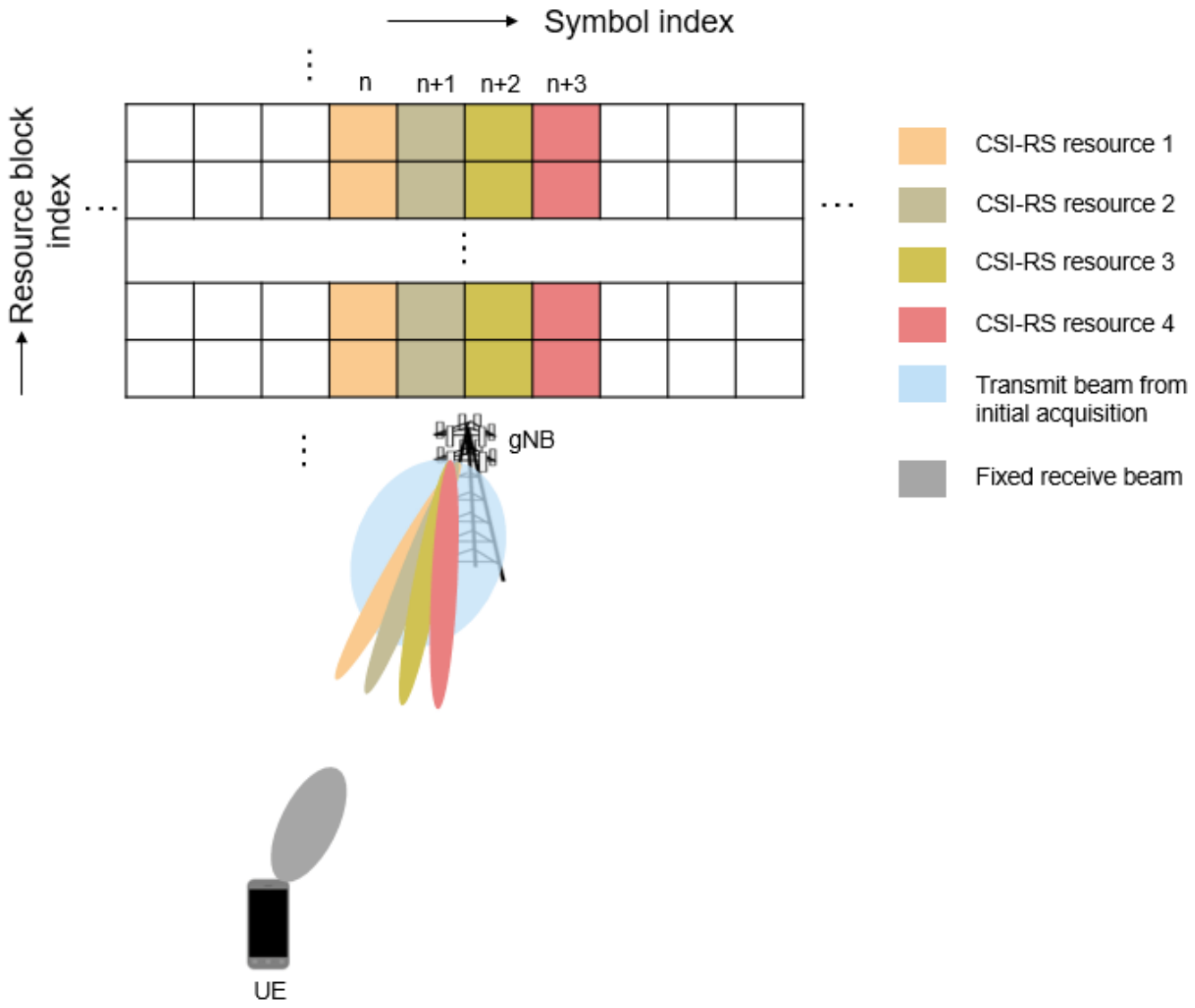
Procedure 1 (P-1): This procedure focuses on the initial acquisition based on synchronization signal blocks (SSB). During the initial acquisition, beam sweeping takes place at both transmit and receive ends to select the best beam pair based on the RSRP measurements. In general, the selected beams are wide and may not be an optimal beam pair for the data transmission and reception. For more details on this procedure, see “NR SSB Beam Sweeping” on page 1-98.

Procedure 2 (P-2): This procedure focuses on transmit-end beam refinement, where the beam sweeping happens at the transmit end by keeping the receive beam fixed. The procedure is based on non-zero-power CSI-RS (NZP-CSI-RS) for downlink transmit-end beam refinement and sounding reference signal (SRS) for uplink transmit-end beam refinement.

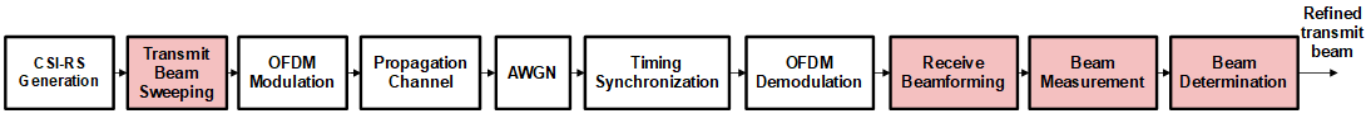
After the initial beam establishment, obtaining a unicast data transmission with high directivity and high gain requires a beam much finer than the SSB beam. Therefore, a set of reference signal resources are configured and transmitted in different directions by using finer beams within the angular range of the beam from the initial acquisition process. Then the user equipment (UE) or the access network node (gNB) measures all these beams by capturing the signals with a fixed receive beam. Finally, the best transmit beam is selected based on the RSRP measurements on all transmit beams.

Procedure 3 (P-3): This procedure focuses on receive-end beam adjustment, where the beam sweeping happens at the receive end given the current transmit beam. This process aims to find the best receive beam, which can be a neighbor beam or a refined beam. For this procedure, a set of reference signal resources (NZP-CSI-RS for downlink and SRS for uplink) are transmitted with the same transmit beam and the UE or gNB receives the signal using different beams from different directions covering an angular range. Finally, the best receive beam is selected based on the RSRP measurements on all receive beams.

This example focuses on downlink beam refinement at the transmitter. The example works for both frequency range 1 (FR1) and frequency range 2 (FR2) of NR 5G. This figure depicts the transmit-end beam refinement procedure, considering four NZP-CSI-RS resources transmitted in four different directions.



This figure shows the main processing steps of this example with the transmit-end beam refinement process-related steps in color.



Generate CSI-RS Resources

Configure Carrier

Create a carrier configuration object representing a 50 MHz carrier with subcarrier spacing of 30 kHz.

```
carrier = nrCarrierConfig;
% Maximum transmission bandwidth configuration for 50 MHz carrier with 30 kHz subcarrier spacing
```

```
carrier.NSizeGrid = 133;
carrier.SubcarrierSpacing = 30;
carrier.NSlot = 0;
carrier.NFrame = 0

carrier =
  nrCarrierConfig with properties:

             NCellID: 1
    SubcarrierSpacing: 30
         CyclicPrefix: 'normal'
            NSizeGrid: 133
          NStartGrid: 0
               NSlot: 0
              NFrame: 0

   Read-only properties:
       SymbolsPerSlot: 14
     SlotsPerSubframe: 2
        SlotsPerFrame: 20
```

Configure CSI-RS

Create a CSI-RS configuration object representing an NZP-CSI-RS resource set with `numNZPRes` number of NZP-CSI-RS resources. For Layer 1 RSRP measurements, configure all the CSI-RS resources in a resource set with the same number of antenna ports (either single-port or dual-port), as specified in TS 38.215 Section 5.1.2 [2 on page 1-124] or TS 38.214 Section 5.1.6.1.2 [3 on page 1-124]. This example works for single-port CSI-RS.

```
numNZPRes = 12;
csirs = nrCSIRSConfig;
csirs.CSIRSType = repmat({'nzp'},1,numNZPRes);
csirs.CSIRSPeriod = 'on';
csirs.Density = repmat({'one'},1,numNZPRes);
csirs.RowNumber = repmat(2,1,numNZPRes);
csirs.SymbolLocations = {0,1,2,3,4,5,6,7,8,9,10,11};
csirs.SubcarrierLocations = repmat({0},1,numNZPRes);
csirs.NumRB = 25

csirs =
  nrCSIRSConfig with properties:

             CSIRSType: {'nzp'  'nzp'  'nzp'  'nzp'  'nzp'  'nzp'  'nzp'  'nzp'  'nzp'  'nzp'
           CSIRSPeriod: 'on'
             RowNumber: [2 2 2 2 2 2 2 2 2 2 2 2]
               Density: {'one'  'one'  'one'  'one'  'one'  'one'  'one'  'one'  'one'  'one'
       SymbolLocations: {[0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]  [11]}
    SubcarrierLocations: {[0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]}
                 NumRB: 25
              RBOffset: 0
                   NID: 0

   Read-only properties:
          NumCSIRSPorts: [1 1 1 1 1 1 1 1 1 1 1 1]
                CDMType: {'noCDM'  'noCDM'  'noCDM'  'noCDM'  'noCDM'  'noCDM'  'noCDM'  'noCDM'
```

```
% Validate CSI-RS antenna ports
validateCSIRSPorts(csirs);

% Get the binary vector to represent the presence of each CSI-RS resource
% in a specified slot
csirsTransmitted = getActiveCSIRSRes(carrier,csirs);
```

Configure the power scaling of all NZP-CSI-RS resources in decibels (dB).

```
powerCSIRS = 0;
```

Generate CSI-RS Symbols and Indices

Generate CSI-RS symbols and indices by using the `carrier` and `csirs` configuration objects. To distinguish each CSI-RS resource output separately, specify the `OutputResourceFormat,'cell'` name-value pair.

```
csirsSym = nrCSIRS(carrier,csirs,'OutputResourceFormat','cell')
```

```
csirsSym=1×12 cell array
    {25x1 double}    {25x1 double}    {25x1 double}    {25x1 double}    {25x1 double}    {25x1 do
```

```
csirsInd = nrCSIRSIndices(carrier,csirs,'OutputResourceFormat','cell')
```

```
csirsInd=1×12 cell array
    {25x1 uint32}    {25x1 uint32}    {25x1 uint32}    {25x1 uint32}    {25x1 uint32}    {25x1 ui
```

Configure Antenna Arrays and Scatterers

Configure Transmit and Receive Antenna Arrays

Configure the carrier frequency and the signal propagation speed.

```
% Set the carrier frequency
fc = 3.5e9;
freqRange = validateFc(fc);

% Set the propagation speed
c = physconst('LightSpeed');
% Calculate wavelength
lambda = c/fc;
```

Configure the size of transmit and receive antenna arrays as a two-element vector, where the first element represents the number of rows and the second element represents the number of columns in the antenna array.

```
txArySize = [8 8];
rxArySize = [2 2];
```

Calculate the total number of transmit and receive antenna elements.

```
nTx = prod(txArySize);
nRx = prod(rxArySize);
```

Configure the positions of transmit and receive antenna arrays. Then calculate the free space path loss based on the spatial separation between transmit and receive antenna array positions.

```
% Configure antenna array positions
txArrayPos = [0;0;0];
rxArrayPos = [100;50;0];

% Calculate the free space path loss
toRxRange = rangeangle(txArrayPos,rxArrayPos);
spLoss = fspl(toRxRange,lambda);
```

Configure the uniform linear array (ULA) or uniform rectangular array (URA) based on the sizes of antennas arrays.

```
% Initialize the flags to choose between URA and ULA
isTxRectArray = false;
isRxRectArray = false;

% Enable isTxRectArray if both the number of rows and columns of transmit
% antenna array are greater than one
if ~any(txArySize == 1)
    isTxRectArray = true;
end
% Enable isRxRectArray if both the number of rows and columns of receive
% antenna array are greater than one
if ~any(rxArySize == 1)
    isRxRectArray = true;
end

% Configure the transmit and receive antenna elements
txAntenna = phased.IsotropicAntennaElement('BackBaffled',true);  % To avoid transmission beyond +
                                                                 % degrees from the broadside, ba
                                                                 % the back of the transmit anten
                                                                 % element by setting the BackBaf
                                                                 % property to true
rxAntenna = phased.IsotropicAntennaElement('BackBaffled',false); % To receive the signal from 360
                                                                 % set the BackBaffled property t

% Configure transmit antenna array
if isTxRectArray
    % Create a URA System object for signal transmission
    txArray = phased.URA('Element',txAntenna,'Size',txArySize,'ElementSpacing',lambda/2);
else
    % Create a ULA System object for signal transmission
    txArray = phased.ULA('Element',txAntenna,'NumElements',nTx,'ElementSpacing',lambda/2);
end

% Configure receive antenna array
if isRxRectArray
    % Create a URA System object for signal reception
    rxArray = phased.URA('Element',rxAntenna,'Size',rxArySize,'ElementSpacing',lambda/2);
else
    % Create a ULA System object for signal reception
    rxArray = phased.ULA('Element',rxAntenna,'NumElements',nRx,'ElementSpacing',lambda/2);
end
```

Configure Scatterers

```
fixedScatMode = true;
rng(42);
if fixedScatMode
```

```
    % Fixed single scatterer location
    numScat = 1;
    scatPos = [60;10;15];
else
    % Generate scatterers at random positions
    numScat = 10; %#ok<UNRCH>
    azRange = -180:180;
    randAzOrder = randperm(length(azRange));
    elRange = -90:90;
    randElOrder = randperm(length(elRange));
    azAngInSph = deg2rad(azRange(randAzOrder(1:numScat)));
    elAngInSph = deg2rad(elRange(randElOrder(1:numScat)));
    r = 20;

    % Transform spherical coordinates to Cartesian coordinates
    [x,y,z] = sph2cart(azAngInSph,elAngInSph,r);
    scatPos = [x;y;z] + (txArrayPos + rxArrayPos)/2;
end
```

Transmit Beamforming and OFDM Modulation

Calculate the Steering Vectors

Create the steering vector System object™ for transmit antenna array.

```
txArrayStv = phased.SteeringVector('SensorArray',txArray,'PropagationSpeed',c);
```

Calculate the angle of scatterer position with respect to the transmit antenna array.

```
[~,scatAng] = rangeangle(scatPos(:,1),txArrayPos); % Pointing towards the first scatterer positio
```

Configure the azimuth and elevation beamwidths of SSB transmit beam from the initial acquisition process (P-1).

```
azTxBeamWidth = 30; % In degrees
elTxBeamWidth = 30; % In degrees
```

Get the SSB transmit beam direction which is aligned (partially or fully) to the position of scatterer, by using the beamwidths in azimuth and elevation planes.

```
ssbTxAng = getInitialBeamDir(scatAng,azTxBeamWidth,elTxBeamWidth);
```

Calculate the beam directions (azimuth and elevation angle pairs) for all active CSI-RS resources within the angular range covered by the SSB transmit beam.

```
% Get the number of transmit beams based on the number of active CSI-RS resources in a slot
numBeams = sum(csirsTransmitted);

% Get the azimuthal sweep range based on the SSB transmit beam direction
% and its beamwidth in azimuth plane
azSweepRange = [ssbTxAng(1) - azTxBeamWidth/2 ssbTxAng(1) + azTxBeamWidth/2];

% Get the elevation sweep range based on the SSB transmit beam direction
% and its beamwidth in elevation plane
elSweepRange = [ssbTxAng(2) - elTxBeamWidth/2 ssbTxAng(2) + elTxBeamWidth/2];

% Get the azimuth and elevation angle pairs for all NZP-CSI-RS transmit beams
azBW = beamwidth(txArray,fc,'Cut','Azimuth');
```

```
elBW = beamwidth(txArray,fc,'Cut','Elevation');
csirsBeamAng = hGetBeamSweepAngles(numBeams,azSweepRange,elSweepRange,azBW,elBW);
```

Calculate the steering vectors for all active CSI-RS resources.

```
wT = zeros(nTx,numBeams);
for beamIdx = 1:numBeams
    tempW = txArrayStv(fc,csirsBeamAng(:,beamIdx));
    wT(:,beamIdx) = tempW;
end
```

Apply Digital Beamforming

Loop over all NZP-CSI-RS resources and apply the digital beamforming to all the active ones. Digital beamforming is considered to offer frequency selective beamforming within the same OFDM symbol.

```
% Number of CSI-RS antenna ports
ports = csirs.NumCSIRSPorts(1);
% Initialize the beamformed grid
bfGrid = nrResourceGrid(carrier,nTx);
% Get the active NZP-CSI-RS resource indices
activeRes = find(logical(csirsTransmitted));
for resIdx = 1:numNZPRes
    % Initialize the carrier resource grid for one slot and map NZP-CSI-RS symbols onto
    % the grid
    txSlotGrid = nrResourceGrid(carrier,ports);
    txSlotGrid(csirsInd{resIdx}) = db2mag(powerCSIRS)*csirsSym{resIdx};
    reshapedSymb = reshape(txSlotGrid,[],ports);

    % Get the transmit beam index
    beamIdx = find(activeRes == resIdx);

    % Apply the digital beamforming
    if ~isempty(beamIdx)
        bfSymb = reshapedSymb * wT(:,beamIdx)';
        bfGrid = bfGrid + reshape(bfSymb,size(bfGrid));
    end
end
```

Perform OFDM Modulation

Generate the time-domain waveform by performing the OFDM modulation.

```
% Perform OFDM modulation
[tbfWaveform,ofdmInfo] = nrOFDMModulate(carrier,bfGrid);

% Normalize the beamformed time-domain waveform over the number of transmit
% antennas
tbfWaveform = tbfWaveform/sqrt(nTx);
```

Scattering MIMO Channel and AWGN

Configure the Channel

Configure the scattering-based MIMO propagation channel by using the System object `phased.ScatteringMIMOChannel` (Phased Array System Toolbox). This channel model applies time delay, gain, Doppler shift, phase change, free space path loss, and optionally, other atmospheric attenuations to the input.

```
chan = phased.ScatteringMIMOChannel;
chan.PropagationSpeed = c;
chan.CarrierFrequency = fc;
chan.Polarization = 'none';
chan.SpecifyAtmosphere = false;
chan.SampleRate = ofdmInfo.SampleRate;
chan.SimulateDirectPath = false;
chan.ChannelResponseOutputPort = true;

% Configure transmit array parameters
chan.TransmitArray = txArray;
chan.TransmitArrayMotionSource = 'property';
chan.TransmitArrayPosition = txArrayPos;

% Configure receive array parameters
chan.ReceiveArray = rxArray;
chan.ReceiveArrayMotionSource = 'property';
chan.ReceiveArrayPosition = rxArrayPos;

% Configure scatterers
chan.ScattererSpecificationSource = 'property';
chan.ScattererPosition = scatPos;
chan.ScattererCoefficient = ones(1,numScat);

% Get the maximum channel delay by transmitting random signal
[~,~,tau] = chan(complex(randn(chan.SampleRate*1e-3,nTx), ...
    randn(chan.SampleRate*1e-3,nTx)));
maxChDelay = ceil(max(tau)*chan.SampleRate);
```

Send the Waveform through the Channel

Append zeros at the end of the transmitted waveform to flush the channel content and then pass the time-domain waveform through the scattering MIMO channel. These zeros take into account any delay introduced in the channel.

```
% Append zeros to the transmit waveform to account for channel delay
tbfWaveform = [tbfWaveform; zeros(maxChDelay,nTx)];
% Pass the waveform through the channel
fadWave = chan(tbfWaveform);
```

Apply AWGN

Configure and apply the receive gain to the faded waveform, to compensate for the path loss. Then apply AWGN to the resultant waveform. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
% Configure the receive gain
rxGain = 10.^((spLoss)/20); % Gain in linear scale
% Apply the gain
fadWaveG = fadWave*rxGain;

% Configure the SNR in dB
SNRdB = 20;
SNR = 10^(SNRdB/10); % SNR in linear scale
% Calculate the standard deviation for AWGN
N0 = 1/sqrt(2.0*nRx*double(ofdmInfo.Nfft)*SNR);

% Generate AWGN
```

```
noise = N0*complex(randn(size(fadWaveG)),randn(size(fadWaveG)));
% Apply AWGN to the waveform
rxWaveform = fadWaveG + noise;
```

Timing Synchronization

Perform the timing synchronization by cross correlating the received reference symbols with a local copy of NZP-CSI-RS symbols.

```
% Generate reference symbols and indices
refSym = nrCSIRS(carrier,csirs);
refInd = nrCSIRSIndices(carrier,csirs);

% Estimate timing offset
offset = nrTimingEstimate(carrier,rxWaveform,refInd,refSym);
if offset > maxChDelay
    offset = 0;
end

% Correct timing offset
syncTdWaveform = rxWaveform(1+offset:end,:);
```

OFDM Demodulation and Receive Beamforming

OFDM Demodulation

OFDM demodulate the synchronized time-domain waveform.

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

Calculate Steering Vector

Create the steering vector System object for the receive antenna array.

```
rxArrayStv = phased.SteeringVector('SensorArray',rxArray,'PropagationSpeed',c);
```

Calculate the angle of scatterer position with respect to the receive antenna array. Assuming this as receive beam direction from the initial acquisition process using SSB.

```
[~,scatRxAng] = rangeangle(scatPos(:,1),rxArrayPos); % Pointing towards the first scatterer posi
```

Configure the azimuth and elevation beamwidths of receive beam from the initial acquisition process (P-1).

```
azRxBeamWidth = 30; % In degrees
elRxBeamWidth = 30; % In degrees
```

Get the initial receive beam direction which is aligned (partially or fully) to the position of scatterer, by using the beamwidths in azimuth and elevation planes from P-1.

```
rxAng = getInitialBeamDir(scatRxAng,azRxBeamWidth,elRxBeamWidth);
```

Calculate the steering vector for the angle of reception.

```
wR = rxArrayStv(fc,rxAng);
```

Apply Receive Beamforming

To perform digital beamforming at the receiver side, apply the steering weights to `rxGrid`, with the assumption that there is no other signal present in `rxGrid` (single UE scenario). Combine the signals

from all receive antenna elements in case of FR2, as specified in TS 38.215 Section 5.1.2 [2 on page 1-124].

```
temp = rxGrid;
if strcmpi(freqRange,'FR1')
    % Beamforming without combining
    rbfGrid = reshape(reshape(temp,[],nRx).*wR',size(temp,1),size(temp,2),[]);
else % 'FR2'
    % Beamforming with combining
    rbfGrid = reshape(reshape(temp,[],nRx)*conj(wR),size(temp,1),size(temp,2),[]);
end
```

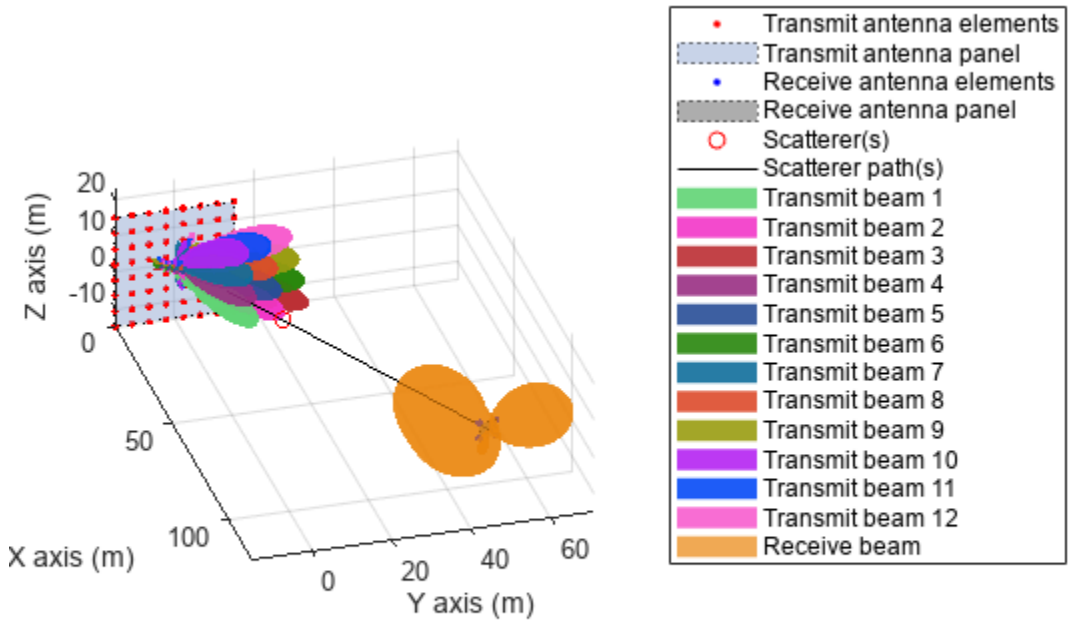
Plot the Scattering MIMO Scenario

Configure the MIMO scene parameters.

```
sceneParams.TxArray = txArray;
sceneParams.RxArray = rxArray;
sceneParams.TxArrayPos = txArrayPos;
sceneParams.RxArrayPos = rxArrayPos;
sceneParams.ScatterersPos = scatPos;
sceneParams.Lambda = lambda;
sceneParams.ArrayScaling = 100;   % To enlarge antenna arrays in the plot
sceneParams.MaxTxBeamLength = 45; % Maximum length of transmit beams in the plot
sceneParams.MaxRxBeamLength = 25; % Maximum length of receive beam in the plot
```

Plot the scattering MIMO scenario (including transmit and receive antenna arrays, scatterer positions and their paths, and all the transmit and receive antenna array beam patterns) by using the helper function `hPlotSpatialMIMOScene`. Beam patterns in this figure resemble the power patterns in linear scale.

```
hPlotSpatialMIMOScene(sceneParams,wT,wR);
axis tight;
view([74 29]);
```



Beam Determination

After the OFDM demodulation, the UE measures the RSRP for all the CSI-RS resources transmitted in different beams, given the current receive beam. Perform these measurements by using the `nrCSIRSMeasurements` function.

```
% Perform RSRP measurements
meas = nrCSIRSMeasurements(carrier,csirs,rbfGrid);

% Display the measurement quantities for all CSI-RS resources in dBm
RSRPdBm = max(meas.RSRPPerAntenna,[],1);
disp(['RSRP measurements of all CSI-RS resources (in dBm):' 13 num2str(RSRPdBm)]);
```

```
RSRP measurements of all CSI-RS resources (in dBm):
42.3147       33.237      29.1999      45.1102      37.0922      31.4861      39.9414      33.505
```

Identify the maximum RSRP value from the measurements and find the best corresponding beam.

```
% Get the transmit beam index with maximum RSRP value
[~,maxRSRPIdx] = max(RSRPdBm(logical(csirsTransmitted)));

% Get the CSI-RS resource index with maximum RSRP value
[~,maxRSRPResIdx] = max(RSRPdBm);
```

Calculate the beamwidth which corresponds to the refined transmit beam.

```
% Get the steering weights corresponding to refined transmit beam
if numBeams == 0
```

```
    disp('Refinement has not happened, as NZP-CSI-RS is not transmitted')
else
    refBeamWts = wT(:,maxRSRPIdx);
    csirsAzBeamWidth = beamwidth(txArray,fc,'PropagationSpeed',c,'Weights',refBeamWts,'CutAngle',
    csirsElBeamWidth = beamwidth(txArray,fc,'PropagationSpeed',c,'Weights',refBeamWts,'Cut','Elev
    disp(['From initial beam acquisition:' 13 '    Beamwidth of initial SSB beam in azimuth plane
        num2str(azTxBeamWidth) ' degrees' 13 ...
        '    Beamwidth of initial SSB beam in elevation plane is: '...
        num2str(elTxBeamWidth) ' degrees' 13 13 ...
        'With transmit-end beam refinement:' 13 '    Refined transmit beam ('...
        num2str(maxRSRPIdx) ') corresponds to CSI-RS resource '...
        num2str(maxRSRPResIdx) ' is selected in the direction ['...
        num2str(csirsBeamAng(1,maxRSRPIdx)) ';' num2str(csirsBeamAng(2,maxRSRPIdx))...
        ']' 13 '    Beamwidth of refined transmit beam in azimuth plane is: '...
        num2str(csirsAzBeamWidth) ' degrees' 13 ...
        '    Beamwidth of refined transmit beam in elevation plane is: '...
        num2str(csirsElBeamWidth) ' degrees']);
end
```

```
From initial beam acquisition:
    Beamwidth of initial SSB beam in azimuth plane is: 30 degrees
    Beamwidth of initial SSB beam in elevation plane is: 30 degrees

With transmit-end beam refinement:
    Refined transmit beam (4) corresponds to CSI-RS resource 4 is selected in the direction [10;1
    Beamwidth of refined transmit beam in azimuth plane is: 13.46 degrees
    Beamwidth of refined transmit beam in elevation plane is: 13.24 degrees
```

Summary and Further Exploration

This example highlights the beam refinement procedure (P-2) using NZP-CSI-RS. The procedure identifies a transmit beam that is finer than the beam from the initial acquisition.

You can configure multiple CSI-RS resources, transmit and receive antenna array configurations, and multiple scatterers to see the variations in the selection of the refined beam. You can also configure the azimuth and elevation angle pairs for the signal transmission and reception.

References

1. 3GPP TR 38.802. "Study on New Radio access technology physical layer aspects." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
2. 3GPP TS 38.215. "NR; Physical layer measurements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
3. 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local Functions

```
function validateCSIRSPorts(csirs)
%   validateCSIRSPorts validates the CSI-RS antenna ports, given the
%   CSI-RS configuration object CSIRS.

    numPorts = csirs.NumCSIRSPorts;
    if any(numPorts > 1)
        error('nr5g:PortsGreaterThan1','CSI-RS resources must be configured for single-port for P
```

```
    end
end

function csirsTransmitted = getActiveCSIRSRes(carrier,csirs)
%   getActiveCSIRSRes returns a binary vector indicating the presence of
%   all CSI-RS resources in a specified slot, given the carrier
%   configuration object CARRIER and CSI-RS configuration object CSIRS.

    % Extract the following properties of carrier
    NSlotA        = carrier.NSlot;          % Absolute slot number
    NFrameA       = carrier.NFrame;         % Absolute frame number
    SlotsPerFrame = carrier.SlotsPerFrame;  % Number of slots per frame

    % Calculate the appropriate frame number (0...1023) based on the
    % absolute slot number
    NFrameR = mod(NFrameA + fix(NSlotA/SlotsPerFrame),1024);
    % Relative slot number (0...slotsPerFrame-1)
    NSlotR = mod(NSlotA,SlotsPerFrame);

    % Loop over the number of CSI-RS resources
    numCSIRSRes = numel(csirs.CSIRSType);
    csirsTransmitted = zeros(1,numCSIRSRes);
    csirs_struct = validateConfig(csirs);
    for resIdx = 1:numCSIRSRes
        % Extract the CSI-RS slot periodicity and offset
        if isnumeric(csirs_struct.CSIRSPeriod{resIdx})
            Tcsi_rs = csirs_struct.CSIRSPeriod{resIdx}(1);
            Toffset = csirs_struct.CSIRSPeriod{resIdx}(2);
        else
            if strcmpi(csirs_struct.CSIRSPeriod{resIdx},'on')
                Tcsi_rs = 1;
            else
                Tcsi_rs = 0;
            end
            Toffset = 0;
        end

        % Check for the presence of CSI-RS, based on slot periodicity and offset
        if (Tcsi_rs ~= 0) && (mod(SlotsPerFrame*NFrameR + NSlotR - Toffset, Tcsi_rs) == 0)
            csirsTransmitted(resIdx) = 1;
        end
    end
end

function freqRange = validateFc(fc)
%   validateFc validates the carrier frequency FC and returns the frequency
%   range as either 'FR1' or 'FR2'.

    if fc >= 410e6 && fc <= 7.125e9
        freqRange = 'FR1';
    elseif fc >= 24.25e9 && fc <= 52.6e9
        freqRange = 'FR2';
    else
        error('nr5g:invalidFreq',['Selected carrier frequency is outside '...
            'FR1 (410 MHz to 7.125 GHz) and FR2 (24.25 GHz to 52.6 GHz).']);
    end
end
```

```
function beamDir = getInitialBeamDir(scatAng,azBeamWidth,elBeamWidth)
%   getInitialBeamDir returns the initial beam direction BEAMDIR, given the
%   angle of scatterer position with respect to transmit or receive antenna
%   array SCATANG, beamwidth of transmit or receive beam in azimuth plane
%   AZBEAMWIDTH, and beamwidth of transmit or receive beam in elevation
%   plane ELBEAMWIDTH.

    % Azimuth angle boundaries of all transmit/receive beams
    azSSBSweep = -180:azBeamWidth:180;
    % Elevation angle boundaries of all transmit/receive beams
    elSSBSweep = -90:elBeamWidth:90;

    % Get the azimuth angle of transmit/receive beam
    azIdx1 = find(azSSBSweep <= scatAng(1),1,'last');
    azIdx2 = find(azSSBSweep >= scatAng(1),1,'first');
    azAng = (azSSBSweep(azIdx1) + azSSBSweep(azIdx2))/2;

    % Get the elevation angle of transmit/receive beam
    elIdx1 = find(elSSBSweep <= scatAng(2),1,'last');
    elIdx2 = find(elSSBSweep >= scatAng(2),1,'first');
    elAng = (elSSBSweep(elIdx1) + elSSBSweep(elIdx2))/2;

    % Form the azimuth and elevation angle pair (in the form of [az;el])
    % for transmit/receive beam
    beamDir = [azAng;elAng];
end
```

See Also

Objects
`nrCSIRSConfig` | `nrCarrierConfig`

Related Examples

- “NR Cell Search and MIB and SIB1 Recovery” on page 1-30
- “NR SSB Beam Sweeping” on page 1-98
- “SNR Definition Used in Link Simulations” on page 5-86

NR Positioning Reference Signal

This example shows how to configure the time-frequency aspects of the new radio (NR) positioning reference signal (PRS). The example shows how different PRS resource set configurations affect the time-frequency structure of the PRS using 5G Toolbox™ features.

Introduction

According to TR 22.872 [1 on page 1-139], these 5G use cases require finding the accurate and real-time location of nodes in the wireless network:

- Location-based services, such as accurate positioning for shared bikes and location-based advertising
- Industry-related use cases, such as waste management and collection
- eHealth-related use cases, such as medical equipment location in hospitals and patient location outside of hospitals
- Emergency-related and mission-critical-related use cases
- Road-related use cases, such as road-user charging
- Rail- and maritime-related use cases, such as asset and freight tracking
- Aerial-related use cases, such as unmanned aerial vehicle missions and operations

To find the position of a node, downlink reference signals are used at the user equipment (UE) side. The existing downlink reference signals, like the channel state information reference signal and the synchronization signals, are not used for the position estimation due to these limitations:

- These reference signals are not capable of detecting a sufficient number of neighbor access network nodes (gNBs) because of the interference from the adjacent cells when the signals from multiple cells collide in both the time-domain and frequency-domain. Due to this interference, signals from nearby cells shadow the weak signals from far away cells, causing difficulty for the UE to detect far away cells or gNBs. This difficulty results in a loss in hearability.
- These reference signals also have weak correlation properties due to low resource element (RE) density and their RE pattern might not spread across all of the subcarriers in the frequency-domain.

To overcome these limitations, the 3GPP introduced a new reference signal called the PRS in Release 16 of 5G specification, with a high RE density and with the correlation properties better than that of existing reference signals due to the diagonal or staggered PRS RE pattern. Hearability of the PRS is achieved with a concept called muting. With PRS muting, multiple cells transmit the PRS in a coordinated manner by muting the relevant PRS transmission occasions to avoid the interference from adjacent cells. This example demonstrates the time-frequency aspects of the PRS and shows how to configure PRS muting.

As per the 3GPP standard, you can configure a UE with one or more downlink PRS positioning frequency layer configurations. A PRS positioning frequency layer is defined as a collection of PRS resource sets with each PRS resource set defining a collection of PRS resources. All of the PRS resource sets defined in the PRS positioning frequency layer are configured with these common parameters:

- **Subcarrier Spacing:** Subcarrier spacing for all PRS resource sets in a PRS positioning frequency layer, specified as 15, 30, 60, or 120. Use the `SubcarrierSpacing` property of the `nrCarrierConfig` to set the subcarrier spacing of a PRS resource set.

- **Cyclic Prefix:** Cyclic prefix for all PRS resource sets in a PRS positioning frequency layer, specified as `'normal'` or `'extended'`. Use the `CyclicPrefix` property of the `nrCarrierConfig` object to set the cyclic prefix of a PRS resource set.
- **PRS Point A:** Absolute frequency of reference resource block or common resource block. The lowest subcarrier of this reference resource block is known as PRS Point A. The 5G specification defines the PRS frequency resource allocation relative to PRS point A. The example shows how to configure the start of PRS frequency-domain allocation using 5G Toolbox™ features and its relation with the standard notion.

The 5G Toolbox™ offers the PRS symbols and indices generation with a configuration object `nrPRSConfig` and functions `nrPRS` and `nrPRSIndices`. The `nrPRSConfig` object bundles all of the parameters related to a PRS resource set.

PRS Slot Configuration

These properties of the `nrPRSConfig` object control the PRS slot configuration.

- `PRSResourceSetPeriod`: Slot periodicity and offset (0-based) of a PRS resource set
- `PRSResourceOffset`: Slot offset (0-based) of each PRS resource defined relative to the slot offset of the PRS resource set
- `PRSResourceRepetition`: Repetition factor of all PRS resources in a PRS resource set
- `PRSResourceTimeGap`: Slot offset between two consecutive repetition indices of all PRS resources in a PRS resource set

The Figure 1 illustrates the case of a PRS resource set with 2 PRS resources. The PRS resource set period is 10 slots and the PRS resource set offset is 3 slots (0-based). The first PRS resource offset is 1 slot (0-based) and the second PRS resource offset is 4 slots (0-based). The PRS resource repetition factor is 2 (which means each PRS resource is repeated twice in all PRS resource set instances), and the PRS resource time gap is 2 (which means an offset of 2 slots).

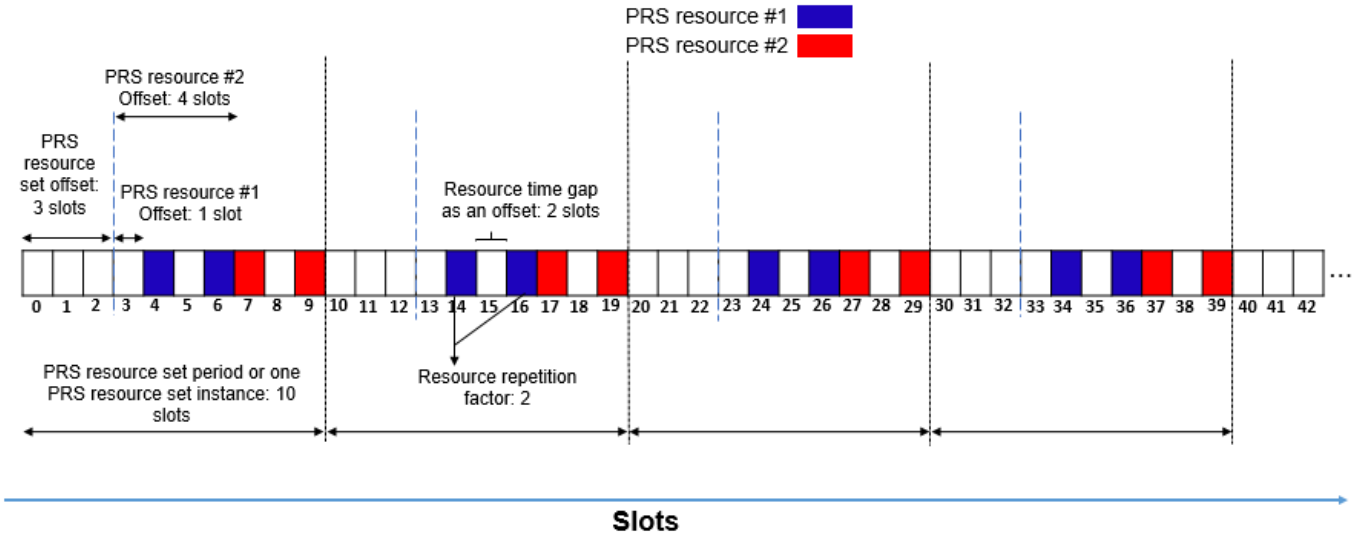
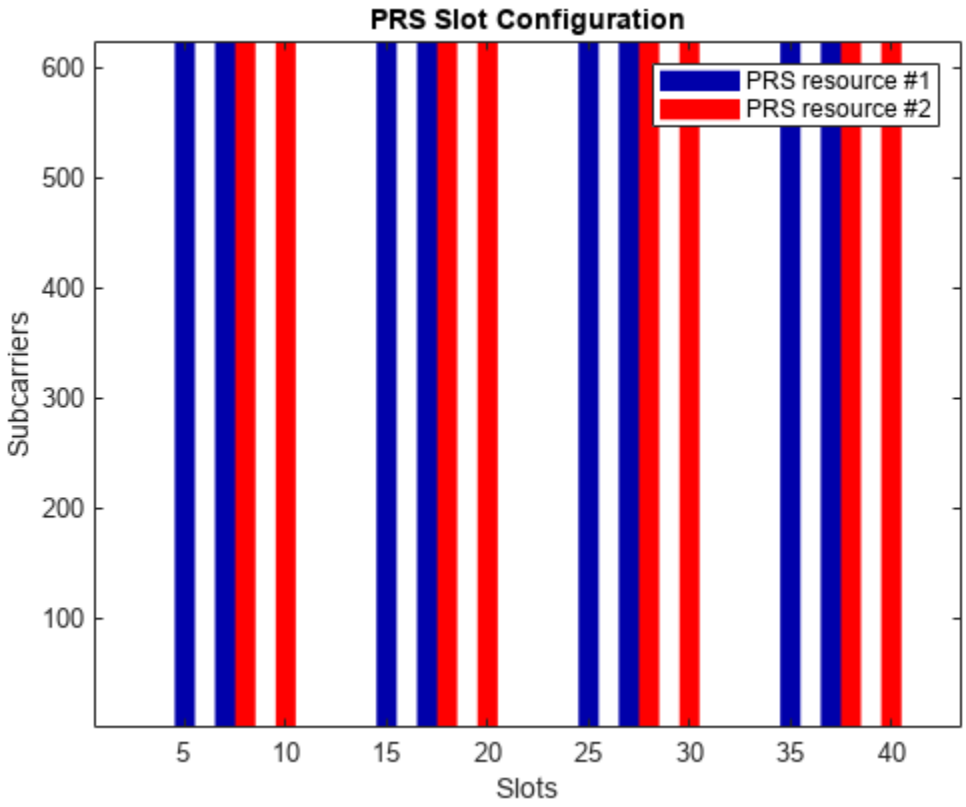


Figure 1: PRS Slot Configuration

Configure the PRS slot configuration parameters and plot the carrier grid in a slot level to highlight the slots in which PRS resources are present.

```
% Set carrier parameters
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'Normal';

% Set parameters related to PRS slot configuration
prs = nrPRSConfig;
prs.PRSResourceSetPeriod = [10 3];
prs.PRSResourceOffset = [1 4];
prs.PRSResourceRepetition = 2;
prs.PRSResourceTimeGap = 2;
numSlots = 43;                                    % Consider 43 slots to compare the plot with
plotTitle = 'PRS Slot Configuration';
plotGrid(carrier,prs,numSlots,'SlotFill',plotTitle); % Slot numbers on the x-axis of the MATLAB
```



PRS Muting Configuration

You can mute a PRS resource in two ways:

- Mute the PRS resource set instances using the properties `MutingPattern1` and `MutingBitRepetition` of the `nrPRSConfig` object
- Mute the PRS resource repetition indices using the property `MutingPattern2` of the `nrPRSConfig` object

`MutingPattern1`: A binary vector that controls the muting of PRS resource set instances. Each element in the binary vector controls the muting of all PRS resources in a PRS resource set instance (one instance corresponds to one period of the PRS resource set). The first element in the binary

vector corresponds to the first PRS resource set instance, the second element corresponds to the second PRS resource set instance, and so on.

- A binary value of 1 indicates that all PRS resources in a PRS resource set instance are transmitted.
- A binary value of 0 indicates that all PRS resources in a PRS resource set instance are muted.

The Figure 2 illustrates the case of muting bit pattern option-1 as [1 0] in addition to the previous PRS slot configuration which is shown in Figure 1. In this case, the effective muting bit pattern option-1 at PRS resource set instance level is [1 0 1 0 1 0 ...], which is the repeated pattern of `MutingPattern1`.

Solid fill in the figure represents the PRS resource set instances that are transmitted, and pattern fill in the figure represents the PRS resource set instances that are muted or not transmitted.

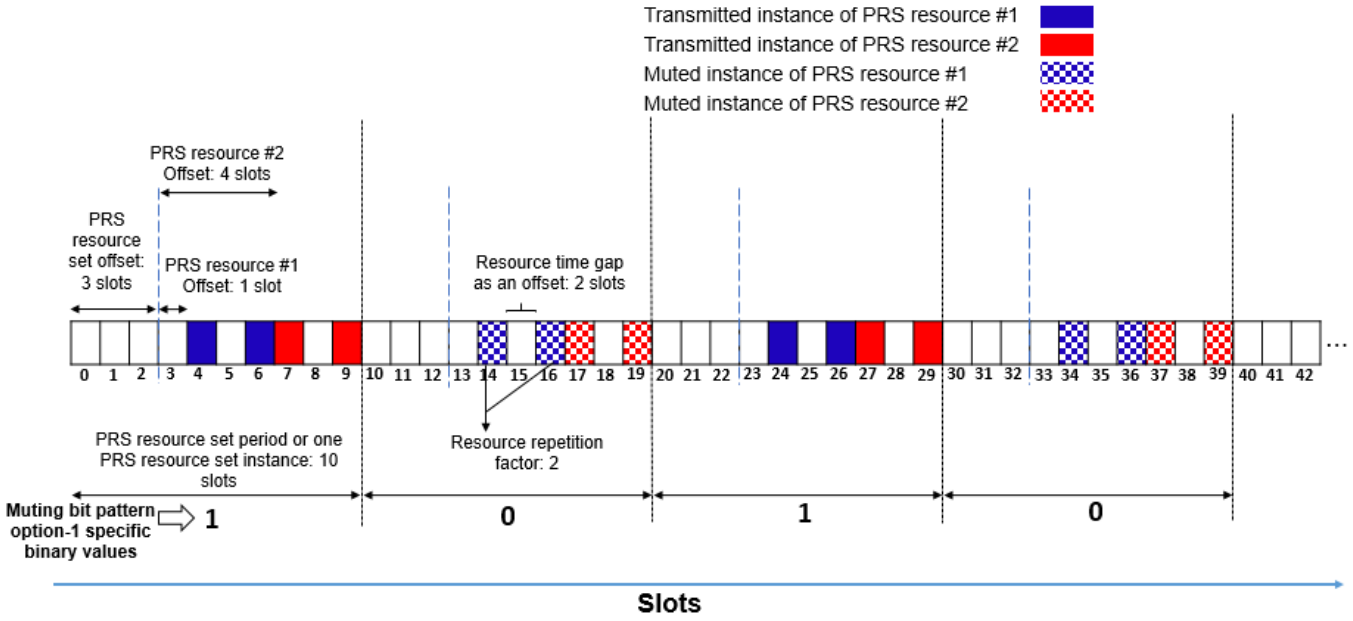


Figure 2: Muting Bit Pattern Option-1 Configuration

`MutingBitRepetition`: Number of consecutive PRS resource set instances (say *N*) corresponding to a single element of the `MutingPattern1` binary vector. The first element in the `MutingPattern1` binary vector corresponds to *N* consecutive instances of a PRS resource set, the second element corresponds to the next *N* consecutive instances of a PRS resource set, and so on.

The Figure 3 illustrates the case of muting pattern option-1 as [1 0] and muting bit repetition factor as 2 in addition to the previous PRS slot configuration which is shown in Figure 1. With these parameters, the effective muting bit pattern at PRS resource set instance level is [1 1 0 0 1 1 0 0 ...].

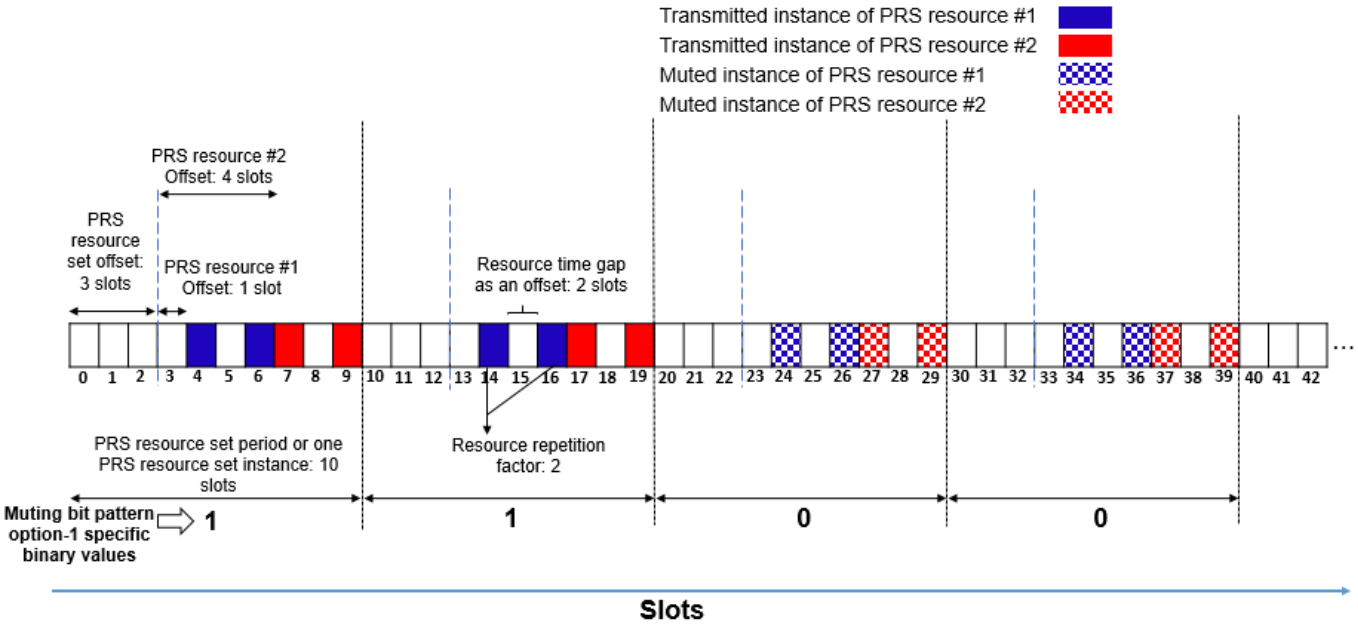


Figure 3: Muting Bit Pattern Option-1 with Muting Bit Repetition Factor Configuration

`MutingPattern2`: A binary vector that controls the muting of PRS resource repetition indices in all active PRS resource set instances. The first element in the binary vector corresponds to the first repetition index of a PRS resource, the second element corresponds to the second repetition index of a PRS resource, and so on. The length of the binary vector is equal to the value of the `PRSResourceRepetition` property and the same binary vector is applicable to all PRS resources in a PRS resource set.

The Figure 4 illustrates the case of muting bit pattern option-2 as [0 1] in addition to the previous PRS slot configuration which is shown in Figure 1. In this case, the effective muting bit pattern option-2 at PRS resource repetition index level is [0 1 0 1 0 1 ...], which is the repeated pattern of `MutingPattern2`.

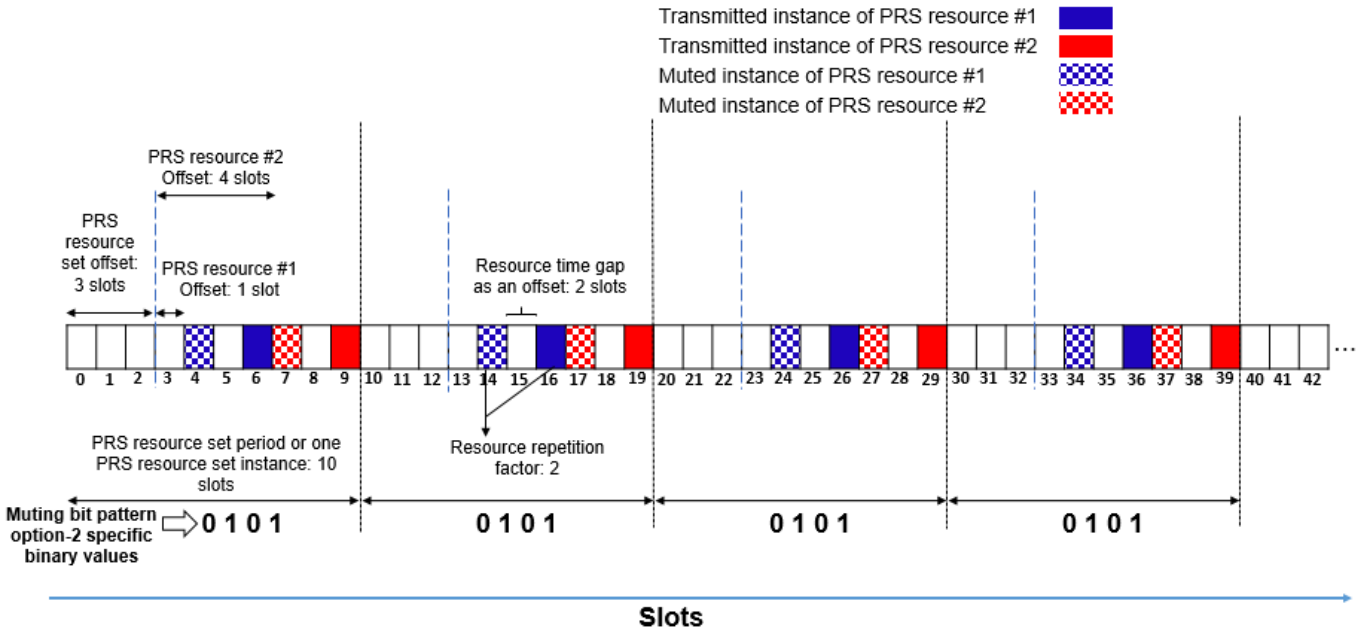


Figure 4: Muting Bit Pattern Option-2 Configuration

When you configure both the `MutingPattern1` and `MutingPattern2` properties, the effective muting bit pattern is equal to the bit-wise AND of muting bit pattern option-1 and muting bit pattern option-2.

The Figure 5 illustrates the case of muting bit pattern option-1 as [1 0], muting bit repetition factor as 2, and muting bit pattern option-2 as [0 1] in addition to the previous PRS slot configuration which is shown in Figure 1. With these parameters, the effective muting bit pattern option-1 at PRS resource set instance level is [1 1 0 0 1 1 ...] and the effective muting bit pattern option-2 at PRS resource repetition index level is [0 1 0 1 0 1 ...].

For the specified configuration, as the number of PRS resources is 2 and the PRS resource repetition factor is 2, one PRS resource set instance contains four resource instances. And below are the muting bit pattern option-1 and the muting bit pattern option-2 at PRS resource repetition index level:

Muting bit pattern option-1 at PRS resource repetition index level, *binaryVec1* = [1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 ...].

Muting bit pattern option-2 at PRS resource repetition index level, *binaryVec2* = [0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ...].

Effective muting bit pattern at PRS resource repetition index level is the bit-wise AND of *binaryVec1* and *binaryVec2*, which is equal to [0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 ...]. You can observe this pattern in the Figure 5.

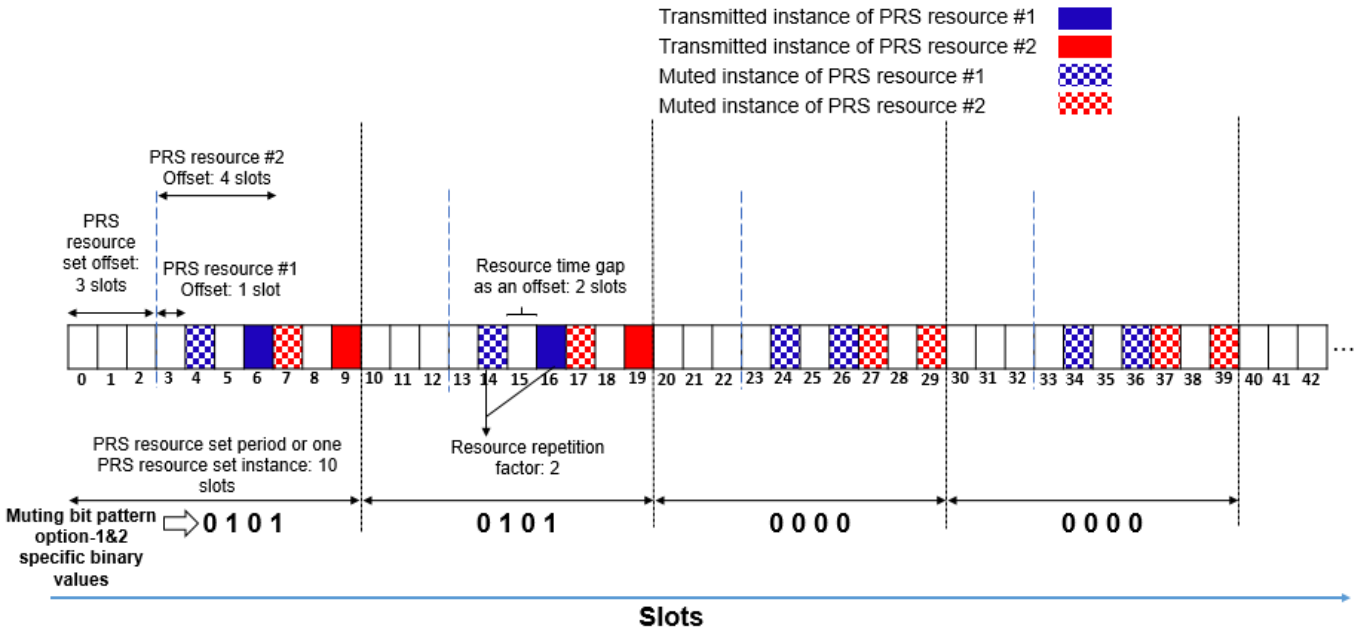
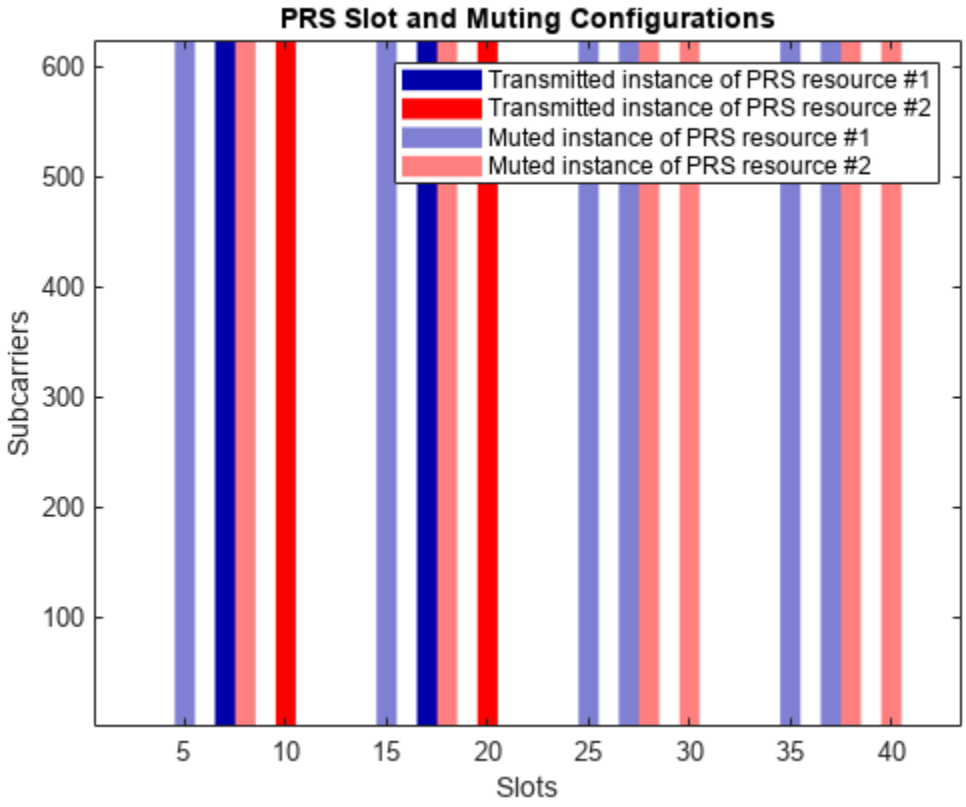


Figure 5: Muting Bit Pattern Option-1 and Option-2 Configuration

Configure the PRS muting patterns in addition to the previous PRS slot configuration in `prs`. MATLAB code generates the PRS resources as empty for the slots that are muted. This example highlights the muted slots with light shade in the generated plots for the easy comparison with the Figures 2 to 5.

```
prs.MutingPattern1 = [1 0];                          % Use [] to disable the muting bit pattern c
prs.MutingBitRepetition = 2;
prs.MutingPattern2 = [0 1];                          % Use [] to disable the muting bit pattern c
plotTitle = 'PRS Slot and Muting Configurations';
plotGrid(carrier,prs,numSlots,'SlotFill',plotTitle); % Slot numbers on the x-axis of the MATLAB p
```



PRS Time-Domain and Frequency-Domain Allocation

These properties of the `nrPRSConfig` object control the time-domain allocation of a PRS resource.

- `NumPRSSymbols` (L_{PRS}): Number of consecutive OFDM symbols in a slot that are allocated for each PRS resource in a PRS resource set.
- `SymbolStart` ($l_{\mathrm{start}}^{\mathrm{PRS}}$): Starting OFDM symbol (0-based) of each PRS resource in a PRS resource set. This property is defined relative to the first OFDM symbol of the slot (symbol #0).

The OFDM symbols allocated for a PRS resource are defined as
$l = l_{\mathrm{start}}^{\mathrm{PRS}}, l_{\mathrm{start}}^{\mathrm{PRS}} + 1, \ldots, l_{\mathrm{start}}^{\mathrm{PRS}} + L_{\mathrm{PRS}} - 1$.

The Figure 6 illustrates the case, where the number of OFDM symbols allocated for a PRS resource is 6 and the starting OFDM symbol of PRS resource allocation is 3 (0-based).

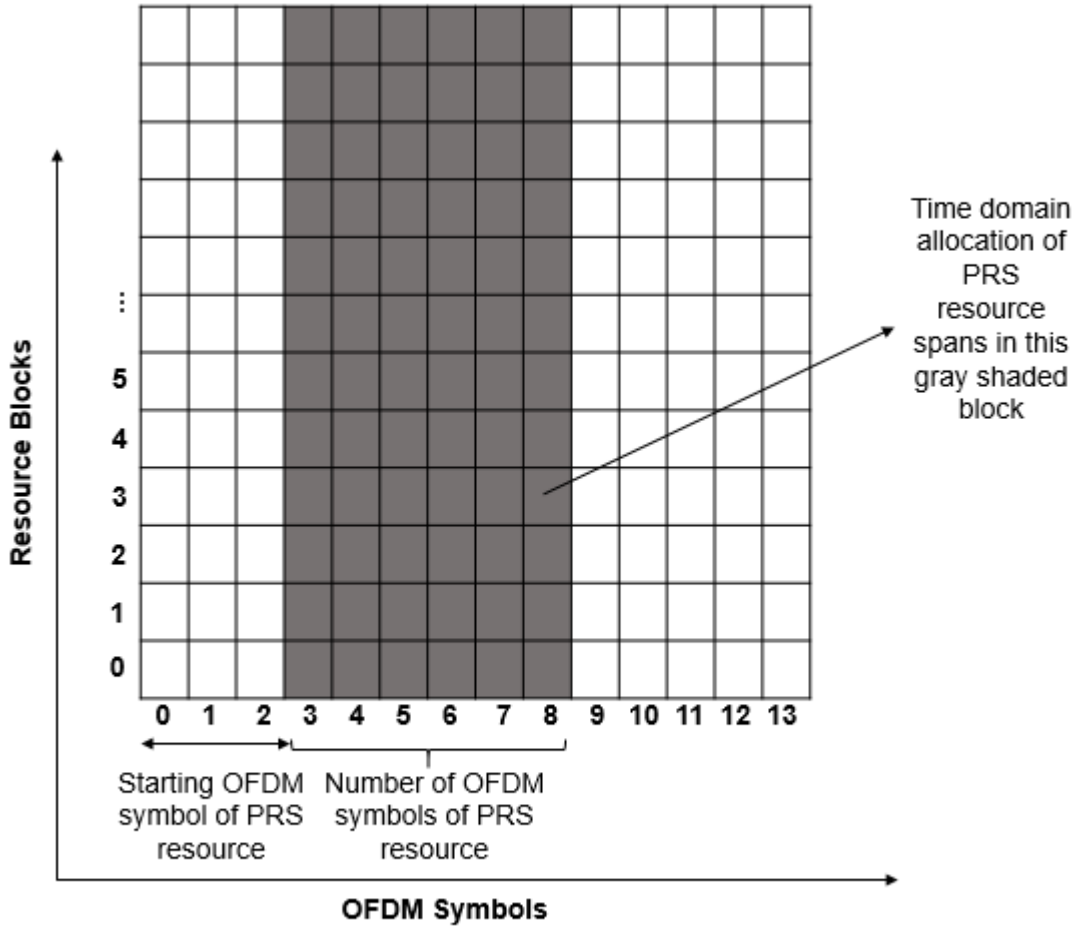


Figure 6: PRS OFDM Symbols Allocation

These properties of the `nrPRSConfig` object control the frequency-domain allocation of a PRS resource at RB level granularity.

- `NumRB`: Number of physical resource blocks (PRBs) allocated for all PRS resources in a PRS resource set.
- `RBOffset`: Starting PRB index (0-based) of all PRS resources in a PRS resource set. This property is defined relative to the carrier resource grid, but the specification defines this offset relative to common resource block 0 (CRB 0) (the Figure 7 highlights these notations).

The Figure 7 illustrates the frequency-domain allocation of a PRS resource and the carrier.

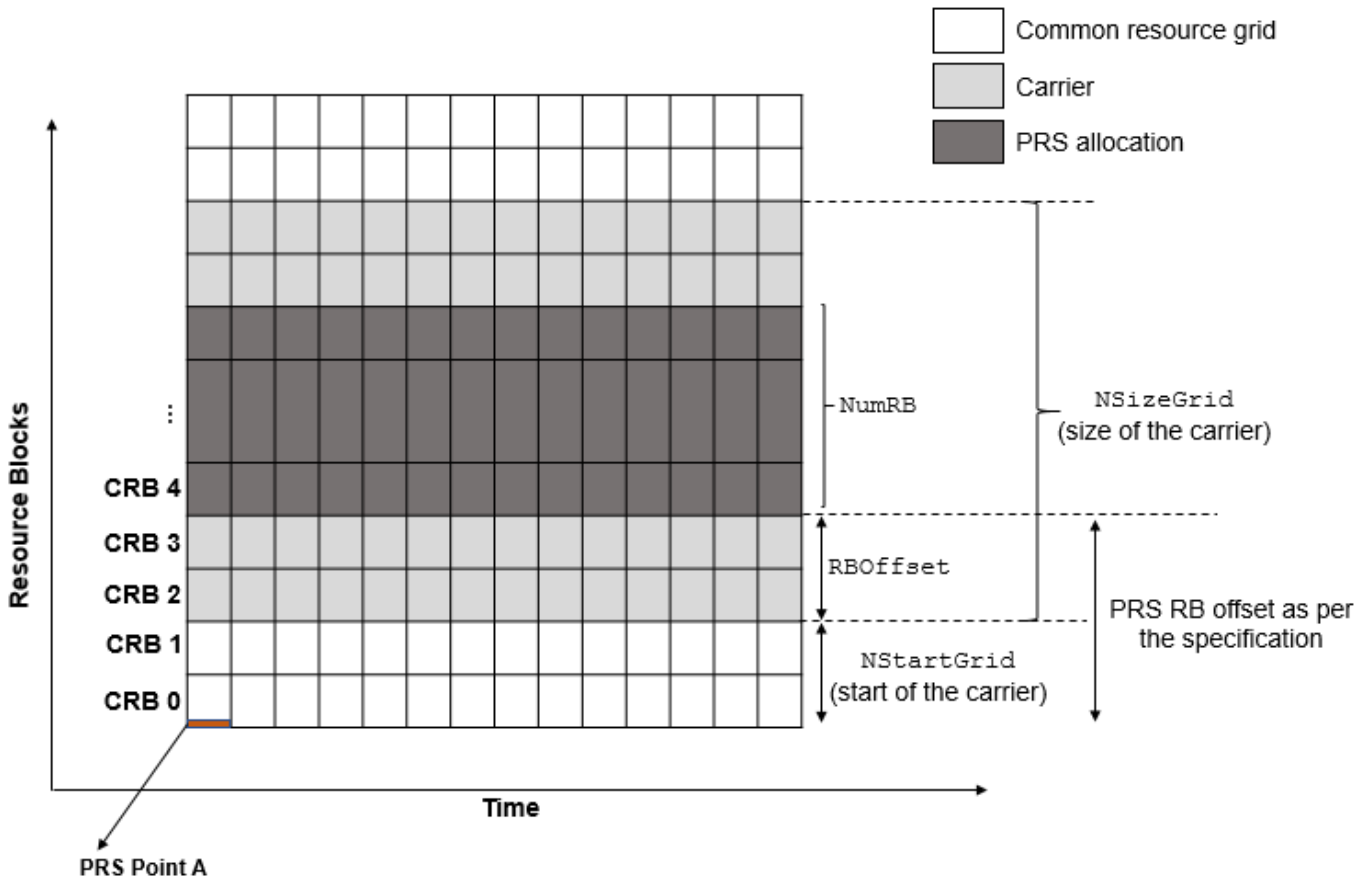


Figure 7: PRS Resource Blocks Allocation

These properties of the `nrPRSConfig` object control the frequency-domain allocation of a PRS resource at RE level granularity.

- `CombSize` ($K_{\text{comb}}^{\text{PRS}}$): RE density of all PRS resources in a PRS resource set. For example, if you configure the value as $i \in \{2, 4, 6, 12\}$, every ith RE in the PRB is allocated for the PRS.
- `REOffset` ($k_{\text{offset}}^{\text{PRS}}$): Starting RE offset (0-based) in the first OFDM symbol of each PRS resource in a PRS resource set. The relative RE offsets of the next or the following PRS OFDM symbols are defined relative to this `REOffset` value, as shown in the table in the Figure 8.

K_{comb}^{PRS}	Symbol Numbers within the Downlink PRS Resource Allocation: $(l - l_{start}^{PRS}) = 0, \ldots, L_{PRS} - 1$											
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
2	0	1	0	1	0	1	0	1	0	1	0	1
4	0	2	1	3	0	2	1	3	0	2	1	3
6	0	3	1	4	2	5	0	3	1	4	2	5
12	0	6	3	9	1	7	4	10	2	8	5	11

Figure 8: PRS Resource Element Offsets

Consider the number of OFDM symbols allocated for a PRS resource as 6, the starting OFDM symbol of PRS resource allocation as 3 (0-based), the PRS comb size as 4, and the RE offset in the first OFDM symbol as 2 (0-based). For PRS comb size 4, the relative RE offsets of the following PRS OFDM symbols are calculated based on row 2 from the table in the Figure 8.

For the specified configuration, the OFDM symbol numbers within the PRS resource allocation are $l - l_{\text{start}}^{\text{PRS}} = 0, \ldots, 5$ (0-based).

From the table in the Figure 8, the relative RE offsets (k') in the PRS OFDM symbols are [0 2 1 3 0 2]. The arrows in the Figure 9 highlight these relative offsets. The figure shows a single PRB in one slot to highlight the RE level pattern.

Effective RE offsets (the second term of the k calculation, as defined in TS 38.211 Section 7.4.1.7.3 [2 on page 1-139]) in the PRS OFDM symbols are $\text{mod}\left(k_{\text{offset}}^{\text{PRS}} + k', K_{\text{comb}}^{\text{PRS}}\right) = \text{mod}([2\ 4\ 3\ 5\ 2\ 4], 4) = [2\ 0\ 3\ 1\ 2\ 0]$ (0-based).

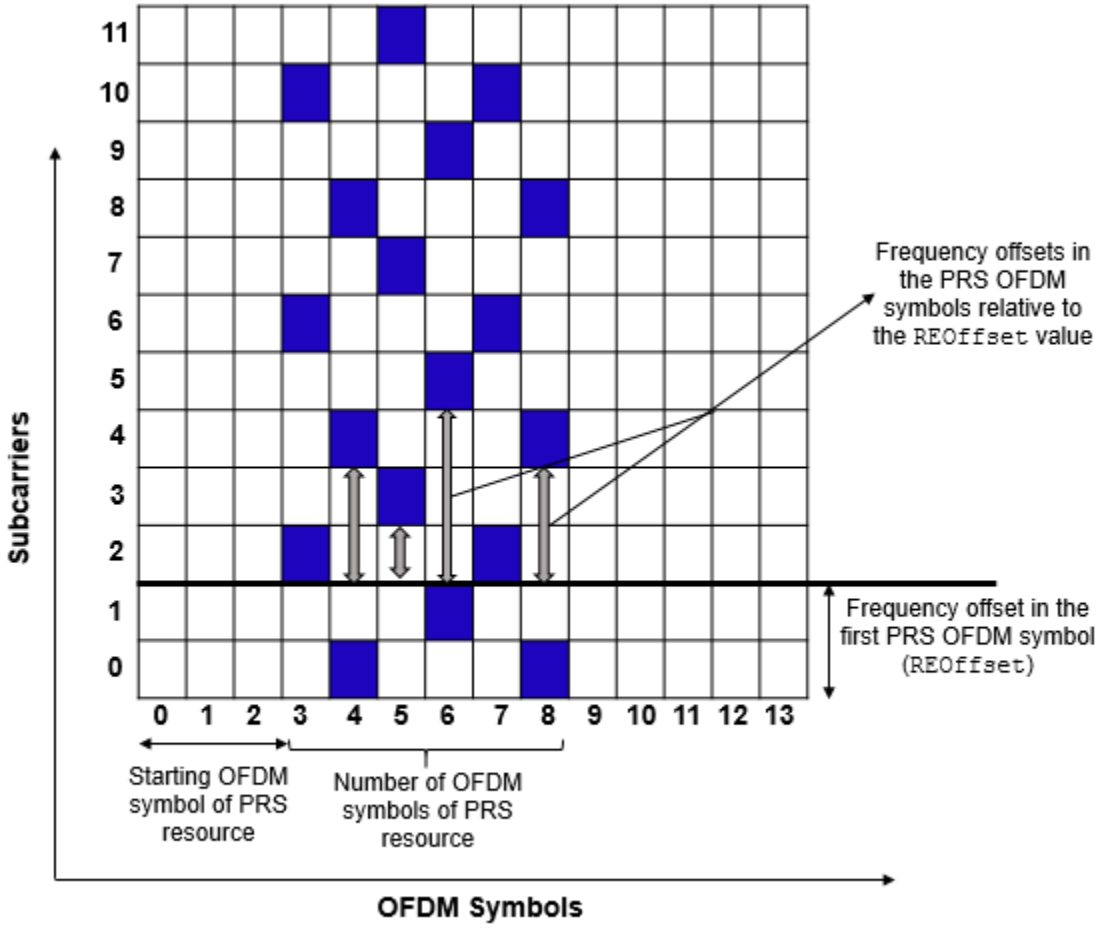
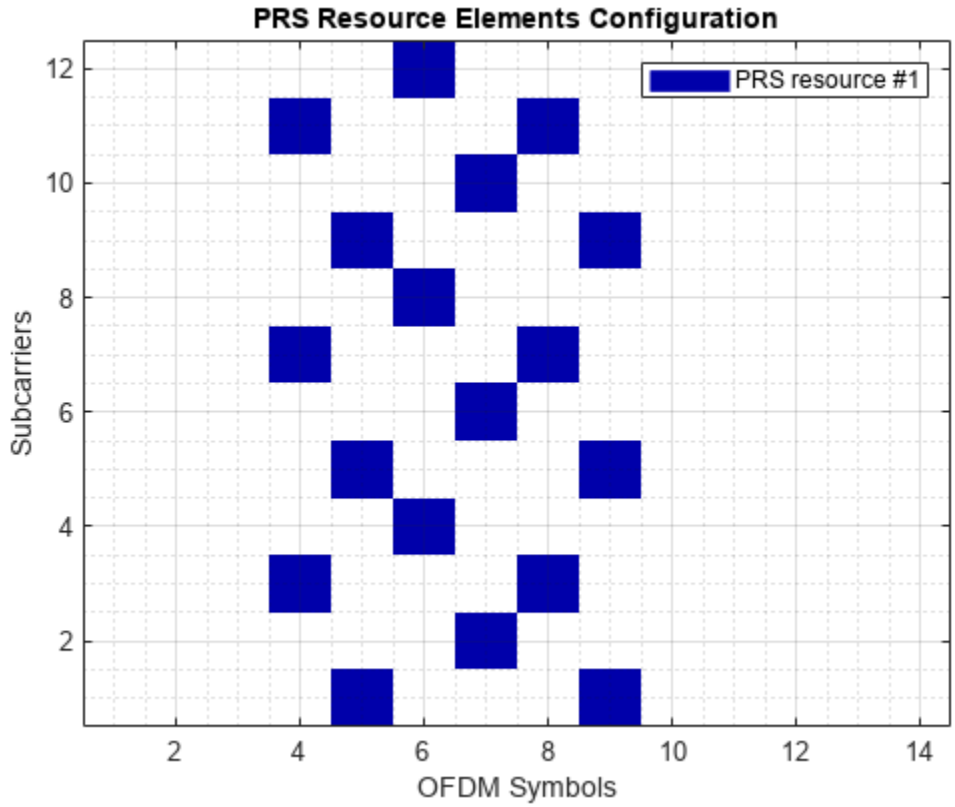


Figure 9: PRS Resource Elements Allocation

Configure the RE and symbol level allocation of the PRS. To observe the frequency-domain pattern at the RE level, this code uses a single PRB in one slot.

```
carrier.NSizeGrid = 1;
prs = nrPRSConfig;
prs.NumPRSSymbols = 6;
prs.SymbolStart = 3;
prs.NumRB = 1;
prs.RBOffset = 0;
prs.CombSize = 4;
prs.REOffset = 2;
numSlots = 1;                                    % Consider one slot to highlight the RE patte
plotTitle = 'PRS Resource Elements Configuration';
plotGrid(carrier,prs,numSlots,'REFill',plotTitle); % OFDM symbol and subcarrier numbers in the MA
grid on;
grid minor;
```



PRS Sequence Identity

The `NPRSID` property of the `nrPRSConfig` object is used in the initialization of a pseudo-random binary sequence to generate the PRS symbols, as defined in TS 38.211 Section 7.4.1.7.2 [2 on page 1-139].

Summary and Further Exploration

The example shows how to configure the time-frequency aspects of the PRS and how different PRS resource set configurations affect the time-frequency structure of the PRS.

Configure multiple PRS resources with different time and frequency allocation aspects and slot configuration aspects, and then observe the variations in the slot allocation and RE positions. Try

different muting configurations (different combinations of muting bit pattern option-1 and option-2), and then see the variations in the grid.

References

1 3GPP TR 22.872. "Study on positioning use cases." *3rd Generation Partnership Project; Technical Specification Group Services And System Aspects*.

2 3GPP TS 38.211. "NR; Physical channels and modulation (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local functions

This example uses these local functions to plot the time-frequency aspects of all PRS resources configured in a PRS resource set.

```
function plotGrid(carrier,prs,numSlots,flag,imTitle)
%   plotGrid(CARRIER,PRS,NUMSLOTS,FLAG,IMTITLE) plots the carrier grid by
%   considering these inputs:
%
%   CARRIER  - Carrier specific configuration object
%   PRS      - Positioning reference signal configuration object
%   NUMSLOTS - Number of slots over which the function plots the
%              carrier grid
%   FLAG     - Flag to consider the plot style, specified as
%              'SlotFill' or 'REFill'. The 'SlotFill' option highlights the
%              slots in which the PRS is present. The 'REFill' option
%              highlights the REs in which the PRS is present.
%   IMTITLE  - Title for the plot

    [transmittedSlots,mutedSlots] = getTransmittedAndMutedSlots(carrier,prs,numSlots); % Of size
    numRes = size(transmittedSlots,2);
    rng(37);                  % Set RNG state for repeatability
    tempColors = reshape(rand((numRes-2)*3,1),[],3);
    map = [1 1 1;...          % White color
           0 0 0.6667;...     % Blue color
           1 0 0;...          % Red color
           tempColors];
    resScalings = 1:numRes+1; % 1 for white color, 2 for blue color, and 3 for red color, and so
    prsREGrid = zeros(carrier.NSizeGrid*12,carrier.SymbolsPerSlot*numSlots);
    prsSlotGrid = resScalings(1)*ones(carrier.NSizeGrid*12,numSlots);                  % For whi

    figure();
    % Plot PRS slot grid
    image(abs(prsSlotGrid));

    % Apply colormap to image
    colormap(map);

    for slotIdx = 0:numSlots-1
        if strcmpi(flag,'REFill')
            carrier.NSlot = slotIdx;
            ind = nrPRSIndices(carrier,prs,'OutputResourceFormat','cell');
            sym = nrPRS(carrier,prs,'OutputResourceFormat','cell');
            slotGrid = nrResourceGrid(carrier);
            for resIdx = 1:numel(ind)
```

```
                slotGrid(ind{resIdx}) = resScalings(resIdx+1)*abs(sym{resIdx});
            end
            prsREGrid(:,(1:carrier.SymbolsPerSlot)+carrier.SymbolsPerSlot*slotIdx) = slotGrid;

            % Replace all zero values of carrier grid with proper scaling
            % for white background
            prsREGrid(prsREGrid == 0) = resScalings(1);

            % Plot carrier grid
            image(abs(prsREGrid));
            axis xy;

            % Apply colormap to image
            colormap(map);

            % Add labels to x-axis and y-axis
            xlabel('OFDM Symbols');
            ylabel('Subcarriers');

            % Generate lines
            L = line(ones(numRes),ones(numRes),'LineWidth',8);
            % Index color map and associate selected colors with lines
            set(L,{'color'},mat2cell(map(resScalings(2:end),:),ones(1,numRes),3));
            % Create legend
            legendNames = cell(1,numRes);
            for resIdx = 1:numRes
                legendNames{resIdx} = ['PRS resource #' num2str(resIdx)];
            end
            legend(legendNames{:});
        else % 'SlotFill'
            axis xy;
            hold on;
            for resIdx = 1:numRes
                ismuted = mutedSlots(slotIdx+1,resIdx);
                isTransmitted = transmittedSlots(slotIdx+1,resIdx);
                temp = [slotIdx+1 slotIdx+1];
                color = map(resIdx+1,:);
                if isTransmitted
                    hT(resIdx) = patch([temp(1)-0.5 temp(1)-0.5 temp(end)+0.5 temp(end)+0.5],[1 0
                        color,'LineStyle','none'); %#ok<AGROW>
                end
                if ismuted
                    hM(resIdx) = patch([temp(1)-0.5 temp(1)-0.5 temp(end)+0.5 temp(end)+0.5],[1 0
                        color,'LineStyle','none','FaceAlpha',0.5); %#ok<AGROW>
                end
            end
        end
    end

    % Add title to image
    title(imTitle);
    if strcmpi(flag,'SlotFill')
        % Add labels to x-axis and y-axis
        xlabel('Slots');
        ylabel('Subcarriers');

        % Create legend
        legendNames = cell(1,numRes);
```

```
            legendNamesTxRes = cell(1,numRes);
            legendNamesMutedRes = cell(1,numRes);
            for resIdx = 1:numRes
                legendNames{resIdx} = ['PRS resource #' num2str(resIdx)];
                legendNamesTxRes{resIdx} = ['Transmitted instance of PRS resource #' num2str(resIdx)];
                legendNamesMutedRes{resIdx} = ['Muted instance of PRS resource #' num2str(resIdx)];
            end
            if sum(mutedSlots(:)) > 0
                legend([hT hM],[legendNamesTxRes legendNamesMutedRes]);
            else
                legend(hT,legendNames);
            end
        end
    end

    function [transmittedSlots,mutedSlots] = getTransmittedAndMutedSlots(carrier,prs,numSlots)
    %   [TRANSMITTEDSLOTS,MUTEDSLOTS] = getTransmittedAndMutedSlots(CARRIER,PRS,NUMSLOTS)
    %   returns logical arrays to give information about transmitted slots
    %   TRANSMITTEDSLOTS and muted slots MUTEDSLOTS for all PRS resources by
    %   considering these inputs:
    %
    %   CARRIER  - Carrier specific configuration object
    %   PRS      - Positioning reference signal configuration object
    %   NUMSLOTS - Number of slots for which the output is returned

        % Take copy of input PRS configuration to retain input muting
        % configuration unchanged for further processing
        prs1 = prs;

        % Extract PRS configuration without considering muting aspects
        prs1.MutingPattern1 = [];
        prs1.MutingBitRepetition = 2;
        prs1.MutingPattern2 = [];
        numResOffVal = numel(prs1.PRSResourceOffset);
        numSymStartVal = numel(prs1.SymbolStart);
        numPRSSymVal = numel(prs1.NumPRSSymbols);
        numREOffVal = numel(prs1.REOffset);
        numNPRSIDVal = numel(prs1.NPRSID);

        % Calculate the number of PRS resources configured in a PRS
        % resource set
        numRes = max([numResOffVal, numSymStartVal, numPRSSymVal, ...
            numREOffVal, numNPRSIDVal]);
        PRSPresenceWithOutMuting = zeros(numSlots,numRes);
        for slotIdx = 0:numSlots-1
            carrier.NSlot = slotIdx;
            [~,PRSPresence] = nr5g.internal.validateAndSchedulePRS(carrier,prs1);
            PRSPresenceWithOutMuting(slotIdx+1,:) = PRSPresence;
        end

        % Consider PRS configuration with muting aspects
        prs1.MutingPattern1 = prs.MutingPattern1;
        prs1.MutingBitRepetition = prs.MutingBitRepetition;
        prs1.MutingPattern2 = prs.MutingPattern2;
        PRSPresenceWithMuting = zeros(numSlots,numRes);
        for slotIdx = 0:numSlots-1
            carrier.NSlot = slotIdx;
            [~,PRSPresence] = nr5g.internal.validateAndSchedulePRS(carrier,prs1);
```

```
            PRSPresenceWithMuting(slotIdx+1,:) = PRSPresence;
        end

        % Identify transmitted and muted slots based on PRS scheduling
        transmittedSlots = PRSPresenceWithMuting;                                   % Of size numSlots-
        mutedSlots = PRSPresenceWithOutMuting ~= PRSPresenceWithMuting;             % Of size numSlots-
    end
```

See Also

Objects
nrPRSConfig | nrCarrierConfig

Functions
nrPRS | nrPRSIndices

NR Downlink Control Information Formats

This example introduces the NR downlink control information (DCI) formats and their definitions, and shows how to use MATLAB® classes to represent DCI formats and encode and decode DCI information bit payloads.

Introduction

NR and LTE use downlink control information (DCI) to send dynamic physical layer control messages from the network to each UE. This information can be system-wide or user-equipment-specific (UE-specific), and contains aspects of uplink and downlink data scheduling, HARQ management, power control, and other signalling. The sidelink uses sidelink control information (SCI) to carry PHY control messages between UEs via a similar mechanism.

NR defines a number of different DCI formats, each serving a different usage, for example, scheduling of PUSCH or PDSCH. Each format specifies an ordered set of bit fields, where each field conveys distinct transmission information, such the frequency resource assignment, time resource assignment, redundancy version, and modulation and coding. The number of bits associated with a field may be fixed, or be dependent on other protocol state, for example, the active BWP size. All the fields map, in order of the format definition, onto a set of information bits, which are then encoded and carried on the physical downlink control channel (PDCCH). The mapping is such that the most significant bit of each field is mapped to the lowest-order information bit for that field. For NR DCI, both padding of zero bits and truncation may be applied to align the payload sizes according to different DCI formats. This size alignment simplifies the blind decoding process and reduces the number of unique payload sizes that have to be searched for.

The fields defined in a format may also depend on the type of RNTI associated with the control information, for example, system information, paging, power control, and user scheduling. This RNTI value scrambles the CRC attached to the information bit payload sent on the PDCCH.

The DCI formats supported by NR Release 16 are:

DCI Format	Usage
0_0	Scheduling of PUSCH in one cell
0_1	Scheduling of one or multiple PUSCH in one cell, or indicating downlink feedback information f
0_2	Scheduling of PUSCH in one cell
1_0	Scheduling of PDSCH in one cell
1_1	Scheduling of PDSCH in one cell, and/or triggering one shot HARQ-ACK codebook feedback
1_2	Scheduling of PDSCH in one cell
2_0	Notifying a group of UEs of the slot format, available RB sets, COT duration and search space s
2_1	Notifying a group of UEs of the PRB(s) and OFDM symbol(s) where UE may assume no transmi
2_2	Transmission of TPC commands for PUCCH and PUSCH
2_3	Transmission of a group of TPC commands for SRS transmissions by one or more UEs
2_4	Notifying a group of UEs of the PRB(s) and OFDM symbol(s) where UE cancels the correspondi
2_5	Notifying the availability of soft resources as defined in Clause 9.3.1 of TS 38.473
2_6	Notifying the power saving information outside DRX Active Time for one or more UEs
3_0	Scheduling of NR sidelink in one cell
3_1	Scheduling of LTE sidelink in one cell

Representing DCI formats with MATLAB classes

MATLAB classes can be used to model DCI formats and fields, where a separate class definition represents each format, and the fields of each format are ordered properties of the class.

In this example, the MATLAB class `BitField` represents a single DCI field. Each field object has properties to store the field value, current bit size, and a set of possible sizes, which may depend on the protocol state. This class also defines methods to map the field value to and from information bits.

The MATLAB class `MessageFormat` provides a base class from which to derive specific format classes. Each derived format class defines a set of properties of type `BitField` for all DCI fields, in the order that they appear for that format. The `MessageFormat` base class also defines methods to map all derived class DCI fields to and from information bits. Additionally, the `MessageFormat` class overloads the display, property assignment, and reference functionality to provide easy, direct access to the field values. This class supports optional zero-padding for width alignment, but does not support automatic alignment truncation.

DCI Format 1_0 with CRC Scrambled by SI-RNTI

NR DCI formats often have a large number of fields whose sizes depend on the semi-static UE RRC protocol state. This example uses DCI format 1_0 scrambled by SI-RNTI due to its simple field sequence.

This table describes the ordered fields and bitwidths associated with DCI format 1_0 when the CRC is scrambled by SI-RNTI.

DCI Format 1_0 field (SI-RNTI)	Size
Frequency domain resource assignment	$\lceil log_2(N_{RB}^{DL,BWP}(N_{RB}^{DL,BWP}+1)/2)\rceil$ bits, where $N_{RB}^{DL,BWP}$ is the size of C
Time domain resource assignment	4 bits as defined in Clause 5.1.2.1 of TS 38.214
VRB-to-PRB mapping	1 bit according to TS 38.212 Table 7.3.1.2.2-5
Modulation and coding scheme	5 bits as defined in Clause 5.1.3 of TS 38.214 Table 5.1.3.1-1
Redundancy version	2 bits as defined in TS 38.212 Table 7.3.1.1.1-2
System information indicator	1 bit as defined in TS 38.212 Table 7.3.1.2.1-2
Reserved bits	17 bits for operation in a cell with shared spectrum channel access;

Define a `DCIFormat1_0_SIRNTI` class for the format by deriving from `MessageFormat` and specifying a `BitField` property for each format field. In this format, the first and last fields have bitwidths that depend on protocol state parameters (CORESET 0 size and whether the cell has shared spectrum access) and therefore the field widths are set in the class constructor. This ensures that the bitwidths are sized correctly before using `DCIFormat1_0_SIRNTI`.

```
% DCI format 1_0 with CRC scrambled by SI-RNTI
classdef DCIFormat1_0_SIRNTI < MessageFormat

    properties

        FrequencyDomainResources = BitField();    % Field size depends on CORESET 0 size provided
        TimeDomainResources = BitField(4);        % 4 bits
        VRBToPRBMapping = BitField(1);            % 1 bit
        ModulationCoding = BitField(5);           % 5 bit MCS
        RedundancyVersion = BitField(2);          % 2 bit RV
        SystemInformationIndicator = BitField(1); % 1 bit
        ReservedBits = BitField(15);              % 15 bits reserved (17 bits if Release 16 share
```

```
    end

    methods

        function obj = DCIFormat1_0_SIRNTI(NDLRB,sharedspectrum)
        % Class constructor
          ...
        end

    end

end
```

DCI Format 1_0 with CRC Scrambled by SI-RNTI

Use `DCIFormat1_0_SIRNTI` objects to map field values to information bit payloads for this format, and to parse information bits back into field values.

```
% Create DCI format 1_0 object for SI-RNTI and the given cell configuration parameters
CORESET0NRB = 24;
Rel16SharedSpectrum = 0;
dciformat1_0 = DCIFormat1_0_SIRNTI(CORESET0NRB,Rel16SharedSpectrum);

% The display customization prints the BitField property values directly
% along with the format padding and alignment related properties
display(dciformat1_0)

dciformat1_0 =
  DCIFormat1_0_SIRNTI with field values:

      FrequencyDomainResources: 0
           TimeDomainResources: 0
               VRBToPRBMapping: 0
              ModulationCoding: 0
             RedundancyVersion: 0
    SystemInformationIndicator: 0
                  ReservedBits: 0

   Writeable properties:
                  AlignedWidth: []

   Read-only properties:
                         Width: 37


% Use the info function to get the individual field sizes
info(dciformat1_0,'fieldsizes')

ans = struct with fields:
      FrequencyDomainResources: 9
           TimeDomainResources: 4
               VRBToPRBMapping: 1
              ModulationCoding: 5
             RedundancyVersion: 2
    SystemInformationIndicator: 1
                  ReservedBits: 15
```

```
% Get the non-padded bitwidth for this format and configuration
%
% Without any further alignment padding, the number of information bits
% is given by the Width property
dciformat1_0.Width

ans = 37

% Set the aligned width if the information bit payload should
% be padded to a specific size
dciformat1_0.AlignedWidth = 40

dciformat1_0 =
  DCIFormat1_0_SIRNTI with field values:

      FrequencyDomainResources: 0
           TimeDomainResources: 0
               VRBToPRBMapping: 0
              ModulationCoding: 0
            RedundancyVersion: 0
    SystemInformationIndicator: 0
                  ReservedBits: 0

   Writeable properties:
                  AlignedWidth: 40

   Read-only properties:
                         Width: 40
                  PaddingWidth: 3


% Create a DCI message for transmission on PDCCH
txdci = dciformat1_0;

% Set the DCI field values
%
% The assignment customization allows the BitField property values to be set directly
txdci.FrequencyDomainResources = 10;
txdci.TimeDomainResources = 3;
txdci.ModulationCoding = 3;
txdci.RedundancyVersion = 3;
display(txdci);

txdci =
  DCIFormat1_0_SIRNTI with field values:

      FrequencyDomainResources: 10
           TimeDomainResources: 3
               VRBToPRBMapping: 0
              ModulationCoding: 3
            RedundancyVersion: 3
    SystemInformationIndicator: 0
                  ReservedBits: 0

   Writeable properties:
                  AlignedWidth: 40

   Read-only properties:
                         Width: 40
```

```
                  PaddingWidth: 3


% Map the DCI format fields into information payload bits
dciinfobits = toBits(txdci)

dciinfobits = 40x1 int8 column vector

   0
   0
   0
   0
   0
   1
   0
   1
   0
   0
      ⋮


% Map DCI format fields from information bits
rxdci = fromBits(dciformat1_0,dciinfobits)

rxdci =
  DCIFormat1_0_SIRNTI with field values:

      FrequencyDomainResources: 10
           TimeDomainResources: 3
               VRBToPRBMapping: 0
              ModulationCoding: 3
            RedundancyVersion: 3
    SystemInformationIndicator: 0
                  ReservedBits: 0

   Writeable properties:
                  AlignedWidth: 40

   Read-only properties:
                         Width: 40
                  PaddingWidth: 3


isequal(txdci,rxdci)

ans = logical
   1


% Encode the DCI information bits and send on PDCCH

ncellid = 1;                    % NCellID
si_rnti = hex2dec('FFFF');      % SI-RNTI for PDCCH in a UE-specific search space
K = dciformat1_0.Width;         % Number of DCI message bits
E = 2*288;                      % Number of bits for PDCCH candidate

% Encode DCI
dciCW = nrDCIEncode(dciinfobits,si_rnti,E);
% Create PDCCH QPSK symbols
```

```
sym = nrPDCCH(dciCW,ncellid,0);

% Add noise to PDCCH symbols
EbNo = 3;                       % EbNo in dB
bps = 2;                        % Bits per symbol, 2 for QPSK
EsNo = EbNo + 10*log10(bps);
snrdB = EsNo + 10*log10(K/E);
rxSym = awgn(sym,snrdB,'measured');

% Recover softbits from PDCCH symbols
noiseVar = 10.^(-snrdB/10);     % Assumes unit signal power
rxCW = nrPDCCHDecode(rxSym,ncellid,0,noiseVar);

% Decode DCI information bits
listLen = 8;                    % Polar decoding list length
decDCIBits = nrDCIDecode(rxCW,K,listLen,si_rnti);

% Map DCI format field from info bits
rxdci = fromBits(dciformat1_0,decDCIBits)

rxdci =
  DCIFormat1_0_SIRNTI with field values:

      FrequencyDomainResources: 10
           TimeDomainResources: 3
               VRBToPRBMapping: 0
              ModulationCoding: 3
            RedundancyVersion: 3
    SystemInformationIndicator: 0
                  ReservedBits: 0

   Writeable properties:
                  AlignedWidth: 40

   Read-only properties:
                         Width: 40
                  PaddingWidth: 3


isequal(rxdci,txdci)

ans = logical
   1
```

For more information about the NR downlink control channel, see “Modeling Downlink Control Information” and “Downlink Control Processing and Procedures” on page 1-77. For more information about applying MATLAB classes, see “Representing Structured Data with Classes”.

References

1 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “Downlink Control Processing and Procedures” on page 1-77

NR Positioning Using PRS

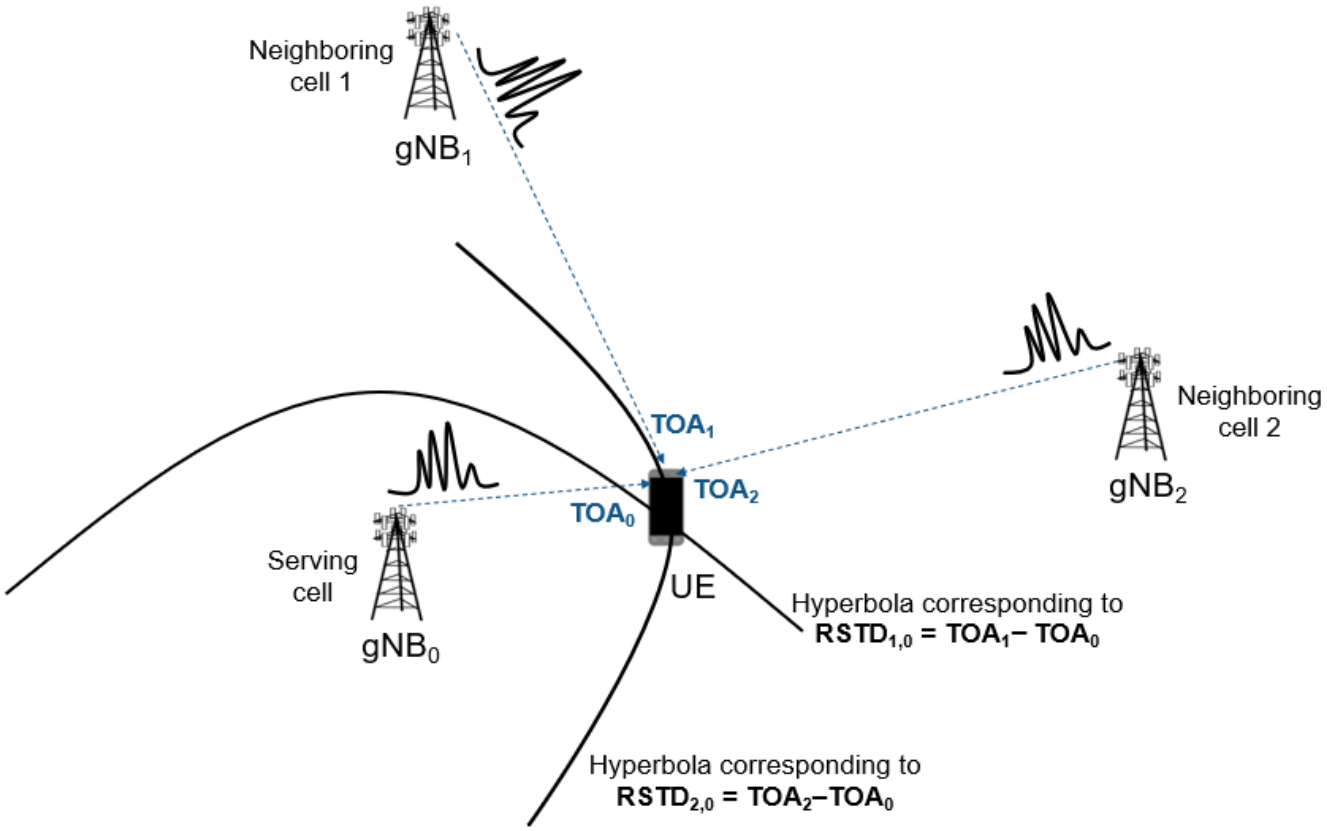
This example shows how to calculate the position of user equipment (UE) within a network of gNodeBs (gNBs) by using an NR positioning reference signal (PRS). The example uses the observed time difference of arrival (OTDOA) positioning approach to estimate the UE position in 2-D, relative to (0,0).

Introduction

The OTDOA positioning approach is one of the downlink-based UE positioning techniques. This technique uses the reference signal timing difference (RSTD) or time difference of arrival (TDOA) measurements to perform multilateration or trilateration by using the theory of hyperbolas. The RSTD is the relative timing difference between the neighboring gNB and the reference gNB (which might be the serving cell), as defined in TS 38.215 Section 5.1.29 [1 on page 1-159]. For the OTDOA technique, the PRS is transmitted by a set of neighboring gNBs along with the serving gNB. In this case, one gNB acts as a reference cell for the RSTD measurements. To avoid a synchronization error in RSTD measurements, all of the gNBs transmit the PRS at the same time (which means synchronized transmission from all gNBs). The time of arrival (TOA) of the PRS signals from different gNBs are estimated at the UE side. RSTD values are computed for a set of neighboring gNB and reference gNB pairs. An RSTD value between neighboring gNB_j and reference gNB_i is defined as, $\mathrm{RSTD}_{j,i} = \mathrm{TOA}_j - \mathrm{TOA}_i$.

Each RSTD value that is obtained for a pair of gNBs corresponds to a hyperbola equation on which the UE is assumed to be located, with foci located at these gNBs. A set of hyperbola equations are deduced from a set of RSTD values, and the equations are solved to estimate the UE position. This process is called *multilateration*. The multilateration process involving one reference gNB and two neighboring gNBs is called *trilateration* (that is, the case of minimum requirement for UE positioning in 2-D).

This figure shows the positioning scenario with one reference gNB (which is the serving gNB in this case) and two neighboring gNBs. The PRS signals that are transmitted by all of the gNBs are received at the UE side at different timing instances (TOA_0, TOA_1, and TOA_2) based on the geographic distance between the UE and gNBs. From this network of three gNBs, two RSTD values are obtained from the TOA values. Each RSTD value corresponds to a hyperbola equation, on which the UE is assumed to be located. You can solve these two hyperbola equations to get UE position in 2-D.



This example shows how to create several gNB transmissions combined with different delays and received powers to model the reception of the waveforms from all of the gNBs by one UE. The UE performs correlation with the PRS to establish the delay from each gNB and subsequently the delay difference between the neighboring gNBs and reference gNB. These delay differences (RSTD values) are used to compute the hyperbolas of constant delay difference and are plotted relative to known gNB positions. The intersection of these hyperbolas is computed to estimate the position of the UE. This example focuses on the 2-D positioning estimate of a UE by assuming that all gNBs are synchronized. This example gives a better estimate for UE position when the UE is located near the center of gNBs network. This example assumes that all gNBs are operated at the same carrier frequency (which corresponds to a single positioning frequency layer). This example also assumes that all of the carriers have the same subcarrier spacing, cyclic prefix, and frequency allocation within the common resource grid.

Simulation Parameters

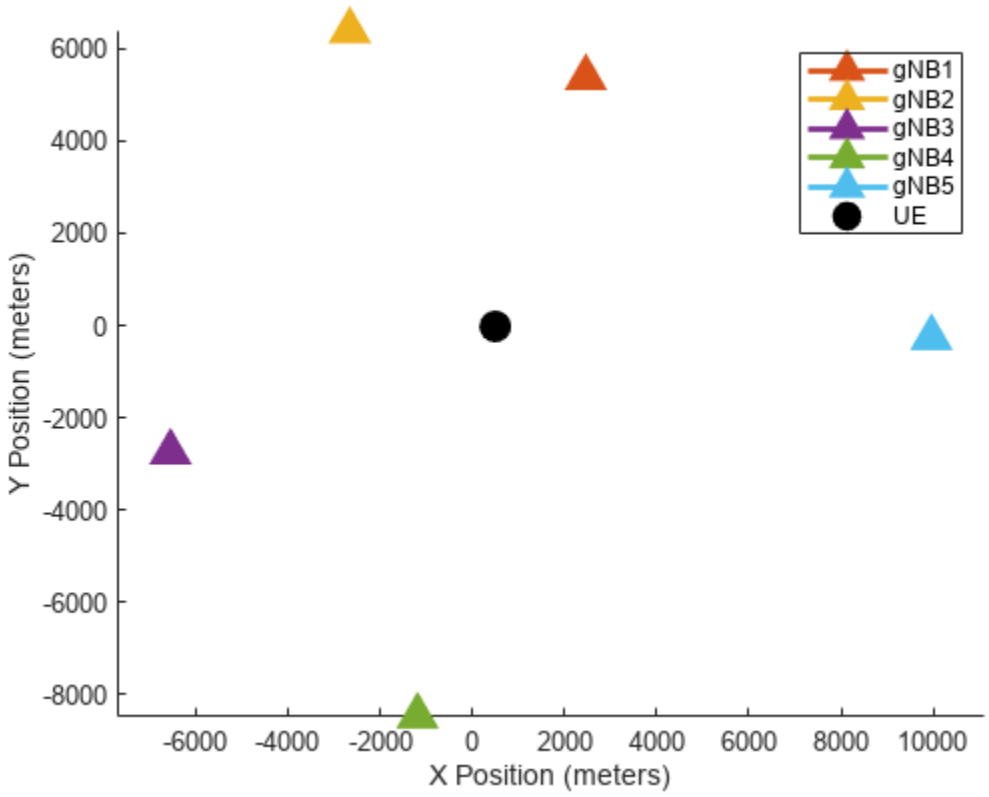
Specify the number of frames and the carrier frequency.

```
nFrames = 1; % Number of 10 ms frames
fc = 3e9;    % Carrier frequency (Hz)
```

Specify UE and gNB positions.

```
% Configure UE position in xy-coordinate plane
UEPos = [500 -20]; % in meters

% Configure number of gNBs and locate them at random positions in
% xy-coordinate plane
```

```
numgNBs = 5;
rng('default');                     % Set RNG state for repeatability
gNBPos = getgNBPositions(numgNBs); % In meters

% Plot UE and gNB positions
plotgNBAndUEPositions(gNBPos,UEPos,1:numgNBs);
```



Configuration Objects

Carrier Configuration

Configure the carriers for all of the gNBs with different physical layer cell identities.

```
cellIds = randperm(1008,numgNBs) - 1;

% Configure carrier properties
carrier = repmat(nrCarrierConfig,1,numgNBs);
for gNBIdx = 1:numgNBs
    carrier(gNBIdx).NCellID = cellIds(gNBIdx);
end
validateCarriers(carrier);
```

PRS Configuration

Configure PRS parameters for all of the gNBs. Configure the PRS for different gNBs such that no overlap exists to avoid the problem of hearability. For more information on how to configure PRS, see “NR Positioning Reference Signal” on page 1-127. This example considers different slot offsets for the PRS from different gNBs to avoid the overlap among PRS signals. This overlapping can be avoided in

multiple ways by configuring the time-frequency aspects of PRS signals in an appropriate way (like choosing the nonoverlapped symbol allocation or frequency allocation, or by using muting pattern configuration parameters).

```
% Slot offsets of different PRS signals
prsSlotOffsets = 0:2:(2*numgNBs - 1);
prsIDs = randperm(4096,numgNBs) - 1;

% Configure PRS properties
prs = nrPRSConfig;
prs.PRSResourceSetPeriod = [10 0];
prs.PRSResourceOffset = 0;
prs.PRSResourceRepetition = 1;
prs.PRSResourceTimeGap = 1;
prs.MutingPattern1 = [];
prs.MutingPattern2 = [];
prs.NumRB = 52;
prs.RBOffset = 0;
prs.CombSize = 12;
prs.NumPRSSymbols = 12;
prs.SymbolStart = 0;
prs = repmat(prs,1,numgNBs);
for gNBIdx = 1:numgNBs
    prs(gNBIdx).PRSResourceOffset = prsSlotOffsets(gNBIdx);
    prs(gNBIdx).NPRSID = prsIDs(gNBIdx);
end
```

PDSCH Configuration

Configure the physical downlink shared channel (PDSCH) parameters for all of the gNBs for the data transmission. This example assumes that the data is transmitted from all of the gNBs that have a single transmission layer.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:51;
pdsch.SymbolAllocation = [0 14];
pdsch.DMRS.NumCDMGroupsWithoutData = 1;
pdsch = repmat(pdsch,1,numgNBs);
validateNumLayers(pdsch);
```

Path Loss Configuration

Create a path loss configuration object for the `'Uma'` scenario.

```
plCfg = nrPathLossConfig;
plCfg.Scenario = 'Uma';
plCfg.BuildingHeight = 5;    % In meters. It is required for 'RMa' scenario
plCfg.StreetWidth = 5;       % In meters. It is required for 'RMa' scenario
plCfg.EnvironmentHeight = 2; % In meters. It is required for 'UMa' and 'UMi' scenarios
```

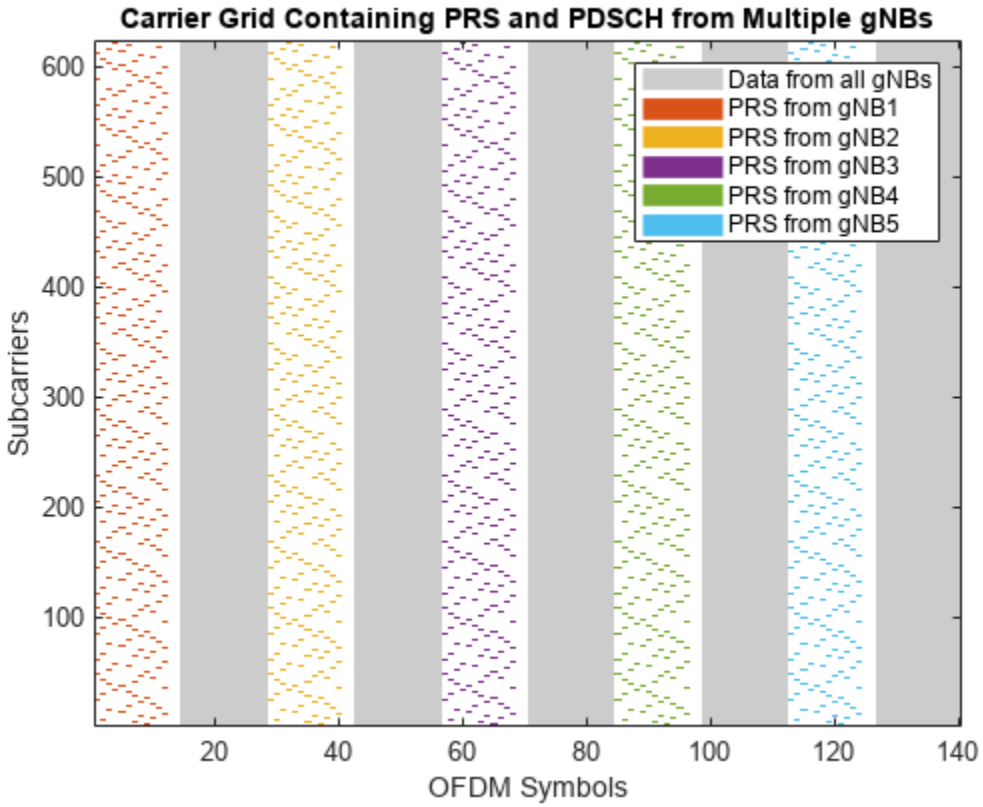
Specify the flag to configure the existence of the line of sight (LOS) path between each gNB and UE pair.

```
los = [true true false true false];
if numel(los) ~= numgNBs
    error('nr5g:InvalidLOSLength',['Length of line of sight flag (' num2str(numel(los)) ...
        ') must be equal to the number of configured gNBs (' num2str(numgNBs) ').']);
end
```

Generate PRS and PDSCH Resources

Generate PRS and PDSCH resources corresponding to all of the gNBs. To improve the hearability of PRS signals, transmit PDSCH resources in the slots in which the PRS is not transmitted by any of the gNBs. This example generates PRS and PDSCH resources with unit power (0 dB).

```
totSlots = nFrames*carrier(1).SlotsPerFrame;
prsGrid = cell(1,numgNBs);
dataGrid = cell(1,numgNBs);
for slotIdx = 0:totSlots-1
    [carrier(:).NSlot] = deal(slotIdx);
    [prsSym,prsInd] = deal(cell(1,numgNBs));
    for gNBIdx = 1:numgNBs
        % Create an empty resource grid spanning one slot in time domain
        slotGrid = nrResourceGrid(carrier(gNBIdx),1);

        % Generate PRS symbols and indices
        prsSym{gNBIdx} = nrPRS(carrier(gNBIdx),prs(gNBIdx));
        prsInd{gNBIdx} = nrPRSIndices(carrier(gNBIdx),prs(gNBIdx));

        % Map PRS resources to slot grid
        slotGrid(prsInd{gNBIdx}) = prsSym{gNBIdx};
        prsGrid{gNBIdx} = [prsGrid{gNBIdx} slotGrid];
    end
    % Transmit data in slots in which the PRS is not transmitted by any of
    % the gNBs (to control the hearability problem)
    for gNBIdx = 1:numgNBs
        dataSlotGrid = nrResourceGrid(carrier(gNBIdx),1);
        if all(cellfun(@isempty,prsInd))
            % Generate PDSCH indices
            [pdschInd,pdschInfo] = nrPDSCHIndices(carrier(gNBIdx),pdsch(gNBIdx));

            % Generate random data bits for transmission
            data = randi([0 1],pdschInfo.G,1);
            % Generate PDSCH symbols
            pdschSym = nrPDSCH(carrier(gNBIdx),pdsch(gNBIdx),data);

            % Generate demodulation reference signal (DM-RS) indices and symbols
            dmrsInd = nrPDSCHDMRSIndices(carrier(gNBIdx),pdsch(gNBIdx));
            dmrsSym = nrPDSCHDMRS(carrier(gNBIdx),pdsch(gNBIdx));

            % Map PDSCH and its associated DM-RS to slot grid
            dataSlotGrid(pdschInd) = pdschSym;
            dataSlotGrid(dmrsInd) = dmrsSym;
        end
        dataGrid{gNBIdx} = [dataGrid{gNBIdx} dataSlotGrid];
    end
end

% Plot carrier grid
plotGrid(prsGrid,dataGrid);
```



Perform OFDM Modulation

Perform orthogonal frequency-division multiplexing (OFDM) modulation of the signals at each gNB.

```
% Perform OFDM modulation of PRS and data signal at each gNB
txWaveform = cell(1,numgNBs);
for waveIdx = 1:numgNBs
    carrier(waveIdx).NSlot = 0;
    txWaveform{waveIdx} = nrOFDMModulate(carrier(waveIdx),prsGrid{waveIdx} + dataGrid{waveIdx});
end

% Compute OFDM information using first carrier, assuming all carriers are
% at same sampling rate
ofdmInfo = nrOFDMInfo(carrier(1));
```

Add Signal Delays and Apply Path Loss

Calculate the time delays from each gNB to UE by using the known gNB and UE positions. This calculation uses the distance between the UE and gNB, `radius`, and the speed of propagation (speed of light). Calculate the sample delay by using the sampling rate, `info.SampleRate`, and store it in `sampleDelay`. This example only applies the integer sample delay by rounding the actual delays to their nearest integers. The example uses these variables to model the environment between gNBs and the UE. The information about these variables is not provided to the UE.

```
speedOfLight = physconst('LightSpeed'); % Speed of light in m/s
sampleDelay = zeros(1,numgNBs);
radius = cell(1,numgNBs);
```

```
for gNBIdx = 1:numgNBs
   radius{gNBIdx} = sqrt((gNBPos{gNBIdx}(1) - UEPos(1))^2 + (gNBPos{gNBIdx}(2) - UEPos(2))^2);
   delay = radius{gNBIdx}/speedOfLight;                      % Delay in seconds
   sampleDelay(gNBIdx) = round(delay*ofdmInfo.SampleRate);   % Delay in samples
end
```

Model the received signal at the UE by delaying each gNB transmission according to the values in `sampleDelay` and by attenuating the received signal from each gNB. Ensure that all of the waveforms are of the same length by padding the received waveform from each gNB with relevant number of zeros.

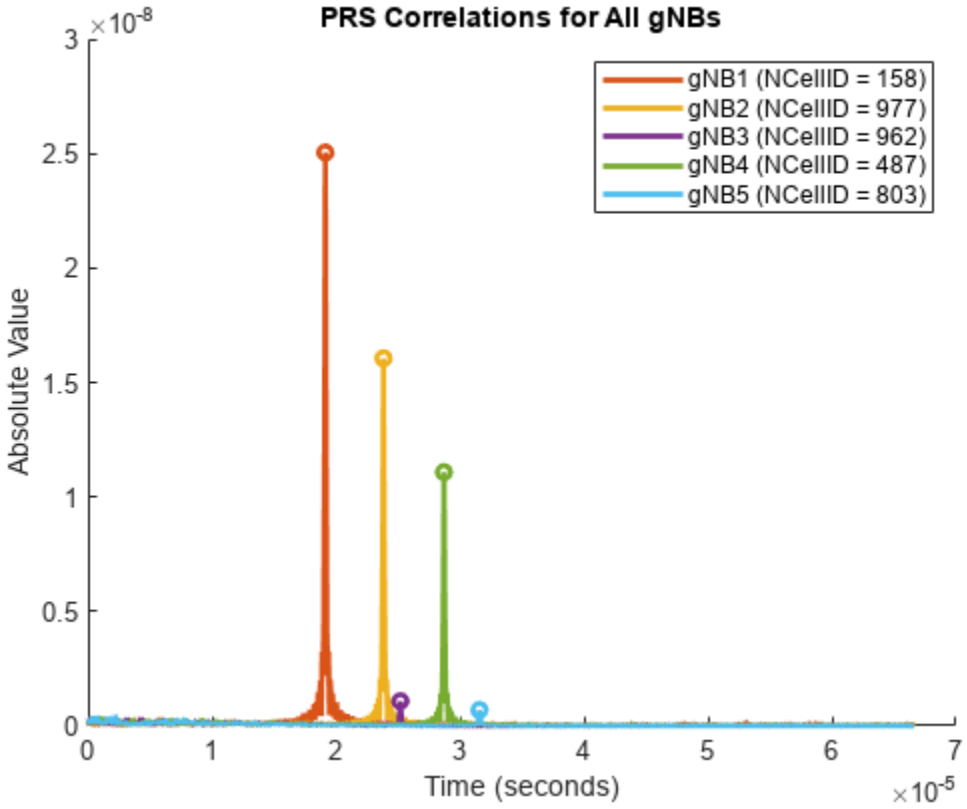
```
rxWaveform = zeros(length(txWaveform{1}) + max(sampleDelay),1);
rx = cell(1,numgNBs);
for gNBIdx = 1:numgNBs
    % Calculate path loss for each gNB and UE pair
    PLdB = nrPathLoss(plCfg,fc,los(gNBIdx),[gNBPos{gNBIdx}(:);0],[UEPos(:);0]);
    if PLdB < 0 || isnan(PLdB) || isinf(PLdB)
        error('nr5g:invalidPL',"Computed path loss (" + num2str(PLdB) + ...
            ") is invalid. Try changing the UE or gNB positions, or path loss configuration.");
    end
    PL = 10^(PLdB/10);

    % Add delay, pad, and attenuate
    rx{gNBIdx} = [zeros(sampleDelay(gNBIdx),1); txWaveform{gNBIdx}; ...
                zeros(max(sampleDelay)-sampleDelay(gNBIdx),1)]/sqrt(PL);

    % Sum waveforms from all gNBs
    rxWaveform = rxWaveform + rx{gNBIdx};
end
```

TOA Estimation

Configure the number of cells or gNBs to be detected for the multilateration process. This example does not consider the transmission of primary and secondary synchronization signals for the purpose of cell search. Instead, the UE performs the correlation on the incoming signal with reference PRS generated for each gNB, and `cellsToBeDetected` number of best cells are selected based on correlation outcome. Compute TOAs of the signals from each gNB by using `nrTimingEstimate` function. To estimate the position of UE, compute RSTD values by using the absolute TOAs corresponding to the best cells.

```
cellsToBeDetected = min(3,numgNBs);
if cellsToBeDetected < 3 || cellsToBeDetected > numgNBs
    error('nr5g:InvalidNumDetgNBs',['The number of detected gNBs (' num2str(cellsToBeDetected) .
        ') must be greater than or equal to 3 and less than or equal to the total number of gNBs
end
corr = cell(1,numgNBs);
delayEst = zeros(1,numgNBs);
maxCorr = zeros(1,numgNBs);
for gNBIdx = 1:numgNBs
    [~,mag] = nrTimingEstimate(carrier(gNBIdx),rxWaveform,prsGrid{gNBIdx});
    % Extract correlation data samples spanning about 1/14 ms for normal
    % cyclic prefix and about 1/12 ms for extended cyclic prefix (this
    % truncation is to ignore noisy side lobe peaks in correlation outcome)
    corr{gNBIdx} = mag(1:(ofdmInfo.Nfft*carrier(1).SubcarrierSpacing/15));
    % Delay estimate is at point of maximum correlation
    maxCorr(gNBIdx) = max(corr{gNBIdx});
    delayEst(gNBIdx) = find(corr{gNBIdx} == maxCorr(gNBIdx),1)-1;
```

```
end

% Get detected gNB numbers based on correlation outcome
[~,detectedgNBs] = sort(maxCorr,'descend');
detectedgNBs = detectedgNBs(1:cellsToBeDetected);

% Plot PRS correlation results
plotPRSCorr(carrier,corr,ofdmInfo.SampleRate);
```



UE Position Estimation Using OTDOA Technique

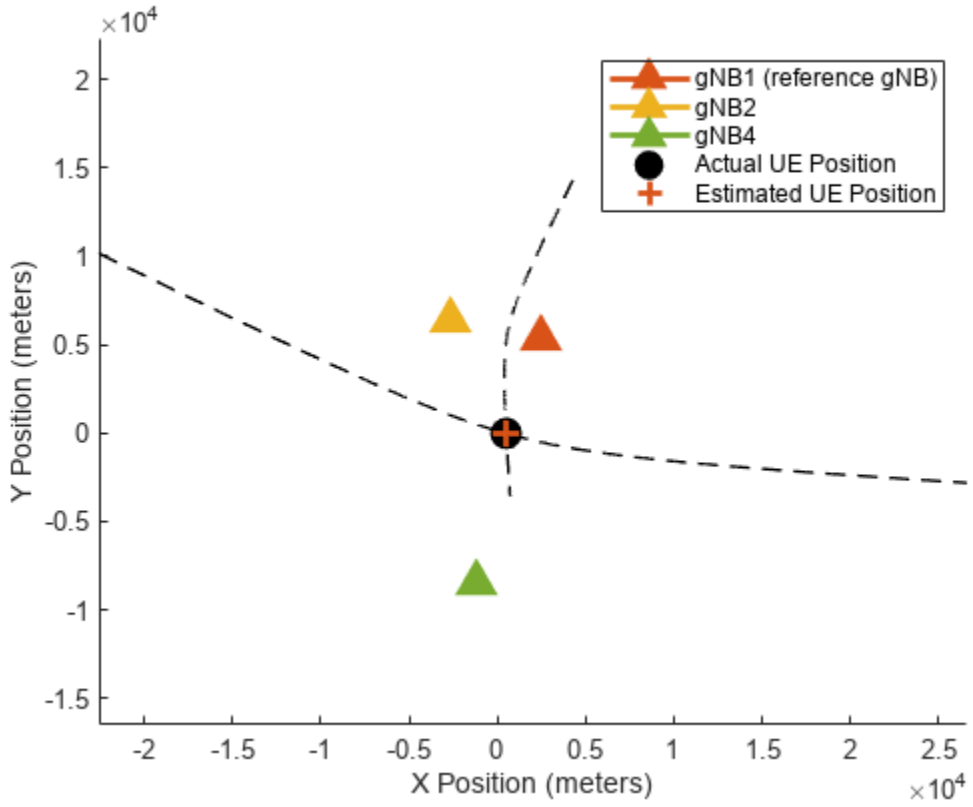
Calculate TDOA or RSTD values for all pairs of gNBs using the estimated TOA values by using the `getRSTDValues` function. Each RSTD value corresponding to a pair of gNBs can result from the UE being located at any position on a hyperbola with foci located at these gNBs.

```
% Compute RSTD values for multilateration or trilateration
rstdVals = getRSTDValues(delayEst,ofdmInfo.SampleRate);
```

Consider RSTD values corresponding to the detected gNBs for the estimation of the UE position. This example assumes the first detected gNB as the reference gNB and the others as neighboring gNBs. Calculate a set of hyperbola equations by using RSTD values corresponding to the pairs of reference gNB and neighboring gNBs. Plot the hyperbola equations, where the intersection of these curves represents the UE position. If you want to know which gNBs correspond to a hyperbola curve, you can know that information from the data tip in the figure.

```
% Plot gNB and UE positions
txCellIDs = [carrier(:).NCellID];
```

```
cellIdx = 1;
curveX = {};
curveY = {};
for jj = detectedgNBs(1) % Assuming first detected gNB as reference gNB
    for ii = detectedgNBs(2:end)
        rstd = rstdVals(ii,jj)*speedOfLight; % Delay distance
        % Establish gNBs for which delay distance is applicable by
        % examining detected cell identities
        txi = find(txCellIDs == carrier(ii).NCellID);
        txj = find(txCellIDs == carrier(jj).NCellID);
        if (~isempty(txi) && ~isempty(txj))
            % Get x- and y- coordinates of hyperbola curve and store them
            [x,y] = getRSTDCurve(gNBPos{txi},gNBPos{txj},rstd);
            if isreal(x) && isreal(y)
                curveX{1,cellIdx} = x; %#ok<*SAGROW>
                curveY{1,cellIdx} = y;

                % Get the gNB numbers corresponding to the current hyperbola
                % curve
                gNBNums{cellIdx} = [jj ii];
                cellIdx = cellIdx + 1;
            else
                warning("Due to the error in timing estimations, " + ...
                    "hyperbola curve cannot be deduced for the combination of gNB" + jj + " and g
            end
        end
    end
end
numHyperbolaCurves = numel(curveX);
if numHyperbolaCurves < 2
    error('nr5g:InsufficientCurves',"The number of hyperbola curves ("+numHyperbolaCurves+") is n
end

% Estimate UE position from hyperbola curves using
% |getEstimatedUEPosition|. This function computes the point of
% intersections of all hyperbola curves and does the average of those
% points to get the UE position.
estimatedPos = getEstimatedUEPosition(curveX,curveY);

% Compute positioning estimation error
EstimationErr = norm(UEPos-estimatedPos); % in meters
disp(['Estimated UE Position       : [' num2str(estimatedPos(1)) ' ' num2str(estimatedPos(2)) ']
      'UE Position Estimation Error: ' num2str(EstimationErr) ' meters']);

Estimated UE Position       : [501.0183 -24.2848]
UE Position Estimation Error: 4.4041 meters

% Plot UE, gNB positions, and hyperbola curves
gNBsToPlot = unique([gNBNums{:}],'stable');
plotPositionsAndHyperbolaCurves(gNBPos,UEPos,gNBsToPlot,curveX,curveY,gNBNums,estimatedPos);
```



Summary and Further Exploration

This example shows OTDOA-based UE positioning estimation in 2-D.

You can vary the number of gNBs, locate the UE and gNBs at different positions.

You can also vary the time-frequency allocation parameters of PRS signals, vary the LOS configuration of each gNB, and see the variations in the estimated UE position.

References

1 3GPP TS 38.215. "NR; Physical layer measurements (Release 16)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local Functions

This example uses these local functions to calculate gNB positions, validate carrier properties and the number of layers, estimate RSTD values, compute the UE position, and plot UE and gNB positions along with the generated hyperbolas.

```
function gNBPos = getgNBPositions(numgNBs)
%   GNBPOS = getgNBPositions(NUMGNBS) returns a cell array of random
%   positions for gNBs, given the number of gNBs NUMGNBS.

    gNBPos = cell(1,numgNBs);
```

```
    for gNBIdx = 1:numgNBs
        % Position gNB randomly within gNBNum*2*pi/numgNBs radian sector
        phi = gNBIdx*2*pi/numgNBs + rand(1,1)*2*pi/(2*numgNBs) - 2*pi/(2*numgNBs);

        % Position gNB randomly between 4000 + (gNBNum*5000/numgNBs) and 5000 +
        % (gNBNum*5000/numgNBs) from UE
        r = randi([0,1000],1,1) + 4000 + (gNBIdx*5000/numgNBs);

        % Convert polar coordinates to Cartesian coordinates
        [x,y] = pol2cart(phi,r);
        gNBPos{gNBIdx} = [x,y];
    end
end

function plotgNBAndUEPositions(gNBPos,UEPos,gNBNums)
%   plotgNBAndUEPositions(GNBPOS,UEPOS,GNBNUMS) plots gNB and UE positions.

    numgNBs = numel(gNBNums);
    colors = getColors(numel(gNBPos));

    figure;
    hold on;
    legendstr = cell(1,numgNBs);

    % Plot position of each gNB
    for gNBIdx = 1:numgNBs
        plot(gNBPos{gNBNums(gNBIdx)}(1),gNBPos{gNBNums(gNBIdx)}(2), ...
            'Color',colors(gNBNums(gNBIdx),:),'Marker','^', ...
            'MarkerSize',11,'LineWidth',2,'MarkerFaceColor',colors(gNBNums(gNBIdx),:));
        legendstr{gNBIdx} = sprintf('gNB%d',gNBIdx);
    end

    % Plot UE position
    plot(UEPos(1),UEPos(2),'ko','MarkerSize',10,'LineWidth',2,'MarkerFaceColor','k');

    axis equal;
    legend([legendstr 'UE']);
    xlabel('X Position (meters)');
    ylabel('Y Position (meters)');

end

function validateCarriers(carrier)
%   validateCarriers(CARRIER) validates the carrier properties of all
%   gNBs.

    % Validate physical layer cell identities
    cellIDs = [carrier(:).NCellID];
    if numel(cellIDs) ~= numel(unique(cellIDs))
        error('nr5g:invalidNCellID','Physical layer cell identities of all of the carriers must b
    end

    % Validate subcarrier spacings
    scsVals = [carrier(:).SubcarrierSpacing];
    if ~isscalar(unique(scsVals))
        error('nr5g:invalidSCS','Subcarrier spacing values of all of the carriers must be same.')
    end
```

```
    % Validate cyclic prefix lengths
    cpVals = {carrier(:).CyclicPrefix};
    if ~all(strcmpi(cpVals{1},cpVals))
        error('nr5g:invalidCP','Cyclic prefix lengths of all of the carriers must be same.');
    end

    % Validate NSizeGrid values
    nSizeGridVals = [carrier(:).NSizeGrid];
    if ~isscalar(unique(nSizeGridVals))
        error('nr5g:invalidNSizeGrid','NSizeGrid of all of the carriers must be same.');
    end

    % Validate NStartGrid values
    nStartGridVals = [carrier(:).NStartGrid];
    if ~isscalar(unique(nStartGridVals))
        error('nr5g:invalidNStartGrid','NStartGrid of all of the carriers must be same.');
    end

end

function validateNumLayers(pdsch)
%   validateNumLayers(PDSCH) validates the number of transmission layers,
%   given the array of physical downlink shared channel configuration
%   objects PDSCH.

numLayers = [pdsch(:).NumLayers];
if ~all(numLayers == 1)
    error('nr5g:invalidNLayers',['The number of transmission layers ' ...
        'configured for the data transmission must be 1.']);
end
end

function plotPRSCorr(carrier,corr,sr)
%   plotPRSCorr(CARRIER,CORR,SR) plots PRS-based correlation for each gNB
%   CORR, given the array of carrier-specific configuration objects CARRIER
%   and the sampling rate SR.

    numgNBs = numel(corr);
    colors = getColors(numgNBs);
    figure;
    hold on;

    % Plot correlation for each gNodeB
    t = (0:length(corr{1}) - 1)/sr;
    legendstr = cell(1,2*numgNBs);
    for gNBIdx = 1:numgNBs
        plot(t,abs(corr{gNBIdx}), ...
            'Color',colors(gNBIdx,:),'LineWidth',2);
        legendstr{gNBIdx} = sprintf('gNB%d (NCellID = %d)', ...
            gNBIdx,carrier(gNBIdx).NCellID);
    end

    % Plot correlation peaks
    for gNBIdx = 1:numgNBs
        c = abs(corr{gNBIdx});
        j = find(c == max(c),1);
        plot(t(j),c(j),'Marker','o','MarkerSize',5, ...
            'Color',colors(gNBIdx,:),'LineWidth',2);
```

```
            legendstr{numgNBs+gNBIdx} = '';
        end
        legend(legendstr);
        xlabel('Time (seconds)');
        ylabel('Absolute Value');
        title('PRS Correlations for All gNBs');
    end

    function rstd = getRSTDValues(toa,sr)
    %   RSTD = getRSTDValues(TOA,SR) computes RSTD values, given the time of
    %   arrivals TOA and the sampling rate SR.

        % Compute number of samples delay between arrivals from different gNBs
        rstd = zeros(length(toa));
        for jj = 1:length(toa)
            for ii = 1:length(toa)
                rstd(ii,jj) = toa(ii) - toa(jj);
            end
        end

        % Get RSTD values in time
        rstd = rstd./sr;

    end

    function [x,y] = getRSTDCurve(gNB1,gNB2,rstd)
    %   [X,Y] = getRSTDCurve(GNB1,GNB2,RSTD) returns the x- and y-coordinates
    %   of a hyperbola equation corresponding to an RSTD value, given the pair
    %   of gNB positions gNB1 and gNB2.

        % Calculate vector between two gNBs
        delta = gNB1 - gNB2;

        % Express distance vector in polar form
        [phi,r] = cart2pol(delta(1),delta(2));
        rd = (r+rstd)/2;

        % Compute the hyperbola parameters
        a = (r/2)-rd;         % Vertex
        c = r/2;              % Focus
        b = sqrt(c^2-a^2);    % Co-vertex
        hk = (gNB1 + gNB2)/2;
        mu = -2:1e-3:2;

        % Get x- and y- coordinates of hyperbola equation
        x = (a*cosh(mu)*cos(phi)-b*sinh(mu)*sin(phi)) + hk(1);
        y = (a*cosh(mu)*sin(phi)+b*sinh(mu)*cos(phi)) + hk(2);

    end

    function plotPositionsAndHyperbolaCurves(gNBPos,UEPos,detgNBNums,curveX,curveY,gNBNums,estPos)
    %   plotPositionsAndHyperbolaCurves(GNBPOS,UEPOS,DETGNBNUMS,CURVEX,CURVEY,GNBNUMS,ESTPOS)
    %   plots gNB, UE positions, and hyperbola curves by considering these
    %   inputs:
    %   GNBPOS     - Positions of gNBs
    %   UEPOS      - Position of UE
    %   DETGNBNUMS - Detected gNB numbers
    %   CURVEX     - Cell array having the x-coordinates of hyperbola curves
```

```
%   CURVEY     - Cell array having the y-coordinates of hyperbola curves
%   GNBNUMS    - Cell array of gNB numbers corresponding to all hyperbolas
%   ESTPOS     - Estimated UE position

    plotgNBAndUEPositions(gNBPos,UEPos,detgNBNums);
    for curveIdx = 1:numel(curveX)
        curves(curveIdx) = plot(curveX{1,curveIdx},curveY{1,curveIdx}, ...
               '--','LineWidth',1,'Color','k');
    end
    % Add estimated UE position to figure
    plot(estPos(1),estPos(2),'+','MarkerSize',10, ...
        'MarkerFaceColor','#D95319','MarkerEdgeColor','#D95319','LineWidth',2);

    % Add legend
    gNBLegends = cellstr(repmat("gNB",1,numel(detgNBNums)) + ...
        detgNBNums + [" (reference gNB)" repmat("",1,numel(detgNBNums)-1)]);
    legend([gNBLegends {'Actual UE Position'} repmat({''},1,numel(curveX)) {'Estimated UE Positio

    for curveIdx = 1:numel(curves)
        % Create a data tip and add a row to the data tip to display
        % gNBs information for each hyperbola curve
        dt = datatip(curves(curveIdx));
        row = dataTipTextRow("Hyperbola of gNB" + gNBNums{curveIdx}(1) + ...
            " and gNB" + gNBNums{curveIdx}(2),'');
        curves(curveIdx).DataTipTemplate.DataTipRows = [row; curves(curveIdx).DataTipTemplate.Dat
        % Delete the data tip that is added above
        delete(dt);
    end

end

function colors = getColors(numgNBs)
%    COLORS = getColors(NUMGNBS) returns the RGB triplets COLORS for the
%    plotting purpose, given the number of gNBs NUMGNBS.

    % Form RGB triplets
    if numgNBs <= 10
        colors = [0.8500, 0.3250, 0.0980; 0.9290, 0.6940, 0.1250; 0.4940, 0.1840, 0.5560; ...
                  0.4660, 0.6740, 0.1880; 0.3010, 0.7450, 0.9330; 0.6350, 0.0780, 0.1840; ...
                  0.9290, 0.9, 0.3; 0.9290, 0.5, 0.9; 0.6660, 0.3740, 0.2880;0, 0.4470, 0.7410];
    else
        % Generate 30 more extra RGB triplets than required. It is to skip
        % the gray and white shades as these shades are reserved for the
        % data and no transmissions in carrier grid plot in this example.
        % With this, you can get unique colors for up to 1000 gNBs.
        colors = colorcube(numgNBs+30);
        colors = colors(1:numgNBs,:);
    end
end

function plotGrid(prsGrid,dataGrid)
%   plotGrid(PRSGRID,DATAGRID) plots the carrier grid from all gNBs.

    numgNBs = numel(prsGrid);
    figure()
    mymap = [1 1 1; ...       % White color for background
             0.8 0.8 0.8; ... % Gray color for PDSCH data from all gNBs
             getColors(numgNBs)];
```

```
        chpval = 3:numgNBs+2;

        gridWithPRS = zeros(size(prsGrid{1}));
        gridWithData = zeros(size(dataGrid{1}));
        names = cell(1,numgNBs);
        for gNBIdx = 1:numgNBs
            gridWithPRS = gridWithPRS + abs(prsGrid{gNBIdx})*chpval(gNBIdx);
            gridWithData = gridWithData + abs(dataGrid{gNBIdx});
            names{gNBIdx} = ['PRS from gNB' num2str(gNBIdx)];
        end
        names = [{'Data from all gNBs'} names];
        % Replace all zero values of gridWithPRS with proper scaling (value of 1)
        % for white background
        gridWithPRS(gridWithPRS == 0) = 1;
        gridWithData(gridWithData ~=0) = 1;

        % Plot grid
        image(gridWithPRS+gridWithData); % In the resultant grid, 1 represents
                                         % the white background, 2 (constitute
                                         % of 1s from gridWithPRS and
                                         % gridWithData) represents PDSCH, and
                                         % so on
        % Apply colormap
        colormap(mymap);
        axis xy;

        % Generate lines
        L = line(ones(numgNBs+1),ones(numgNBs+1),'LineWidth',8);
        % Index colormap and associate selected colors with lines
        set(L,{'color'},mat2cell(mymap(2:numgNBs+2,:),ones(1,numgNBs+1),3)); % Set the colors

        % Create legend
        legend(names{:});

        % Add title
        title('Carrier Grid Containing PRS and PDSCH from Multiple gNBs');

        % Add labels to x-axis and y-axis
        xlabel('OFDM Symbols');
        ylabel('Subcarriers');
    end

    function estPos = getEstimatedUEPosition(xCell,yCell)
    %   ESTPOS = getEstimatedUEPosition(XCELL,YCELL) returns the x- and y-
    %   coordinates of the UE position, given the hyperbolas XCELL and YCELL.

        % Steps involved in this computation:
        % 1. Find closest points between hyperbolic surfaces
        % 2. Linearize surfaces around the points closest to intersection to
        % interpolate actual intersection location

        numCurves = numel(xCell);
        % Make all vectors have equal length
        maxLen = max(cellfun(@length,xCell));
        for curveIdx = 1:numCurves
            xCell{curveIdx} = [xCell{curveIdx} inf(1,maxLen-length(xCell{curveIdx}))];
            yCell{curveIdx} = [yCell{curveIdx} inf(1,maxLen-length(yCell{curveIdx}))];
        end
```

```
    tempIdx = 1;
    for idx1 = 1:numCurves-1
        for idx2 = (idx1+1):numCurves
            [firstCurve,secondCurve] = findMinDistanceElements(xCell{idx1},yCell{idx1},xCell{idx2
            for idx3 = 1:numel(firstCurve)
                [x1a,y1a,x1b,y1b] = deal(firstCurve{idx3}(1,1),firstCurve{idx3}(1,2), ...
                                         firstCurve{idx3}(2,1),firstCurve{idx3}(2,2));
                [x2a,y2a,x2b,y2b] = deal(secondCurve{idx3}(1,1),secondCurve{idx3}(1,2), ...
                                         secondCurve{idx3}(2,1),secondCurve{idx3}(2,2));
                a1 = (y1b-y1a)/(x1b-x1a);
                b1 = y1a - a1*x1a;

                a2 = (y2b-y2a)/(x2b-x2a);
                b2 = y2a - a2*x2a;

                xC(tempIdx) = (b2-b1)/(a1-a2);
                yC(tempIdx) = a1*xC(tempIdx) + b1;
                tempIdx = tempIdx+1;
            end
        end
    end
    estPos = [mean(xC) mean(yC)];
end

function [firstCurvePoints,secondCurvePoints] = findMinDistanceElements(xA,yA,xB,yB)
%   [FIRSTCURVEPOINTS,SECONDCURVEPOINTS] = findMinDistanceElements(XA,YA,XB,YB)
%   returns the closest points between the given hyperbolic surfaces.

    distAB1 = zeros(numel(xA),numel(xB));
    for idx1 = 1:numel(xA)
        distAB1(idx1,:) = sqrt((xB-xA(idx1)).^2 + (yB-yA(idx1)).^2);
    end
    [~,rows] = min(distAB1,[],'omitnan');
    [~,col] = min(min(distAB1,[],'omitnan'));

    % Extract the indices of the points on two curves which are up to 5
    % meters apart. This is to identify the multiple intersections between
    % two hyperbola curves.
    [allRows,allCols] = find(distAB1 <= distAB1(rows(col),col)+5);
    diffVec = abs(allRows - allRows(1));
    index = find(diffVec > (min(diffVec) + max(diffVec))/2,1);
    allRows = [allRows(1);allRows(index)];
    allCols = [allCols(1);allCols(index)];

    for idx = 1:numel(allRows)
        firstCurveIndices = allRows(idx);
        secondCurveIndices = allCols(idx);
        x1a = xA(firstCurveIndices);
        y1a = yA(firstCurveIndices);
        x2a = xB(secondCurveIndices);
        y2a = yB(secondCurveIndices);

        % Use subsequent points to create line for linearization
        if firstCurveIndices == numel(rows)
            x1b = xA(firstCurveIndices-1);
            y1b = yA(firstCurveIndices-1);
        else
            x1b = xA(firstCurveIndices+1);
```

```
            y1b = yA(firstCurveIndices+1);
        end
        if secondCurveIndices == numel(rows)
            x2b = xB(secondCurveIndices-1);
            y2b = yB(secondCurveIndices-1);
        else
            x2b = xB(secondCurveIndices+1);
            y2b = yB(secondCurveIndices+1);
        end
        firstCurvePoints{idx} = [x1a y1a;x1b y1b]; %#ok<*AGROW>
        secondCurvePoints{idx} = [x2a y2a;x2b y2b];
    end
end
```

See Also

Objects
nrPRSConfig | nrCarrierConfig

Functions
nrPRS | nrPRSIndices

Related Examples

- “NR Positioning Reference Signal” on page 1-127

Configure OFDM Sample Rate and FFT Size

5G NR downlink and uplink transmissions use orthogonal frequency-division multiplexing (OFDM) modulation. You can perform OFDM modulation and OFDM demodulation by calling the `nrOFDMModulate` and `nrOFDMDemodulate` functions, respectively. You can also obtain dimensional information about the modulation by calling the `nrOFDMInfo` function.

These functions enable you to specify the OFDM sample rate and FFT size by using the `SampleRate` and `Nfft` name-value input arguments. When the specified sample rate and the nominal sample rate, corresponding to the FFT size, do not match, the OFDM functions resample the waveform. To learn more about the internal resampling filter, see “Resampling Filter Design in OFDM Functions” on page 1-182. To learn more about how the values you specify for the OFDM sample rate and FFT size influence the output of the `nrOFDMInfo` function, see Get OFDM Information on page 1-0 .

If you do not specify the `SampleRate` and `Nfft` inputs, the functions set default values for these inputs.

To learn more about the various OFDM sample rate and FFT configuration options and their effect on the bandwidth occupancy, see these examples.

- The “Use Default OFDM Sample Rate and Default FFT Size” on page 1-168 example plots the bandwidth occupancy for the default values.
- The “Use Default OFDM Sample Rate and Custom FFT Size” on page 1-171 example plots the bandwidth occupancy for the default sample rate with the FFT size selected for maximum occupancy of 90%.
- The “Use Custom OFDM Sample Rate and Default FFT Size” on page 1-174 example plots the FFT occupancy for a custom sample rate with a bandwidth occupancy of 75%.
- The “Use Custom OFDM Sample Rate and Custom FFT Size” on page 1-177 example plots the FFT occupancy for a custom sample rate with a bandwidth occupancy of 90%.

See Also

Functions
`nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

Use Default OFDM Sample Rate and Default FFT Size

This example shows how the OFDM functions (`nrOFDMModulate`, `nrOFDMInfo`, and `nrOFDMDemodulate`) set the default value for the sample rate input, `SampleRate`, and the default value for the fast Fourier transform (FFT) size input, `Nfft`, when you call an OFDM function and these conditions apply.

- You do not specify a value for the `SampleRate` input, or you specify `'SampleRate',[]`.
- You do not specify a value for the `Nfft` input, or you specify `'Nfft',[]`.

Default OFDM Sample Rate

The default value set for the `SampleRate` input is equal to `Nfft*carrier.SubcarrierSpacing*1000`, where `carrier` is the input argument of the function call that specifies the carrier configuration.

Default FFT Size

The default value set for the `Nfft` input satisfies these conditions.

- `Nfft` is an integer greater than 127 (to ensure integer-valued cyclic prefix lengths)
- `Nfft` is a power of 2.
- `Nfft` results in a maximum occupancy of 85%. The actual occupancy is equal to `carrier.NSizeGrid*12/Nfft`.

Plot Bandwidth Occupancy

Create a carrier configuration object.

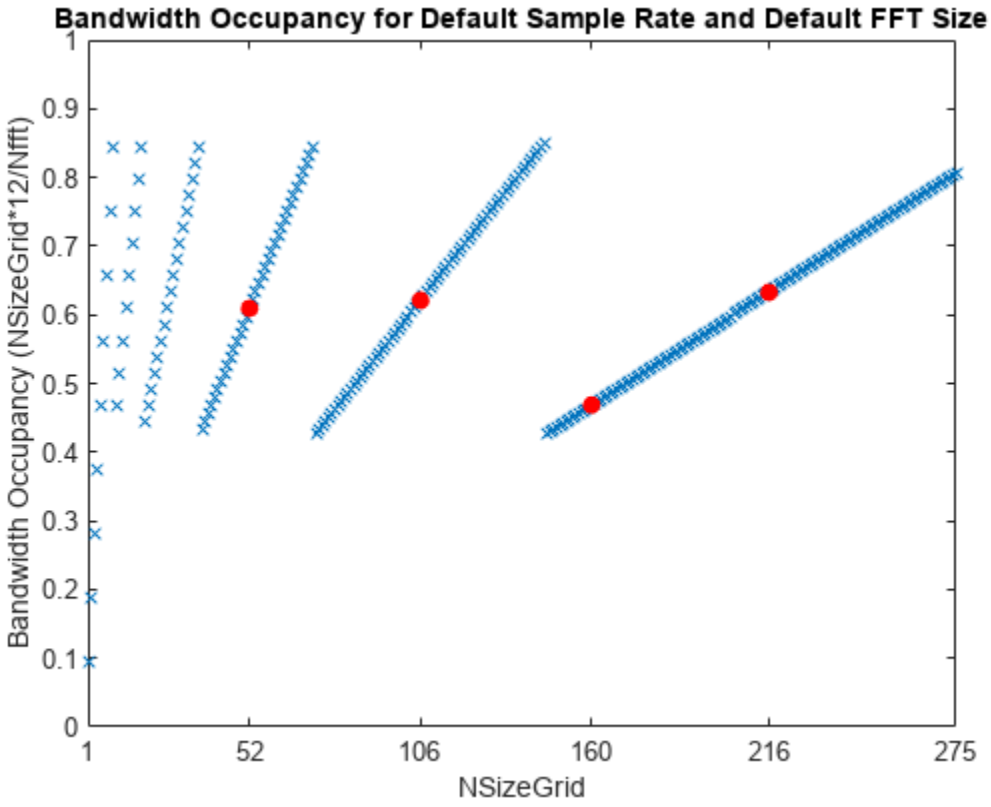
```
carrier = nrCarrierConfig;
```

Calculate the actual occupancy equal to `carrier.NSizeGrid*12/Nfft` for all `NSizeGrid` values.

```
nSizeGrids = 1:275;
defaultOccupancy = zeros(1,275);
for nSizeGrid = nSizeGrids
    carrier.NSizeGrid = nSizeGrid;
    ofdmInfo = nrOFDMInfo(carrier);
    defaultOccupancy(nSizeGrid) = carrier.NSizeGrid*12/ofdmInfo.Nfft;
end
```

Plot the actual occupancy. Highlight the occupancy for `NSizeGrid` values `52`, `106`, `160`, and `216`.

```
figure;
plot(nSizeGrids,defaultOccupancy,'x');
hold on;
typicalNSizeGrids = [52 106 160 216];
plot(typicalNSizeGrids,defaultOccupancy(typicalNSizeGrids),'ro','MarkerFaceColor','r');
title('Bandwidth Occupancy for Default Sample Rate and Default FFT Size');
axis([1 275 0 1]);
xlabel('NSizeGrid');
xticks([1 typicalNSizeGrids 275]);
ylabel('Bandwidth Occupancy (NSizeGrid*12/Nfft)');
```



Ignore the smallest `NSizeGrid` values.

```
defaultOccupancy(1:5) = NaN;
```

Find the minimum occupancy and the corresponding `NSizeGrid` value.

```
[occupancyMin,nSizeGridMin] = min(defaultOccupancy)
```

```
occupancyMin = 0.4277
```

```
nSizeGridMin = 73
```

Find the maximum occupancy and the corresponding `NSizeGrid` value.

```
[occupancyMax,nSizeGridMax] = max(defaultOccupancy)
```

```
occupancyMax = 0.8496
```

```
nSizeGridMax = 145
```

See Also

Functions

`nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

More About

- “Configure OFDM Sample Rate and FFT Size” on page 1-167

Use Default OFDM Sample Rate and Custom FFT Size

This example shows how the OFDM functions (`nrOFDMModulate`, `nrOFDMInfo`, and `nrOFDMDemodulate`) set the default value for the sample rate input, `SampleRate`, when you call an OFDM function and these conditions apply.

- You do not specify a value for the `SampleRate` input, or you specify `'SampleRate',[]`.
- You specify a custom value for the fast Fourier transform (FFT) size input, `Nfft`.

Default OFDM Sample Rate

The default value set for the `SampleRate` input is equal to `Nfft*carrier.SubcarrierSpacing*1000`.

Custom FFT Size

The value that you set for the `Nfft` input must satisfy these conditions.

- `Nfft` is an integer (to ensure integer-valued cyclic prefix lengths).
- `Nfft` is a power of 2.
- `Nfft` results in a maximum occupancy of 100%. The actual occupancy is equal to `carrier.NSizeGrid*12/Nfft`, where `carrier` is the input argument of the function call that specifies the carrier configuration.

Plot Bandwidth Occupancy

Create a carrier configuration object.

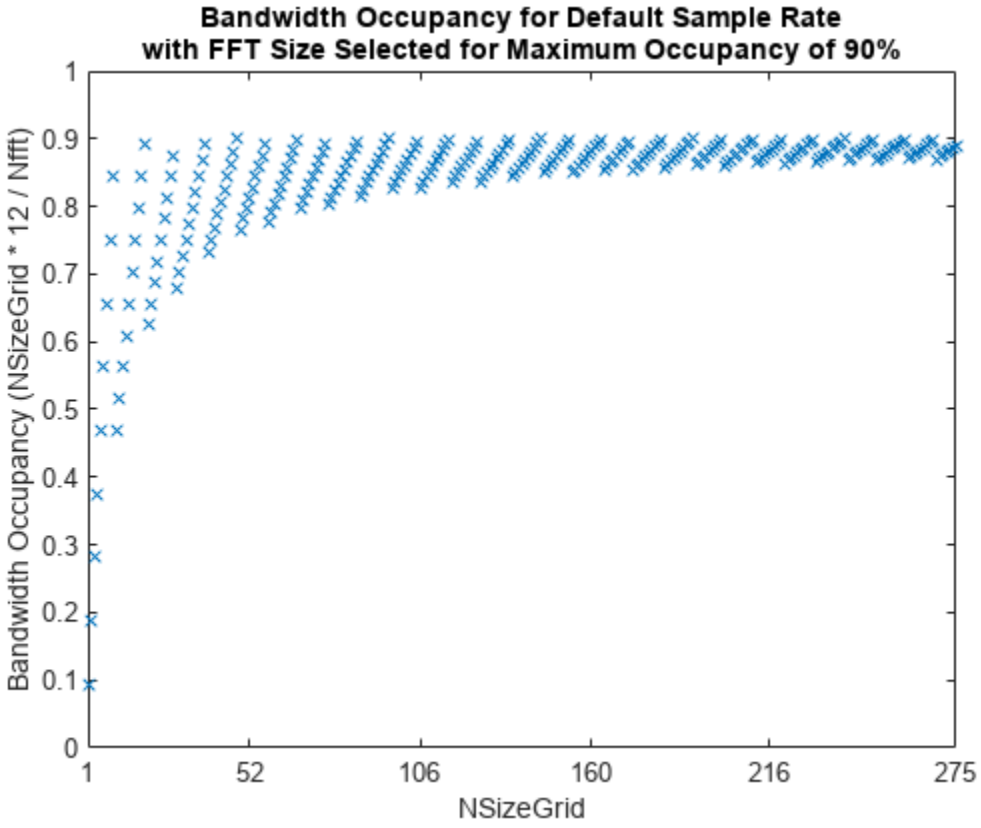
```
carrier = nrCarrierConfig;
```

Set the `Nfft` for each `NSizeGrid` value to give an occupancy of at most 90%.

```
nSizeGrids = 1:275;
userNfftOccupancy = zeros(1,275);
sampleRate = zeros(1,275);
for nSizeGrid = nSizeGrids
    carrier.NSizeGrid = nSizeGrid;
    nfft = 128 * ceil(carrier.NSizeGrid*12/0.9/128);
    ofdmInfo = nrOFDMInfo(carrier,'Nfft',nfft);
    userNfftOccupancy(nSizeGrid) = carrier.NSizeGrid*12/ofdmInfo.Nfft;
    sampleRate(nSizeGrid) = ofdmInfo.SampleRate/1e6;
end
```

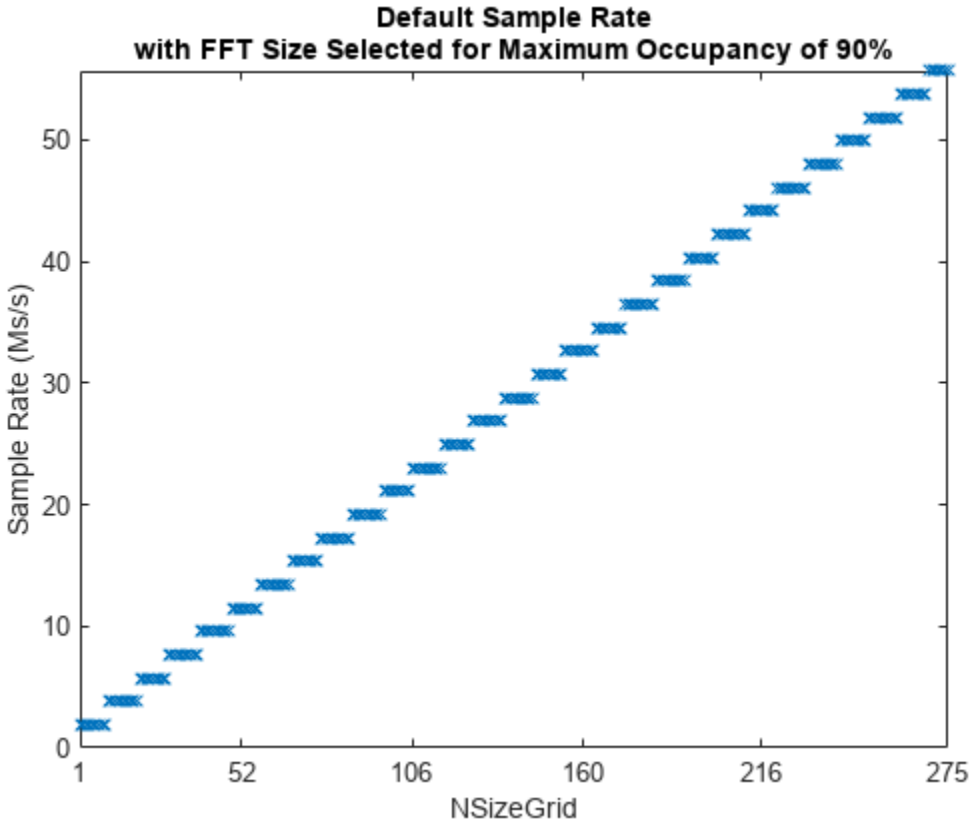
Plot the resulting occupancy.

```
figure;
plot(nSizeGrids,userNfftOccupancy,'x');
title({'Bandwidth Occupancy for Default Sample Rate' 'with FFT Size Selected for Maximum Occupanc
axis([1 275 0 1]);
xlabel('NSizeGrid');
xticks([1 52 106 160 216 275]);
ylabel('Bandwidth Occupancy (NSizeGrid * 12 / Nfft)');
```



Plot the resulting sample rate.

```
figure;
plot(nSizeGrids,sampleRate,'x');
title({'Default Sample Rate' 'with FFT Size Selected for Maximum Occupancy of 90%'});
axis([1 275 0 max(sampleRate)]);
xlabel('NSizeGrid');
xticks([1 52 106 160 216 275]);
ylabel('Sample Rate (Ms/s)');
```



See Also

Functions

`nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

More About

- “Configure OFDM Sample Rate and FFT Size” on page 1-167

Use Custom OFDM Sample Rate and Default FFT Size

This example shows how the OFDM functions (`nrOFDMModulate`, `nrOFDMInfo`, and `nrOFDMDemodulate`) set the default value for the fast Fourier transform (FFT) size input, `Nfft`, when you call an OFDM function and these conditions apply.

- You specify a custom value for the sample rate input, `SampleRate`.
- You do not specify a value for the `Nfft` input, or you specify `'Nfft',[]`.

Custom OFDM Sample Rate

The value that you set for the `SampleRate` input determines the sample rate of the waveform.

The nominal sample rate corresponding to the FFT size used in the OFDM modulation, FFT_{SR}, is equal to `Nfft*carrier.SubcarrierSpacing*1000`, where `carrier` is the input argument of the function call that specifies the carrier configuration.

Because the resampling of the OFDM-modulated waveform is by a factor of `SampleRate`/FFT_{SR}, the resampling is costly if `SampleRate` and FFT_{SR} do not have large common factors.

Default FFT Size

The default value set for the `Nfft` input satisfies these conditions.

- `Nfft` is a multiple of 128, that is, `Nfft = 128 *` Y (to ensure integer-valued cyclic prefix lengths).
- The lower bound of Y is equal to `ceil((carrier.NSizeGrid*12/0.85)/128)`. That is, the lower bound depends on the maximum FFT occupancy of 85%.
- The upper bound of Y is equal to `ceil((2*SampleRate/(carrier.SubcarrierSpacing*1000))/128)`. That is, the upper bound depends on the sample rate you specify.
- Y maximizes the common factors of the nominal and specified sample rates, that is, `gcd`(FFT_{SR},`SampleRate`).

Plot Bandwidth Occupancy

Create a carrier configuration object.

```
carrier = nrCarrierConfig;
```

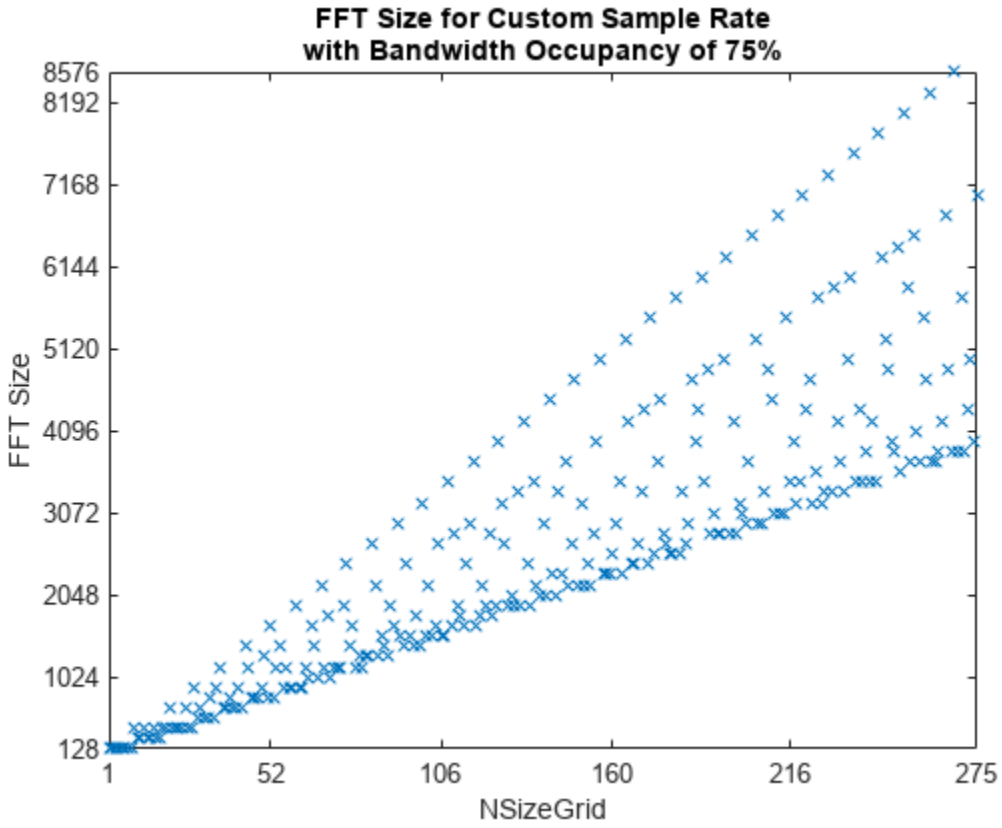
Set the `SampleRate` for each `NSizeGrid` value to give an occupancy of 75%.

```
nSizeGrids = 1:275;
userSampleRateDefaultNfft = zeros(1,275);
fftOccupancy = zeros(1,275);
for nSizeGrid = 1:275
    carrier.NSizeGrid = nSizeGrid;
    % Transmission bandwidth of OFDM waveform
    txBW = carrier.NSizeGrid*12*carrier.SubcarrierSpacing*1000;
    sr = txBW / 0.75;
    ofdmInfo = nrOFDMInfo(carrier,'SampleRate',sr);
    userSampleRateDefaultNfft(nSizeGrid) = ofdmInfo.Nfft;
```

```
    fftOccupancy(nSizeGrid) = carrier.NSizeGrid*12/ofdmInfo.Nfft;
end
```

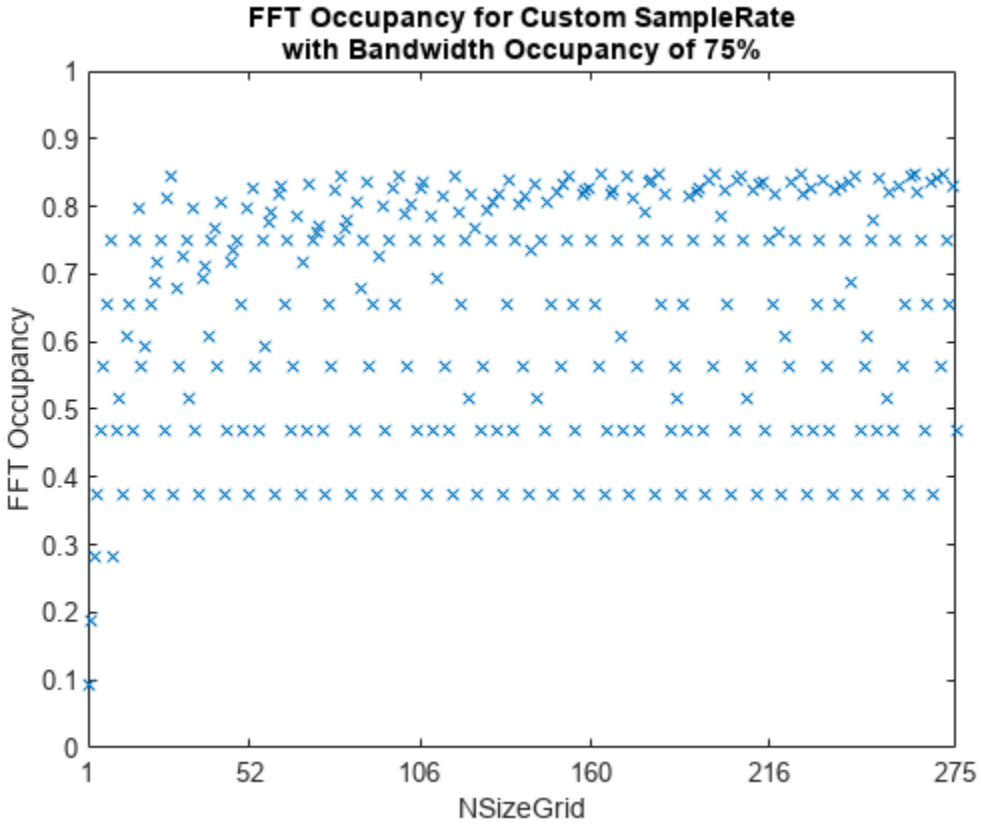
Plot the resulting `Nfft` values.

```
figure;
plot(nSizeGrids,userSampleRateDefaultNfft,'x');
title({'FFT Size for Custom Sample Rate' 'with Bandwidth Occupancy of 75%'});
axis([1 275 min(userSampleRateDefaultNfft) max(userSampleRateDefaultNfft)]);
xlabel('NSizeGrid');
xticks([1 52 106 160 216 275]);
ylabel('FFT Size');
yticks([min(userSampleRateDefaultNfft) 1024:1024:8192 max(userSampleRateDefaultNfft)]);
```



Plot the resulting FFT occupancy.

```
figure;
plot(nSizeGrids,fftOccupancy,'x');
title({'FFT Occupancy for Custom SampleRate' 'with Bandwidth Occupancy of 75%'});
axis([1 275 0 1]);
xlabel('NSizeGrid');
xticks([1 52 106 160 216 275]);
ylabel('FFT Occupancy');
```



See Also

Functions

`nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

More About

- “Resampling Filter Design in OFDM Functions” on page 1-182
- “Configure OFDM Sample Rate and FFT Size” on page 1-167

Use Custom OFDM Sample Rate and Custom FFT Size

This example shows how to set custom values for the sample rate input, `SampleRate`, and the fast Fourier transform (FFT) size input, `Nfft`, when you call an OFDM function (`nrOFDMModulate`, `nrOFDMInfo`, or `nrOFDMDemodulate`).

Custom OFDM Sample Rate

The value that you set for the `SampleRate` input determines the sample rate of the waveform.

The nominal sample rate corresponding to the FFT size used in the OFDM modulation, FFT_{SR}, is equal to `Nfft*carrier.SubcarrierSpacing*1000`, where `carrier` is the input argument of the function call that specifies the carrier configuration.

Because the resampling of the OFDM-modulated waveform is by a factor of `SampleRate`/FFT_{SR}, the resampling is costly if `SampleRate` and FFT_{SR} do not have large common factors.

Custom FFT Size

The value that you set for the `Nfft` input must satisfy these conditions.

- `Nfft` is an integer (to ensure integer-valued cyclic prefix lengths).
- `Nfft` is a power of 2.
- `Nfft` results in a maximum occupancy of 100%. The actual occupancy is equal to `carrier.NSizeGrid*12/Nfft`.

You can only achieve a bandwidth occupancy of exactly 100% when these conditions apply.

- The FFT and the carrier grid have the same size. That is, `Nfft` is `carrier.NSizeGrid*12`.
- Resampling is not needed. That is, `SampleRate` is `Nfft*carrier.SubcarrierSpacing*1000`.

Plot Bandwidth Occupancy

Create a carrier configuration object.

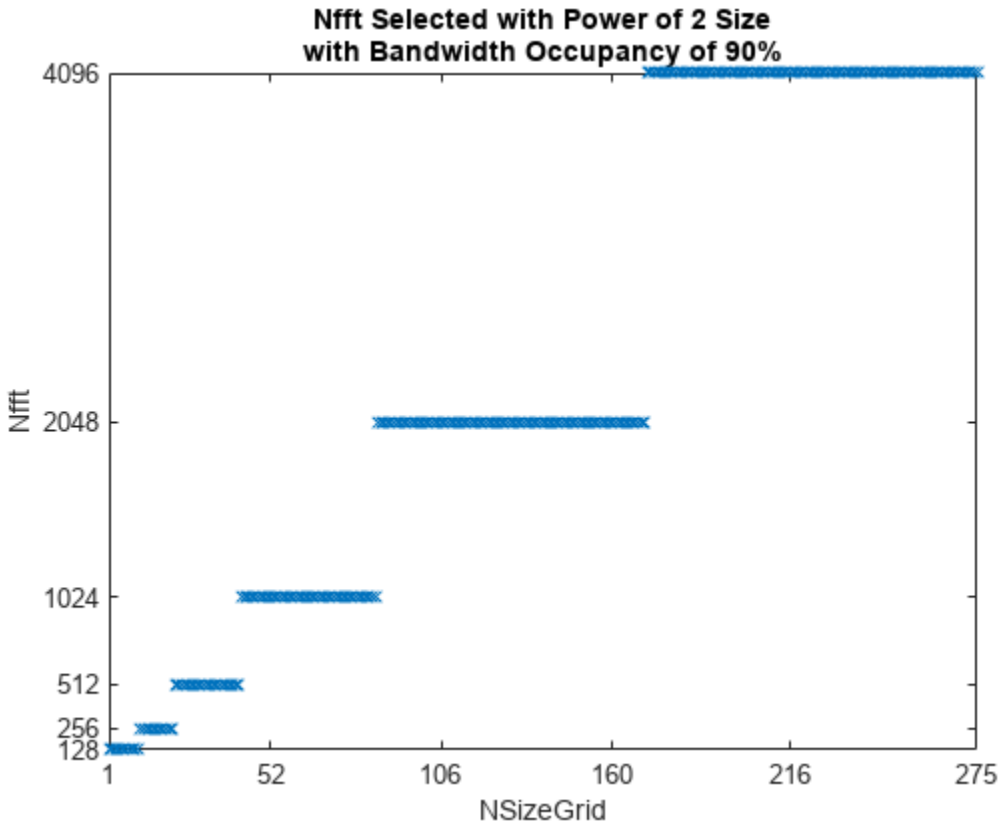
```
carrier = nrCarrierConfig;
```

Set the `SampleRate` for each `NSizeGrid` such that the bandwidth occupancy (`txBW/SampleRate`) is 90%. Set `Nfft` to a power of 2.

```
nSizeGrids = 1:275;
userSampleRateUserNfft = zeros(1,275);
fftOccupancy = zeros(1,275);
for nSizeGrid = 1:275
    carrier.NSizeGrid = nSizeGrid;
    % Transmission bandwidth of OFDM waveform
    txBW = carrier.NSizeGrid*12*carrier.SubcarrierSpacing*1000;
    sr = txBW / 0.9;
    nfft = max(128,2^ceil(log2(carrier.NSizeGrid*12)));
    ofdmInfo = nrOFDMInfo(carrier,'SampleRate',sr,'Nfft',nfft);
    userSampleRateUserNfft(nSizeGrid) = ofdmInfo.Nfft;
    fftOccupancy(nSizeGrid) = carrier.NSizeGrid*12/ofdmInfo.Nfft;
end
```

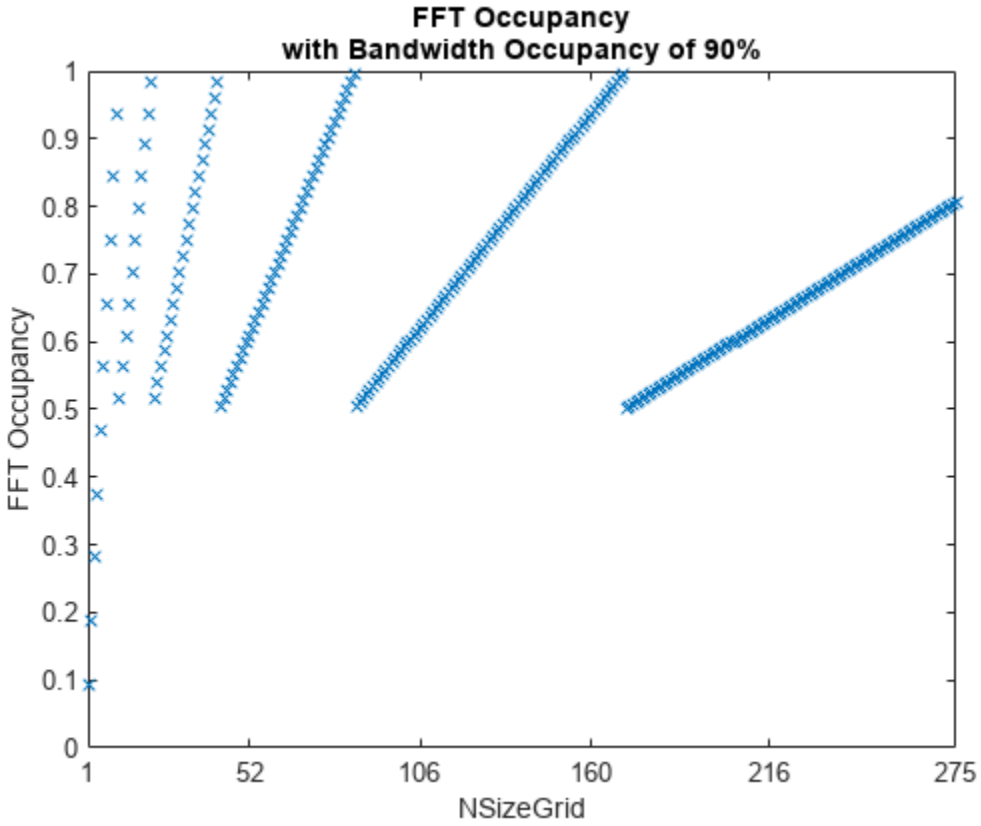
Plot the resulting FFT size.

```
figure;
plot(nSizeGrids,userSampleRateUserNfft,'x');
title({'Nfft Selected with Power of 2 Size' 'with Bandwidth Occupancy of 90%'});
axis([1 275 min(userSampleRateUserNfft) max(userSampleRateUserNfft)]);
xlabel('NSizeGrid');
xticks([1 52 106 160 216 275]);
ylabel('Nfft');
yticks(2.^(7:12));
```



Plot the resulting FFT occupancy.

```
figure;
plot(nSizeGrids,fftOccupancy,'x');
title({'FFT Occupancy' 'with Bandwidth Occupancy of 90%'});
axis([1 275 0 1]);
xlabel('NSizeGrid');
xticks([1 52 106 160 216 275]);
ylabel('FFT Occupancy');
```



Get OFDM Information

Update the carrier to 25 resource blocks (RBs).

```
carrier.NSizeGrid = 25;
```

Get OFDM information.

```
ofdmInfo = nrOFDMInfo(carrier)
```

```
ofdmInfo = struct with fields:
                    Nfft: 512
              SampleRate: 7680000
    CyclicPrefixLengths: [40 36 36 36 36 36 36 40 36 36 36 36 36 36]
          SymbolLengths: [552 548 548 548 548 548 548 552 548 548 548 548 548 548]
              Windowing: 18
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         SymbolsPerSlot: 14
        SlotsPerSubframe: 1
           SlotsPerFrame: 10
```

The `CyclicPrefixLengths` and `SymbolLengths` fields in the output structure return the cyclic prefix lengths and total OFDM symbol lengths for each OFDM symbol in a subframe. The total OFDM symbol length is composed of the cyclic prefix and the nominal OFDM symbol length equal to the FFT size.

```
ofdmInfo.SymbolLengths - ofdmInfo.CyclicPrefixLengths
```

```
ans = 1×14

   512   512   512   512   512   512   512   512   512   512   512   512   512   512
```

Get OFDM information for a specified FFT size. Because OFDM symbol construction is performed using an IFFT of size that you specify in the `Nfft` input, the `CyclicPrefixLengths` and `SymbolLengths` fields return values in terms of this `Nfft` value. The specified `Nfft` value is also returned in the `Nfft` field of the output.

```
nfft = 640;
ofdmInfo = nrOFDMInfo(carrier,'Nfft',nfft)
```

```
ofdmInfo = struct with fields:
                    Nfft: 640
              SampleRate: 9600000
    CyclicPrefixLengths: [50 45 45 45 45 45 45 50 45 45 45 45 45 45]
          SymbolLengths: [690 685 685 685 685 685 685 690 685 685 685 685 685 685]
              Windowing: 22
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         SymbolsPerSlot: 14
        SlotsPerSubframe: 1
           SlotsPerFrame: 10
```

```
ofdmInfo.SymbolLengths - ofdmInfo.CyclicPrefixLengths
```

```
ans = 1×14

   640   640   640   640   640   640   640   640   640   640   640   640   640   640
```

Get OFDM information for a specified FFT size and sample rate. For the default sample rate, the number of time-domain samples of each OFDM symbol matches the values of the `SymbolLengths` field. If you specify the `SampleRate` input, the specified value is returned in the `SampleRate` field of the output. However, because of the `CyclicPrefixLengths` and `SymbolLengths` fields are expressed in terms of the IFFT size used during OFDM symbol construction, these values do not change in the output. The waveform is resampled to the sample rate that you specify after OFDM symbol construction. Depending on the sample rate, the length of the cyclic prefixes and nominal OFDM symbols in the corresponding OFDM waveform might not be an integer number of samples.

```
sr = 1.35 * nfft * carrier.SubcarrierSpacing * 1e3;
ofdmInfo = nrOFDMInfo(carrier,'Nfft',640,'SampleRate',sr)
```

```
ofdmInfo = struct with fields:
                    Nfft: 640
              SampleRate: 12960000
    CyclicPrefixLengths: [50 45 45 45 45 45 45 50 45 45 45 45 45 45]
          SymbolLengths: [690 685 685 685 685 685 685 690 685 685 685 685 685 685]
              Windowing: 22
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         SymbolsPerSlot: 14
        SlotsPerSubframe: 1
           SlotsPerFrame: 10
```

```
ratio = ofdmInfo.SampleRate / (ofdmInfo.Nfft * carrier.SubcarrierSpacing * 1e3)
```

```
ratio = 1.3500
```

```
disp(num2str(ofdmInfo.SymbolLengths*ratio,'%0.3f '));
```

```
931.500 924.750 924.750 924.750 924.750 924.750 924.750 931.500 924.750 924.750 924.750 924.750 9
```

See Also

Functions
`nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

More About

- “Resampling Filter Design in OFDM Functions” on page 1-182
- “Configure OFDM Sample Rate and FFT Size” on page 1-167

Resampling Filter Design in OFDM Functions

This example shows the internal resampling filter design of the OFDM functions `nrOFDMModulate`, `nrOFDMInfo`, and `nrOFDMDemodulate`. These functions enable you to specify the OFDM sample rate and fast Fourier transform (FFT) size by using the `SampleRate` and `Nfft` name-value input arguments. When the specified sample rate and the nominal sample rate, corresponding to the FFT size, do not match, the OFDM functions resample the waveform using an internal multirate FIR filter.

OFDM Sample Rate and Nominal Sample Rate

The value that you set for the `SampleRate` input determines the sample rate of the waveform.

The nominal sample rate corresponding to the FFT size used in the OFDM modulation, FFT_{SR}, is equal to `Nfft*carrier.SubcarrierSpacing*1000`, where `carrier` is the input argument of the function call that specifies the carrier configuration.

When the specified sample rate and the nominal sample rate do not match, the OFDM functions resample the waveform using an internal resampling filter. Setting the sample rate to the nominal sample rate does not result in resampling during OFDM modulation.

OFDM Modulation without Resampling

Create a carrier configuration object with 25 resource blocks.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 25;
```

Get OFDM information.

```
ofdmInfo = nrOFDMInfo(carrier);
```

Display nominal sample rate and the sample rate returned in the OFDM information.

```
ofdmInfo.Nfft*carrier.SubcarrierSpacing*1000
```

```
ans = 7680000
```

```
ofdmInfo.SampleRate
```

```
ans = 7680000
```

Verify that specifying a sample rate of `7.68e6` does not result in resampling during OFDM modulation.

```
grid = nrResourceGrid(carrier);
grid(:) = nrSymbolModulate(randi([0 1],numel(grid)*2,1),'QPSK');
waveform1 = nrOFDMModulate(carrier,grid);
waveform2 = nrOFDMModulate(carrier,grid,'SampleRate',7.68e6);
isequal(waveform1,waveform2)
```

```
ans = logical
   1
```

Filter Design

The resampling filter is a function of these two ratios.

- The ratio of the transmission bandwidth (TX_{BW}) to the nominal sample rate: TX_{BW} / FFT_{SR}, where TX_{BW} is equal to `carrier.NSizeGrid*12*carrier.SubcarrierSpacing*1000`.
- The ratio of the nominal sample rate to the specified sample rate: FFT_{SR} / `SampleRate`.

To design the resampling filter, the OFDM functions call the `designMultirateFIR` function with these input arguments.

- Interpolation factor, `L`, equal to `SampleRate/g`, where `g = gcd(`FFT_{SR}`,SampleRate)`. Because the resampling of the OFDM-modulated waveform is by a factor of `SampleRate/`FFT_{SR}, the resampling is costly if `SampleRate` and FFT_{SR} do not have large common factors.
- Decimation factor, `M`, equal to FFT_{SR}`/g`.
- Transition width, `TW`, such that the transition band starts at ±TX_{BW} / 2 (that is, the transition band starts at the edge of the occupied bandwidth).
- Stopband attenuation, `Astop`, equal to 70 dB.

If `SampleRate` > FFT_{SR}, then:

- `L` > `M` and the transition band stops at ±FFT_{SR} / 2 (corresponding to a normalized frequency of 1/`L`).
- The filter acts as an anti-imaging filter after upsampling by `L`.

If `SampleRate` < FFT_{SR}, then:

- `M` > `L` and the transition band stops at `±SampleRate/2` (corresponding to a normalized frequency of 1/`M`).
- The filter acts as an anti-aliasing filter before downsampling by `M`.

OFDM Modulation with Resampling

Create a carrier configuration object with 25 resource blocks.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 25;
```

Compute the transmission bandwidth.

```
txBW = carrier.NSizeGrid*12*carrier.SubcarrierSpacing*1000
```

```
txBW = 4500000
```

Specify a custom sample rate. Obtain the corresponding OFDM information and calculate the nominal sample rate for the default FFT size.

```
SampleRate = 6e6;
ofdmInfo = nrOFDMInfo(carrier,'SampleRate',SampleRate);
Nfft = ofdmInfo.Nfft
```

```
Nfft = 640
```

```
fftSR = Nfft*carrier.SubcarrierSpacing*1000
```

```
fftSR = 9600000
```

Calculate `g`, `L`, and `M`.

```
g = gcd(fftSR,SampleRate)
```

```
g = 1200000
```

```
L = SampleRate/g
```

```
L = 5
```

```
M = fftSR/g
```

```
M = 8
```

Plot the spectrum of the OFDM waveform before resampling by setting the sample rate to the nominal sample rate.

```
figure;
hold on;
[f,S] = measureSpectrum(carrier,Nfft,fftSR,1);
plot(f,S,'LineWidth',2);
```

Plot the spectrum of the OFDM waveform after upsampling by `L`, before applying the filter.

```
[fL,S] = measureSpectrum(carrier,Nfft,fftSR,L);
hL = plot(fL,S,':');
```

Plot the spectrum of the OFDM waveform after applying the filter.

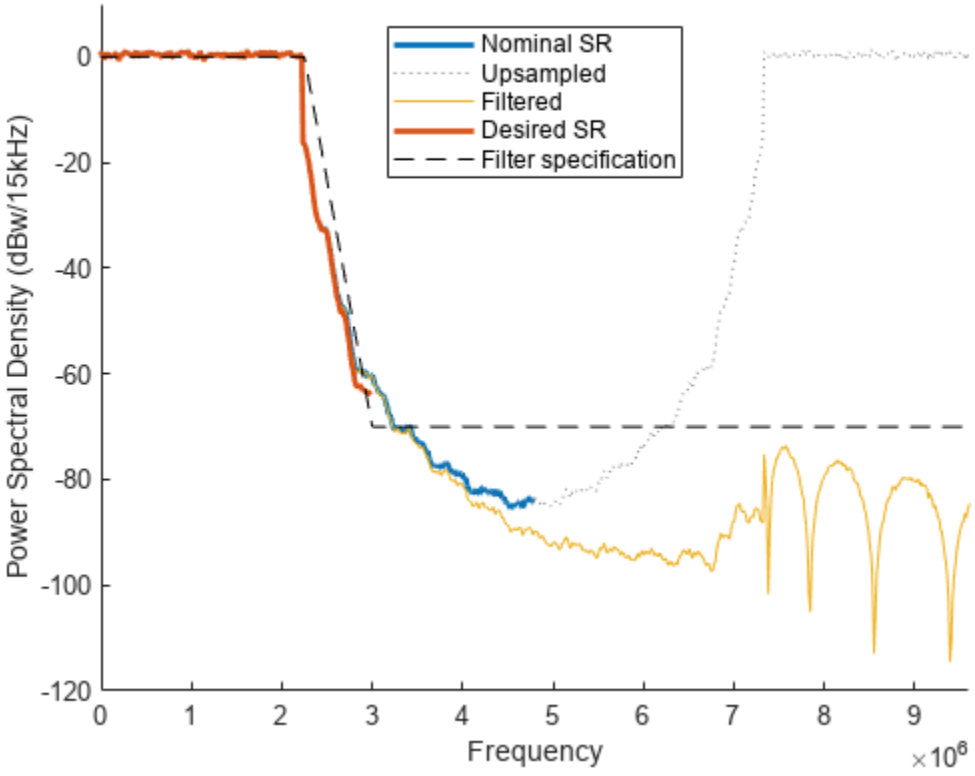
```
[f,S] = measureSpectrum(carrier,Nfft,fftSR*L,1);
plot(f,S);
```

Plot the spectrum of the final OFDM waveform after downsampling.

```
[f,S] = measureSpectrum(carrier,Nfft,SampleRate,1);
plot(f,S,'Color',hL.Color,'LineWidth',2);
hL.Color = [0.5 0.5 0.5];
```

Overlay the filter design specification.

```
fStop = fftSR/2*L/max([L M]);
aStop = -70.0;
spec = [fL(1) aStop; -fStop aStop; -txBW/2 0; txBW/2 0; fStop aStop; fL(end) aStop];
plot(spec(:,1),spec(:,2),'k--');
legend('Nominal SR','Upsampled','Filtered','Desired SR','Filter specification','Location','north
xlabel('Frequency');
ylabel(['Power Spectral Density (dBw/' num2str(carrier.SubcarrierSpacing) 'kHz)']);
axis([0 fftSR -120 10]);
```



Local Function

```
function [f,S] = measureSpectrum(carrier,Nfft,SampleRate,L)

    % Subcarrier spacing in Hz
    SCS = carrier.SubcarrierSpacing*1e3;

    % Set up spectrum estimator
    spectrumEstimator = dsp.SpectrumEstimator;
    spectrumEstimator.SampleRate = SampleRate;
    spectrumEstimator.AveragingMethod = 'Exponential';
    spectrumEstimator.ForgettingFactor = 0.99;
    spectrumEstimator.FrequencyRange = 'centered';
    spectrumEstimator.PowerUnits = 'dBW';
    spectrumEstimator.SpectrumType = 'Power density';
    spectrumEstimator.FFTLengthSource = 'Property';
    spectrumEstimator.FFTLength = floor(SampleRate*L/SCS);

    % For 100 slots
    rs = RandStream('mt19937ar','Seed',1);
    for nSlot = 0:99

        % Create a slot grid filled with QPSK symbols and OFDM modulate
        grid = nrResourceGrid(carrier);
        grid(:) = nrSymbolModulate(rs.randi([0 1],numel(grid)*2,1),'QPSK');
        waveform = nrOFDMModulate(carrier,grid,'Nfft',Nfft,'SampleRate',SampleRate);
        waveform = waveform * Nfft;
```

```
        % Apply interpolation if required
        if (L>1)
            T = size(waveform,1);
            waveform = reshape([waveform.'; zeros(L-1,T)],[],1)*sqrt(L);
        end

        % Measure the spectrum
        S = spectrumEstimator(waveform);

    end

    % Create frequency axis
    N = spectrumEstimator.FFTLength;
    f = (-N/2:(N/2 - 1))*SCS;

    % Translate from dBw/Hz to dBW/SCS
    S = S + 10*log10(SCS);

end
```

See Also

Functions
`nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

More About

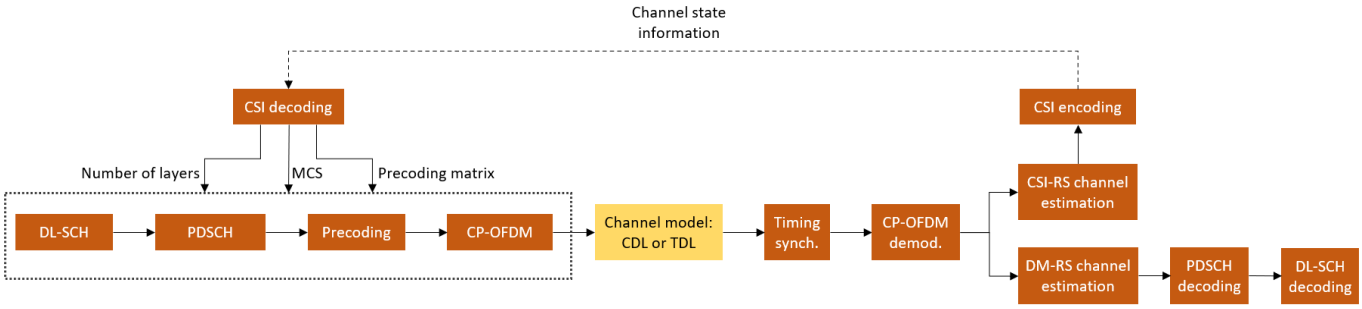
- “Configure OFDM Sample Rate and FFT Size” on page 1-167

NR PDSCH Throughput Using Channel State Information Feedback

This reference simulation measures the physical downlink shared channel (PDSCH) throughput of a 5G new radio (NR) link. The simulation uses channel state information (CSI) feedback to adjust these downlink shared channel (DL-SCH) and PDSCH transmission parameters: target code rate, modulation, number of layers, and multiple-input multiple-output (MIMO) precoding matrix. In addition, this simulation enables you to explore CSI compression techniques beyond 5G NR based on artificial intelligence (AI).

Introduction

This figure shows the steps of PDSCH throughput estimation in this reference simulation. The transmitter (gNodeB) includes transport channel coding stages (DL-SCH) and PDSCH generation. The transmission parameters are based on the CSI measurements that the receiver (UE) performs and feeds back to the transmitter. The aim of this mechanism is to adapt transmission parameters to the channel conditions. The simulation also controls the CSI feedback delay.



This simulation is based on the “NR PDSCH Throughput” on page 1-58 example with the addition of using CSI feedback from the receiver to adjust transmission parameters.

CSI Feedback and Link Adaptation in 5G NR

User equipment (UE) uses channel state information reference signals (CSI-RS) to estimate the downlink channel response and reports a set of CSI indicators. The CSI report includes these indicators, as defined in TS 38.214 Section 5.2.2:

- Rank indicator (RI)
- Precoding matrix indicator (PMI)
- Channel quality indicator (CQI)

The gNodeB uses the reported CSI to configure the target code rate, modulation, number of layers, and MIMO precoding matrix in subsequent PDSCH transmissions after a configurable delay. Generally, the gNodeB can select different transmission parameters from those a UE reports. However, in this simulation, the gNodeB strictly follows the recommendation from the UE. For more information on how the UE selects RI, PMI, and CQI, see the “5G NR Downlink CSI Reporting” on page 5-47 example.

CSI Feedback and Link Adaptation Beyond 5G NR

You can use this simulation to explore other CSI techniques, such as the one described in "CSI Feedback with Autoencoders" on page 5-91. You can configure the UE to use an autoencoder AI neural network to encode channel estimates and feed back this form of CSI to the gNodeB. The gNodeB decodes this compressed CSI and adapts the target code rate, modulation, number of layers, and MIMO precoding matrix. The gNodeB selects the MIMO precoding matrix from the eigenvectors of the recovered channel estimates and a suitable modulation and coding scheme (MCS) from TS 38.214 Table 5.1.3.1-1, conditioned on the selected MIMO precoder. Finally, it applies the selected configuration in a subsequent PDSCH transmission after a configurable delay.

The simulation assumes these conditions for both forms of CSI feedback:

- No uplink transmissions. The CSI feedback is transmitted without errors from the UE to the gNodeB.
- Fixed PDSCH PRB allocation for the length of the simulation. No scheduler exists to adapt the allocation to the channel conditions.
- FDD operation.
- No HARQ support.

Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10 ms frames. To produce statistically meaningful throughput results, you must update the number of frames (`NFrames`) to a large number. Set the signal-to-noise ratio (SNR) points to simulate. The SNR for each layer is defined per resource element (RE) and includes the effect of signal and noise across all antennas. For an explanation of the SNR definition that this example uses, see the "SNR Definition Used in Link Simulations" on page 5-86 example.

```
simParameters = struct();       % Clear simParameters variable to contain all key simulation para
simParameters.NFrames = 2;      % Number of 10 ms frames
simParameters.SNRIn = [-10 10]; % SNR range (dB)
```

Channel Estimator Configuration

The logical variable `PerfectChannelEstimator` controls channel estimation and synchronization behavior. When this variable is set to `true`, the simulation uses perfect channel estimation and synchronization. When this variable is set to `false`, the simulation uses practical channel estimation and synchronization based on the values of the received PDSCH demodulation reference signal (DM-RS).

```
simParameters.PerfectChannelEstimator = ;
```

Simulation Diagnostics

The variable `DisplaySimulationInformation` controls the display of simulation information.

```
simParameters.DisplaySimulationInformation = ;
```

The `DisplayDiagnostics` flag enables the plotting of the error vector magnitude (EVM) per layer. This plot monitors the quality of the received signal after equalization. The EVM per layer figure shows:

- The EVM per layer per slot, which shows the EVM evolving with time.
- The EVM per layer per resource block, which shows the EVM in frequency.

This figure evolves with the simulation and is updated with each slot. Typically, low SNR or channel fades can result in decreased signal quality (high EVM). The channel affects each layer differently. Therefore, the EVM values can differ across layers.

```
simParameters.DisplayDiagnostics = ;
```

Carrier and PDSCH Configuration

Set waveform type, PDSCH numerology (subcarrier spacing (SCS) and cyclic prefix (CP) type), and other transmission configuration parameters.

```
% SCS carrier parameters
simParameters.Carrier = nrCarrierConfig;          % Carrier resource grid configuration
simParameters.Carrier.NSizeGrid = 52;             % Bandwidth in number of resource blocks
simParameters.Carrier.SubcarrierSpacing = 15;     % 15, 30, 60, 120 (kHz)
simParameters.Carrier.CyclicPrefix = 'Normal';    % 'Normal' or 'Extended' (Extended CP is relevar
simParameters.Carrier.NCellID = 1;                % Cell identity
```

Specify a full-band slot-wise PDSCH transmission and reserve the first 2 OFDM symbols.

```
simParameters.PDSCH = nrPDSCHConfig;      % This PDSCH definition is the basis for all PDSCH tran

% Define PDSCH time allocation in a slot
simParameters.PDSCH.MappingType = Type A ; % PDSCH mapping type ('A'(slot-wise),'B
symAlloc = [2 simParameters.Carrier.SymbolsPerSlot-2];  % Starting symbol and number of symbols
simParameters.PDSCH.SymbolAllocation = symAlloc;

% Define PDSCH frequency resource allocation per slot to be full grid
simParameters.PDSCH.PRBSet = 0:simParameters.Carrier.NSizeGrid-1;

% Scrambling identifiers
simParameters.PDSCH.NID = simParameters.Carrier.NCellID;
simParameters.PDSCH.RNTI = 1;
```

Specify the DM-RS configuration, as defined in TS 38.211 Section 7.4.1.1. The DM-RS port set is dynamic and based on CSI feedback (RI).

```
simParameters.PDSCH.DMRS.DMRSTypeAPosition       = 2 ; % Mapping type A only.
simParameters.PDSCH.DMRS.DMRSLength              = 1 ; % Number of front-load
simParameters.PDSCH.DMRS.DMRSAdditionalPosition  = 2 ; % Additional DM-RS sym
simParameters.PDSCH.DMRS.DMRSConfigurationType   = 2 ; % DM-RS configuration
simParameters.PDSCH.DMRS.NumCDMGroupsWithoutData = 3 ; % CDM groups without d
```

Specify additional simulation parameters for the DL-SCH and PDSCH. Set the frequency granularity of the MIMO precoder, as defined in TS 38.214 Section 5.1.2.3, the modulation and coding scheme table, as defined in TS 38.214 Section 5.1.3.1, and the overhead for transport block size calculation. Set `XOverhead` to empty (`[]`) for automatic selection based on the CSI-RS allocated REs.

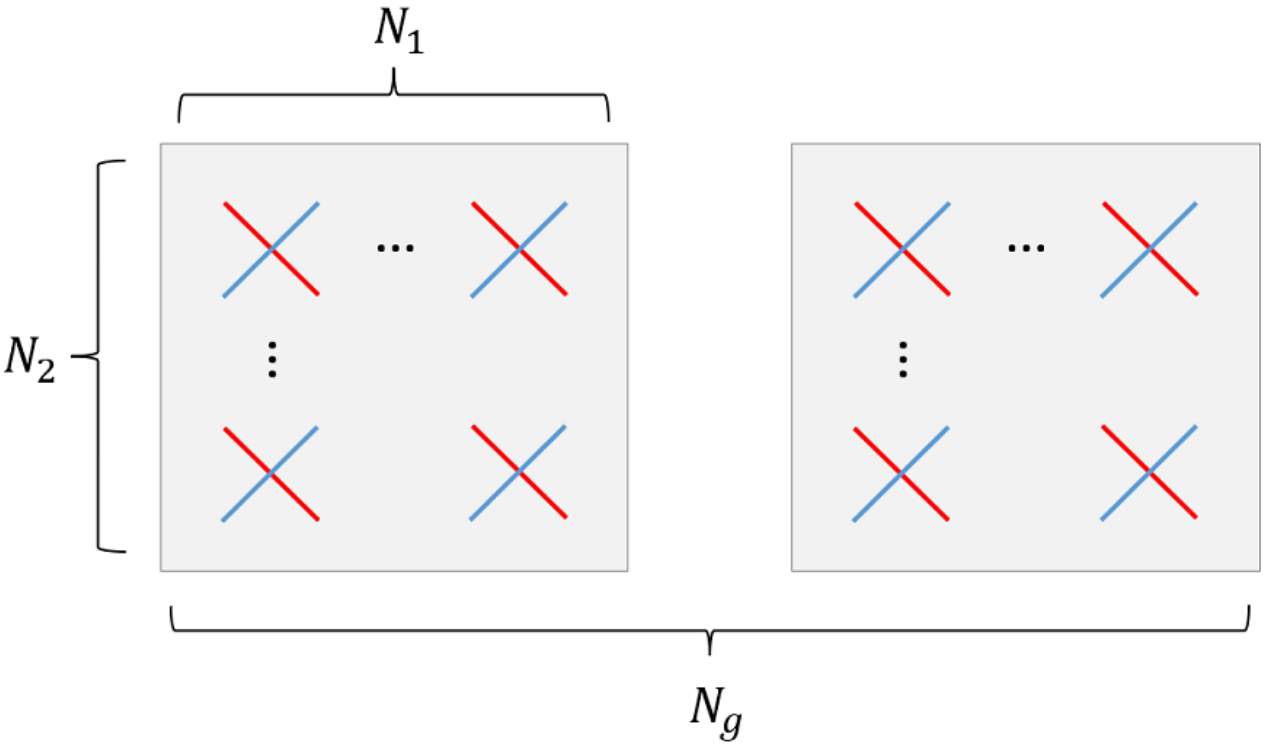
```
simParameters.PDSCHExtension = struct();
simParameters.PDSCHExtension.PRGBundleSize = 4; % 2, 4, or [] to signify "wi
simParameters.PDSCHExtension.MCSTable      = Table 1; % 'Table1',...,'Table4'
simParameters.PDSCHExtension.XOverhead     = []; % 0, 6, 12, 18, or [] for au
```

Configure the LDPC decoding algorithm and maximum LDPC iterations. The available decoding algorithms are 'Belief propagation', 'Layered belief propagation', 'Normalized min-sum', and 'Offset min-sum'.

```
simParameters.PDSCHExtension.LDPCDecodingAlgorithm = Normalized min-sum;
simParameters.PDSCHExtension.MaximumLDPCIterationCount = 6;
```

Antenna Panel Configuration

Configure the antenna panel geometry by specifying the number of antenna panels (N_g) and their dimensions (N_1 and N_2). N_1 and N_2 are the number of cross-polarized antenna elements in horizontal and vertical directions for each panel, respectively, and N_g is the number of column array panels. This diagram shows an antenna panel arrangement and its dimensions.



The MIMO precoding codebooks defined in TS 38.214 Section 5.2.2.2 restrict the values of N_g, N_1, and N_2 that 5G NR supports. To configure single-input single-output (SISO) transmissions with single-polarized antenna elements, set the panel dimensions to [1 1]. The receiver panel dimensions are not limited by the codebooks.

```
simParameters.TransmitAntennaArray.NumPanels        = 1;      % Number of panels
simParameters.TransmitAntennaArray.PanelDimensions  = [2 2];  % Number of columns
simParameters.ReceiveAntennaArray.NumPanels         = 1;      % Number of panels
simParameters.ReceiveAntennaArray.PanelDimensions   = [2 1];  % Number of columns a
```

Due to polarization, the overall number of antennas is $2 \times N_1 \times N_2 \times N_g$.

```
simParameters.NTxAnts = numAntennaElements(simParameters.TransmitAntennaArray);
simParameters.NRxAnts = numAntennaElements(simParameters.ReceiveAntennaArray);
```

CSI-RS Configuration

The simulation configures an 8-port CSI-RS, which maps to each antenna element of the transmit panel. The number of CSI-RS ports must match the transmit array panel's dimensions, as defined in TS 38.214 Table 5.2.2.2.1-2 for single-panel and Table 5.2.2.2.2-1 for multi-panel codebooks. These tables show what CSI-RS row numbers are compatible with the selected array panel dimensions.

Single panel

(N1,N2)	CSI-RS row numbers	Number of CSI-RS ports
(2,1)	(4,5)	4
(2,2)	(6,7,8)	8
(4,1)		
(3,2)	(9,10)	12
(6,1)		
(4,2)	(11,12)	16
(8,1)		
(4,3)	(13,14,15)	24
(6,2)		
(12,1)		
(4,4)	(16,17,18)	32
(8,2)		
(16,1)		

Multipanel

(Ng,N1,N2)	CSI-RS row numbers	Number of CSI-RS ports
(2,2,1)	(6,7,8)	8
(2,4,1)	(11,12)	16
(4,2,2)		
(2,2,2)		
(2,8,1)	(16,17,18)	32
(4,4,1)		
(2,4,2)		
(4,2,2)		

Configure the CSI-RS transmission parameters.

```
simParameters.CSIRS = nrCSIRSConfig;
simParameters.CSIRS.CSIRSType = Non-zero power; % 'nzp','zp'
simParameters.CSIRS.RowNumber = 6; % 1...18
simParameters.CSIRS.NumRB = simParameters.Carrier.NSizeGrid - simParameters.CSIRS.RBOffset;
simParameters.CSIRS.CSIRSPeriod = [10 0];
simParameters.CSIRS.SymbolLocations = 4;
simParameters.CSIRS.SubcarrierLocations = [0,3,6,9];
simParameters.CSIRS.Density = 'one';

disp(['Number of CSI-RS ports: ' num2str(simParameters.CSIRS.NumCSIRSPorts) '.'])

Number of CSI-RS ports: 8.

csirsCDMLengths = getCSIRSCDMLengths(simParameters.CSIRS);

% Check that the number of CSI-RS ports and transmit antenna elements match
% and the consistency of multiple CSI-RS resources
validateCSIRSConfig(simParameters.Carrier,simParameters.CSIRS,simParameters.NTxAnts);
```

CSI Feedback Configuration

Specify the CSI feedback mode as one of 'RI-PMI-CQI', 'AI CSI compression', or 'Perfect CSI'. In all modes, the UE can perform perfect or practical channel estimation using CSI-RS.

- **RI-PMI-CQI**: The UE selects and reports the appropriate RI, PMI, and CQI periodically. The gNodeB follows the reported CSI to configure the target code rate, modulation, number of layers, and MIMO precoding matrix in subsequent transmissions after a configurable delay.
- **AI CSI compression**: The UE compresses channel estimates using an AI neural network autoencoder and reports them to the gNodeB periodically. The gNodeB recovers the channel estimates and selects appropriate target code rate, modulation, number of layers, and MIMO precoding matrix. The gNodeB applies the selected configuration in a subsequent PDSCH transmission after a configurable delay. You can specify the filename of the neural network. This simulation downloads a pretrained neural network that is valid only when the number of transmit antennas is 8 and a carrier grid size of 52 RB and 15 kHz subcarrier spacing. For other number of transmit antennas and bandwidths, you need to train the neural network as described in “CSI Feedback with Autoencoders” on page 5-91.
- **Perfect CSI**: The UE reports channel estimates to the gNodeB periodically. This mode is equivalent to the 'AI CSI compression' mode without compression losses.

```
simParameters.CSIReportMode = RI-PMI-CQI; % 'RI-PMI-CQI','AI CSI compression','Perfec
```

Specify the CSI report configuration. For more information on the CSI report configuration, see the “5G NR Downlink CSI Reporting” on page 5-47 example.

```
simParameters.CSIReportConfig = struct();
simParameters.CSIReportConfig.Period = [5 0];  % Peridocity and offset of the CSI report in slots

if simParameters.CSIReportMode == "RI-PMI-CQI"

    simParameters.CSIReportConfig.CQITable = Table 1; % 'Table1','Table2'
    simParameters.CSIReportConfig.CQIMode = Wideband; % 'Wideband','Subba
```

```
    simParameters.CSIReportConfig.PMIMode           = Subband; % 'Wideband','Subba
    simParameters.CSIReportConfig.CodebookType      = Type1SinglePanel; % 'Type1SinglePanel
    simParameters.CSIReportConfig.SubbandSize       = 4; % Subband size in R
    simParameters.CSIReportConfig.CodebookMode      = 1; % 1,2
    simParameters.CSIReportConfig.RIRestriction     = [];                % Empty for no rank
    simParameters.CSIReportConfig.NumberOfBeams     = 2; % 2,3,4. Only for T
    simParameters.CSIReportConfig.PhaseAlphabetSize = 8; % 4,8. Only for Typ
    simParameters.CSIReportConfig.SubbandAmplitude  = ;                  % true/false. Only fo
    simParameters.CSIReportConfig.NStartBWP         = [];                % Empty to signal the
    simParameters.CSIReportConfig.NSizeBWP          = [];                % Empty to signal the

    % Configure the CSI report with the antenna panel dimensions specified
    simParameters.CSIReportConfig.PanelDimensions = getCSIReportPanelDimensions(simParameters.Tra

else % AI CSI compression

    % Specify the file name of the AI neural network
    simParameters.AINetworkFilename = 'csiTrainedNetwork.mat';

end
```

Configure the CSI processing delay in slots. For the UE, this delay is the number of time slots between the reception of the CSI-RS and the availability of the CSI feedback. For the gNodeB, the delay is the number of time slots between the reception of the CSI report and the transmission using the recommended CSI.

```
simParameters.UEProcessingDelay = 7;
simParameters.BSProcessingDelay = 1;
```

Propagation Channel Configuration

Configure the delay profile, delay spread, and maximum Doppler shift of the propagation channel for the simulation. Both CDL and TDL channel models are supported. The panel dimensions and cross-polarized elements specified in Antenna Panel Configuration on page 1-190 define the geometry of the antenna arrays for CDL channels.

```
simParameters.DelayProfile = 'CDL-C';   % 'CDL-A',...,'CDL-E','TDL-A',...,'TDL-E'
simParameters.DelaySpread = 300e-9;     % s
simParameters.MaximumDopplerShift = 5;  % Hz

simParameters.Channel = createChannel(simParameters);
```

Processing Loop

To determine the PDSCH throughput for each SNR point, the simulation performs these steps:

1. Update DL-SCH and PDSCH transmission parameters (target code rate, number of layers, modulation, and MIMO precoding matrix). This step applies only when a new CSI report is available.
2. Map CSI-RS signals to the resource grid.

3. Perform channel coding (DL-SCH) and PDSCH encoding. Map PDSCH and PDSCH DM-RS to the resource grid.
4. OFDM modulate the generated grid.
5. Pass the signal through a fading channel with AWGN.
6. Perform synchronization and OFDM demodulation.
7. Perform PDSCH DM-RS based channel estimation.
8. Perform equalization.
9. Decode the PDSCH and DL-SCH, and measure the PDSCH throughput.
10. Perform CSI-RS based channel estimation.
11. Create CSI report.
12. Feed the CSI report back to the transmitter with the appropriate delay.

To reduce the total simulation time, you can execute the SNR loop in parallel by using the Parallel Computing Toolbox™. Comment out the `for` statement and uncomment the `parfor` statement. If the Parallel Computing Toolbox is not installed, `parfor` defaults to normal `for` statement. Because parfor-loop iterations are executed in parallel in a nondeterministic order, the simulation information displayed for each SNR point can be intertwined. To switch off simulation information display, set the `displaySimulationInformation` variable above to false.

```
% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(simParameters.SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(simParameters.SNRIn),1);

% Cell array to store CSI reports per SNR point
CSIReport = {};

for snrIdx = 1:numel(simParameters.SNRIn)
% parfor snrIdx = 1:numel(simParameters.SNRIn)
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'
% statement and uncomment the 'parfor' statement.

    % Reset the random number generator for repeatability
    rng(0,"twister");

    % Display simulation information at this SNR point
    displaySNRPointProgress(simParameters,snrIdx);

    % Take full copies of the simulation-level parameter structures so that
    % they are not PCT broadcast variables when using parfor
    simParamLocal = simParameters;

    % Set up the transmitter, propagation channel, receiver, and CSI
    % feedback configuration parameters.
    [carrier,encodeDLSCH,pdsch,pdschextra,csirs,wtx] = setupTransmitter(simParamLocal);
    [channel,maxChDelay,N0] = setupChannel(simParamLocal,snrIdx);
    [decodeDLSCH,pathFilters,timingOffset] = setupReceiver(simParamLocal);
    csiFeedbackOpts = getCSIFeedbackOptions(simParamLocal,snrIdx);

    % Obtain an initial CSI report based on perfect channel estimates that
    % the Tx can use to adapt the transmission parameters.
    csiReports = initialCSIReport(simParamLocal,snrIdx,carrier,csirs,channel,csiFeedbackOpts);
```

```
csiAvailableSlots = 0;

% Total number of slots in the simulation period
NSlots = simParamLocal.NFrames * carrier.SlotsPerFrame;

% Loop over the entire waveform length
for nslot = 0:NSlots-1

    % Update the carrier slot numbers for new slot
    carrier.NSlot = nslot;

    % Use new CSI report to configure the number of layers and
    % modulation of the PDSCH and target code rate of the DL-SCH if
    % there is a new report available.
    [isNewReport,repIdx] = ismember(nslot,csiAvailableSlots);
    if isNewReport
        [pdsch.Modulation,pdschextra.TargetCodeRate,wtx] = hCSIDecode(carrier,pdsch,pdschextr
        pdsch.NumLayers = size(wtx,1);
        encodeDLSCH.TargetCodeRate = pdschextra.TargetCodeRate;
    end

    % Create an OFDM resource grid for a slot
    dlGrid = nrResourceGrid(carrier,csirs.NumCSIRSPorts);

    % CSI-RS mapping to the slot resource grid
    [csirsInd,csirsInfo] = nrCSIRSIndices(carrier,csirs);
    csirsSym = nrCSIRS(carrier,csirs);
    dlGrid(csirsInd) = csirsSym;
    csirsTransmission = ~isempty(csirsInd);

    % PDSCH reserved REs for CSI-RS
    pdsch.ReservedRE = csirsInd-1; % 0-based indices

    % PDSCH generation
    % Calculate the transport block sizes for the transmission in the slot
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);
    trBlkSizes = nrTBS(pdsch.Modulation,pdsch.NumLayers,numel(pdsch.PRBSet),pdschIndicesInfo

    % Transport block generation
    for cwIdx = 1:pdsch.NumCodewords
        % New data for current codeword then create a new DL-SCH transport block
        trBlk = randi([0 1],trBlkSizes(cwIdx),1);
        setTransportBlock(encodeDLSCH,trBlk,cwIdx-1);
        resetSoftBuffer(decodeDLSCH,cwIdx-1);
    end

    % Encode the DL-SCH transport blocks
    RV = zeros(1,pdsch.NumCodewords);
    codedTrBlocks = encodeDLSCH(pdsch.Modulation,pdsch.NumLayers, ...
        pdschIndicesInfo.G,RV);

    % PDSCH modulation and precoding
    pdschSymbols = nrPDSCH(carrier,pdsch,codedTrBlocks);
    [pdschAntSymbols,pdschAntIndices] = hPRGPrecode(size(dlGrid),carrier.NStartGrid,pdschSym
    dlGrid(pdschAntIndices) = pdschAntSymbols;

    % PDSCH DM-RS precoding and mapping
    dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
```

```
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);
[dmrsAntSymbols,dmrsAntIndices] = hPRGPrecode(size(dlGrid),carrier.NStartGrid,dmrsSymbols
dlGrid(dmrsAntIndices) = dmrsAntSymbols;

% OFDM modulation
txWaveform = nrOFDMModulate(carrier,dlGrid);

% Pass data through channel model. Append zeros at the end of the
% transmitted waveform to flush channel content. These zeros take
% into account any delay introduced in the channel.
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))]; %#ok<AGROW>
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

% Add AWGN to the received time-domain waveform
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

if (simParamLocal.PerfectChannelEstimator)
    % Perfect synchronization. Use information provided by the
    % channel to find the strongest multipath component
    pathFilters = getPathFilters(channel);
    [timingOffset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
else
    % Practical synchronization. Correlate the received waveform
    % with the PDSCH DM-RS to obtain the timing offset and
    % correlation magnitude. The receiver updates the timing offset
    % only when the correlation magnitude is high.
    [t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols);
    timingOffset = hSkipWeakTimingOffset(timingOffset,t,mag);
    % Display a warning if the estimated timing offset exceeds the
    % maximum channel delay
    if timingOffset > maxChDelay
        warning(['Estimated timing offset (%d) is greater than the maximum channel delay
            ' This will result in a decoding failure. This may be caused by low SNR,' ..
            ' or not enough DM-RS symbols to synchronize successfully.'],timingOffset,max
    end
end
rxWaveform = rxWaveform(1+timingOffset:end,:);

% Perform OFDM demodulation on the received data to recreate the
% resource grid, including padding in the event that practical
% synchronization results in an incomplete slot being demodulated
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end

if (simParamLocal.PerfectChannelEstimator)
    % Perfect channel estimation, using the value of the path gains
    % provided by the channel. This channel estimate does not
    % include the effect of transmitter precoding
    Hest = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,timingOffset,sampleTime

    % Get perfect noise estimate (from the noise realization)
    noiseGrid = nrOFDMDemodulate(carrier,noise(1+timingOffset:end ,:));
    noiseEst = var(noiseGrid(:));
```

```
        % Get PDSCH resource elements from the received grid and
        % channel estimate
        [pdschRx,pdschHest,~,pdschHestIndices] = nrExtractResources(pdschIndices,rxGrid,Hest

        % Apply precoding to channel estimate
        pdschHest = hPRGPrecode(size(Hest),carrier.NStartGrid,pdschHest,pdschHestIndices,perm
    else
        % Practical channel estimation between the received grid and
        % each transmission layer, using the PDSCH DM-RS for each
        % layer. This channel estimate includes the effect of
        % transmitter precoding
        [Hest,noiseEst] = hSubbandChannelEstimate(carrier,rxGrid,dmrsIndices,dmrsSymbols,pdsc

        % Average noise estimate across PRGs and layers
        noiseEst = mean(noiseEst,'all');

        % Get PDSCH resource elements from the received grid and
        % channel estimate
        [pdschRx,pdschHest] = nrExtractResources(pdschIndices,rxGrid,Hest);
    end

    % Equalization
    [pdschEq,eqCSIScaling] = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

    % Decode PDSCH physical channel
    [dlschLLRs,rxSymbols] = nrPDSCHDecode(carrier,pdsch,pdschEq,noiseEst);

    % Display EVM per layer, per slot and per RB
    if (simParamLocal.DisplayDiagnostics)
        plotLayerEVM(NSlots,nslot,pdsch,size(dlGrid),pdschIndices,pdschSymbols,pdschEq);
    end

    % Scale LLRs
    eqCSIScaling = nrLayerDemap(eqCSIScaling); % CSI scaling layer demapping
    for cwIdx = 1:pdsch.NumCodewords
        Qm = length(dlschLLRs{cwIdx})/length(rxSymbols{cwIdx});        % bits per symbol
        eqCSIScaling{cwIdx} = repmat(eqCSIScaling{cwIdx}.',Qm,1);      % expand by each bit p
        dlschLLRs{cwIdx} = dlschLLRs{cwIdx} .* eqCSIScaling{cwIdx}(:); % scale LLRs
    end

    % Decode the DL-SCH transport channel
    decodeDLSCH.TransportBlockLength = trBlkSizes;
    decodeDLSCH.TargetCodeRate = pdschextra.TargetCodeRate;
    [decbits,blkerr] = decodeDLSCH(dlschLLRs,pdsch.Modulation,pdsch.NumLayers,RV);

    % Store values to calculate throughput
    simThroughput(snrIdx) = simThroughput(snrIdx) + sum(~blkerr .* trBlkSizes);
    maxThroughput(snrIdx) = maxThroughput(snrIdx) + sum(trBlkSizes);

    % CSI measurements and encoding
    if csirsTransmission

        if (~simParamLocal.PerfectChannelEstimator)
            % Consider only the NZP-CSI-RS symbols and indices for CSI-RS based
            % channel estimation
            nzpind = (csirsSym ~= 0);

            % Calculate practical channel estimate based on CSI-RS. Use
```

```
                % a time-averaging window that covers all of the
                % transmitted CSI-RS symbols.
                [Hest,noiseEst] = nrChannelEstimate(carrier,rxGrid, ...
                    csirsInd(nzpind),csirsSym(nzpind),'CDMLengths',csirsCDMLengths);
            end

            % Generate CSI report. Store the report for use at the
            % transmitter. The CSI feedback is subject to a delay that
            % depends on the CSI report periodicity and the UE processing
            % delay. The slot in which the CSI is available to use at the
            % transmitter depends on the BS processing delay as well.
            rxCSIReport = hCSIEncode(carrier,csirs,Hest,noiseEst,csiFeedbackOpts);
            csiFeedbackSlot = nextCSISlot(csiFeedbackOpts.CSIReportPeriod,1+nslot+simParamLocal.U
            csiAvailableSlots(end+1) = 1+csiFeedbackSlot+simParamLocal.BSProcessingDelay; %#ok<SA
            csiReports(end+1) = rxCSIReport; %#ok<SAGROW>
        end

        % Print slot-wise information
        if (simParamLocal.DisplaySimulationInformation)
            printSlotInfo(NSlots,carrier,pdsch,pdschextra,blkerr,trBlkSizes./pdschIndicesInfo.G,c
        end
    end

    % Store CSI report for each SNR point
    CSIReport{snrIdx} = csiReports; %#ok<SAGROW>

    % Display the results dynamically in the command window
    if (simParamLocal.DisplaySimulationInformation)
        fprintf('\n');
    end
    fprintf('\nThroughput(Mbps) for %d frame(s) = %.4f\n',simParamLocal.NFrames,1e-6*simThroughpu

end
```

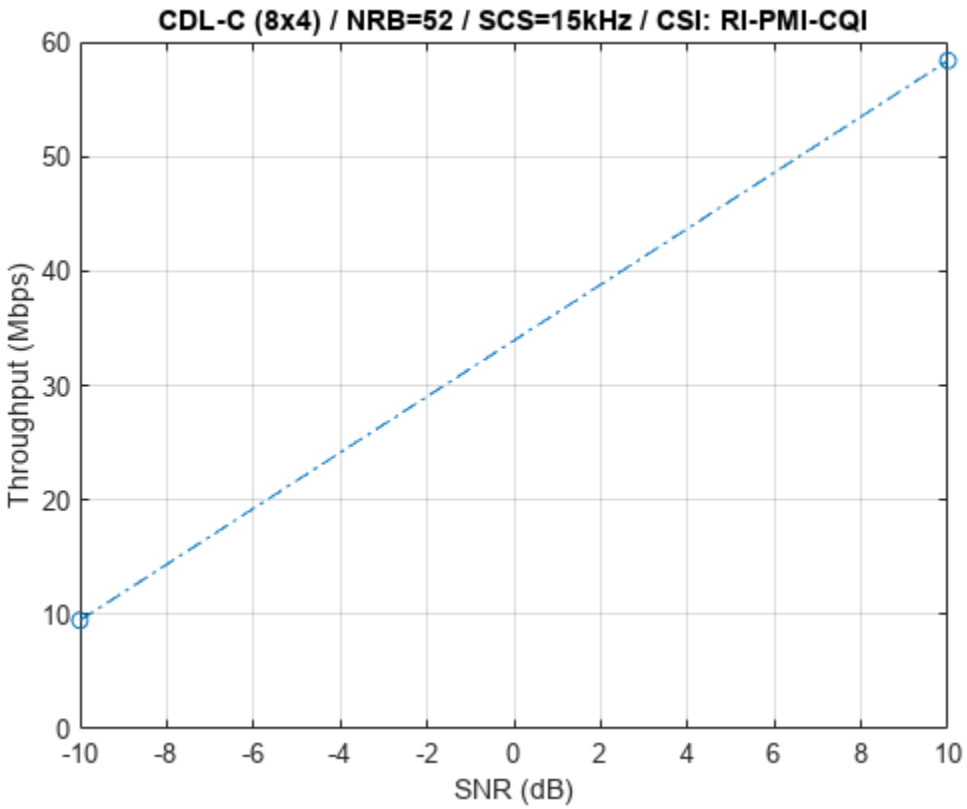
```
Simulating transmission scheme 1 (8x4) and SCS=15kHz with CDL-C channel at -10dB SNR for 2 10ms f
Using RI, PMI, and CQI as CSI feedback.

( 5.00%) NSlot= 0: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.456). Using initi
(10.00%) NSlot= 1: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(15.00%) NSlot= 2: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(20.00%) NSlot= 3: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(25.00%) NSlot= 4: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(30.00%) NSlot= 5: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(35.00%) NSlot= 6: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(40.00%) NSlot= 7: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(45.00%) NSlot= 8: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(50.00%) NSlot= 9: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(55.00%) NSlot=10: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.456). CSI-RS tran
(60.00%) NSlot=11: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(65.00%) NSlot=12: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422). Using CSI f
(70.00%) NSlot=13: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(75.00%) NSlot=14: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(80.00%) NSlot=15: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(85.00%) NSlot=16: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(90.00%) NSlot=17: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(95.00%) NSlot=18: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).
(100.00%) NSlot=19: Transmission succeeded (Layers=1, Mod=16QAM, TCR=0.479, CR=0.422).

Throughput(Mbps) for 2 frame(s) = 9.4800
```

```
Simulating transmission scheme 1 (8x4) and SCS=15kHz with CDL-C channel at 10dB SNR for 2 10ms fr
Using RI, PMI, and CQI as CSI feedback.

( 5.00%) NSlot= 0: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.624). Using initi
(10.00%) NSlot= 1: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(15.00%) NSlot= 2: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(20.00%) NSlot= 3: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(25.00%) NSlot= 4: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(30.00%) NSlot= 5: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(35.00%) NSlot= 6: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(40.00%) NSlot= 7: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(45.00%) NSlot= 8: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(50.00%) NSlot= 9: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(55.00%) NSlot=10: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.624). CSI-RS tran
(60.00%) NSlot=11: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(65.00%) NSlot=12: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578). Using CSI f
(70.00%) NSlot=13: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(75.00%) NSlot=14: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(80.00%) NSlot=15: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(85.00%) NSlot=16: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(90.00%) NSlot=17: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(95.00%) NSlot=18: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).
(100.00%) NSlot=19: Transmission succeeded (Layers=3, Mod=64QAM, TCR=0.650, CR=0.578).

Throughput(Mbps) for 2 frame(s) = 58.3840
```

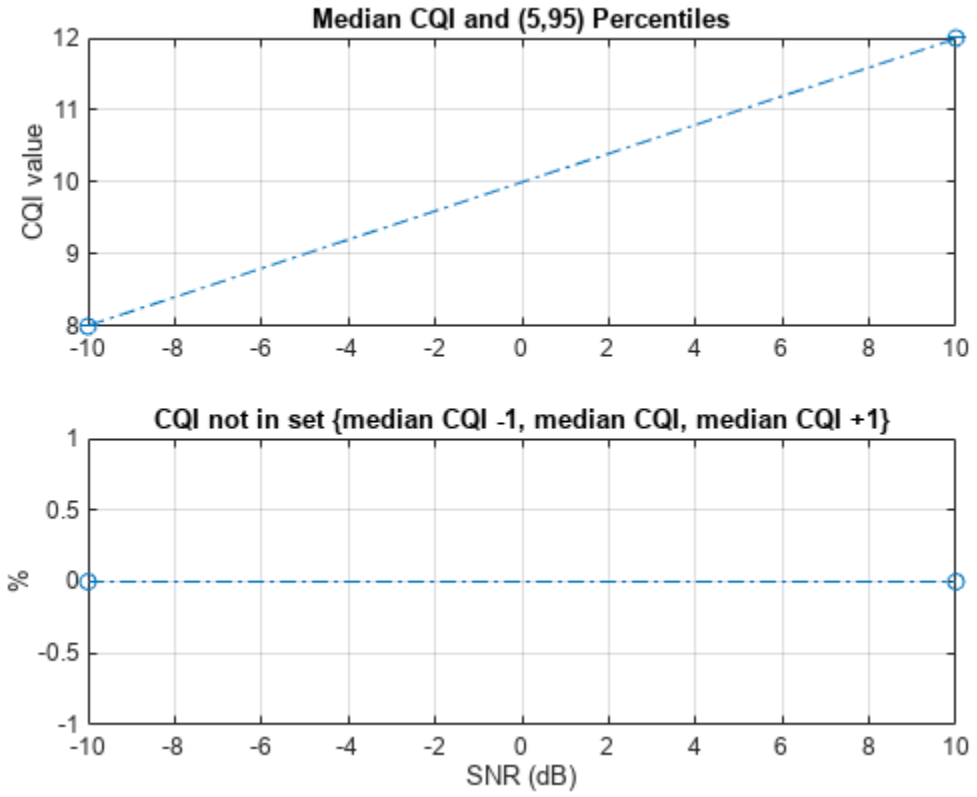
Results

Display the measured throughput as a function of the SNR.

```
figure;
plot(simParameters.SNRIn,1e-6*simThroughput/(simParameters.NFrames*10e-3),'o-.')

xlabel('SNR (dB)'); ylabel('Throughput (Mbps)'); grid on;
title(sprintf('%s (%dx%d) / NRB=%d / SCS=%dkHz / CSI: %s', ...
              simParameters.DelayProfile,simParameters.NTxAnts,simParameters.NRxAnts, ...
              simParameters.Carrier.NSizeGrid,simParameters.Carrier.SubcarrierSpacing,...
              char(simParameters.CSIReportMode)));
```




```
if simParameters.CSIReportMode == "RI-PMI-CQI"
    perc = 90;
    plotCQI(simParameters,CSIReport,perc)
end
```




```
% Bundle key parameters and results into a combined structure for recording
simResults.simParameters = simParameters;
simResults.simThroughput = simThroughput;
simResults.maxThroughput = maxThroughput;
simResults.CSIReport = CSIReport;
```

Local Functions

```
function [carrier,eDLSCH,pdsch,pdschextra,csirs,wtx] = setupTransmitter(simParameters)
% Extract channel and signal-level parameters, create DL-SCH encoder, and
% initialize MIMO precoding matrix.

    carrier = simParameters.Carrier;
    pdsch = simParameters.PDSCH;
    pdschextra = simParameters.PDSCHExtension;
    csirs = simParameters.CSIRS;

    % Select XOverhead for TBS calculation if required
    if isempty(pdschextra.XOverhead)
        pdschextra.XOverhead = getXOverhead(carrier,csirs);
    end

    % Create DL-SCH encoder system object to perform transport channel
    % encoding
    eDLSCH = nrDLSCH;

    % Initialize MIMO precoding matrix
```

```
        wtx = 1;

    end

    function [decodeDLSCH,pathFilters,timingOffset] = setupReceiver(simParameters)
    % Create and configure DL-SCH decoder and initialize receiver parameters

        % Create DL-SCH decoder system object to perform transport channel
        % decoding
        decodeDLSCH = nrDLSCHDecoder;
        decodeDLSCH.LDPCDecodingAlgorithm = simParameters.PDSCHExtension.LDPCDecodingAlgorithm;
        decodeDLSCH.MaximumLDPCIterationCount = simParameters.PDSCHExtension.MaximumLDPCIterationCou

        % Initialize channel path filters and timing offset. Timing offset is
        % updated in every slot for perfect synchronization and when the
        % correlation is strong for practical synchronization
        pathFilters = [];
        timingOffset = 0;

    end

    function channel = createChannel(simParameters)
    % Create and configure the propagation channel. If the number of antennas
    % is 1, configure only 1 polarization, otherwise configure 2 polarizations.

        % Number of antenna elements and polarizations
        nTxAnts = simParameters.NTxAnts;
        numTxPol = 1 + (nTxAnts>1);
        nRxAnts = simParameters.NRxAnts;
        numRxPol = 1 + (nRxAnts>1);

        if contains(simParameters.DelayProfile,'CDL')

            % Create CDL channel
            channel = nrCDLChannel;

            % Tx antenna array configuration in CDL channel. The number of antenna
            % elements depends on the panel dimensions. The size of the antenna
            % array is [M,N,P,Mg,Ng]. M and N are the number of rows and columns in
            % the antenna array. P is the number of polarizations (1 or 2). Mg and
            % Ng are the number of row and column array panels respectively. Note
            % that N1 and N2 in the panel dimensions follow a different convention
            % and denote the number of columns and rows, respectively.
            txArray = simParameters.TransmitAntennaArray;
            M = txArray.PanelDimensions(2);
            N = txArray.PanelDimensions(1);
            Ng = txArray.NumPanels;

            channel.TransmitAntennaArray.Size = [M N numTxPol 1 Ng];
            channel.TransmitAntennaArray.ElementSpacing = [0.5 0.5 1 1]; % Element spacing in wavele
            channel.TransmitAntennaArray.PolarizationAngles = [-45 45];  % Polarization angles in de

            % Rx antenna array configuration in CDL channel
            rxArray = simParameters.ReceiveAntennaArray;
            M = rxArray.PanelDimensions(2);
            N = rxArray.PanelDimensions(1);
            Ng = rxArray.NumPanels;
```

```
            channel.ReceiveAntennaArray.Size = [M N numRxPol 1 Ng];
            channel.ReceiveAntennaArray.ElementSpacing = [0.5 0.5 1 1];  % Element spacing in wavele
            channel.ReceiveAntennaArray.PolarizationAngles = [0 90];     % Polarization angles in de

        elseif contains(simParameters.DelayProfile,'TDL')

            channel = nrTDLChannel;
            channel.NumTransmitAntennas = nTxAnts;
            channel.NumReceiveAntennas = nRxAnts;

        else

            error('Channel not supported.')

        end

        % Configure common channel parameters: delay profile, delay spread, and
        % maximum Doppler shift
        channel.DelayProfile = simParameters.DelayProfile;
        channel.DelaySpread = simParameters.DelaySpread;
        channel.MaximumDopplerShift = simParameters.MaximumDopplerShift;

        % Get information about the baseband waveform after OFDM modulation step
        waveInfo = nrOFDMInfo(simParameters.Carrier);

        % Update channel sample rate based on carrier information
        channel.SampleRate = waveInfo.SampleRate;

    end

    function [channel,maxChannelDelay,N0] = setupChannel(simParameters,snrIdx)
    % Reset the propagation channel. Obtain the maximum channel
    % delay and calculate the AWGN standard deviation.

        % Extract channel
        channel = simParameters.Channel;
        channel.reset();

        % Get the channel information
        chInfo = info(channel);
        maxChannelDelay = chInfo.MaximumChannelDelay;

        % Calculate noise standard deviation. Normalize noise power by the IFFT
        % size used in OFDM modulation, as the OFDM modulator applies this
        % normalization to the transmitted waveform.
        SNRdB = simParameters.SNRIn(snrIdx);
        SNR = 10^(SNRdB/10);
        waveInfo = nrOFDMInfo(simParameters.Carrier);
        N0 = 1/sqrt(2.0*double(waveInfo.Nfft)*SNR);

        % Also normalize by the number of receive antennas if the channel
        % applies this normalization to the output
        if channel.NormalizeChannelOutputs
            N0 = N0/sqrt(chInfo.NumOutputSignals);
        end

    end
```

```
function csiReport = initialCSIReport(simParameters,snrIdx,carrier,csirs,channel,csiFeedbackOpts)
% Get an initial CSI report using a perfect channel estimate

    [Hest,nVar] = getInitialChannelEstimate(carrier,channel,simParameters.SNRIn(snrIdx));
    csiFeedbackOpts.PerfectChannelEstimator = true;
    csirs.CSIRSPeriod = 'on';
    csiReport = hCSIEncode(carrier,csirs,Hest,nVar,csiFeedbackOpts);

end

function [estChannelGrid,nVar] = getInitialChannelEstimate(carrier,propchannel,SNRdB)
% Obtain channel estimate before first transmission. This can be used to
% obtain initial transmission parameters

    ofdmInfo = nrOFDMInfo(carrier);

    chInfo = info(propchannel);

    % Clone channel and get path gains and sample times for perfect timing
    % and channel estimation
    channel = clone(propchannel);
    release(channel);
    channel.ChannelFiltering = false;
    channel.NumTimeSamples = (ofdmInfo.SampleRate/1000/carrier.SlotsPerSubframe)+chInfo.MaximumCh
    [pathGains,sampleTimes] = channel();

    % Perfect timing synch
    pathFilters = getPathFilters(channel);
    offset = nrPerfectTimingEstimate(pathGains,pathFilters);

    % Perfect channel estimate
    estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sampleTimes);

    % Noise variance (does not account for channel effects at this point.)
    nVar = 10^(-SNRdB/10)/size(estChannelGrid,3);

end

function XOverhead = getXOverhead(carrier,csirs)
% Calculate XOverhead for transport block size determination based on
% CSI-RS resource grid occupancy

    [~,csirsInfo] = nrCSIRSIndices(carrier,csirs);
    csirsRE = length(csirsInfo.KBarLBar{1})*length(csirsInfo.KPrime{1})*length(csirsInfo.LPrime{1
    [~,XOverhead] = quantiz(csirsRE,[0 6 12],[0 6 12 18]);

end

function cdmLengths = getCSIRSCDMLengths(csirs)
%   CDMLENGTHS = getCSIRSCDMLengths(CSIRS) returns the CDM lengths given
%   the CSI-RS configuration object CSIRS.

    CDMType = csirs.CDMType;
    if ~iscell(csirs.CDMType)
        CDMType = {csirs.CDMType};
    end
    CDMTypeOpts = {'noCDM','fd-CDM2','CDM4','CDM8'};
    CDMLengthOpts = {[1 1],[2 1],[2 2],[2 4]};
```

```
    cdmLengths = CDMLengthOpts{strcmpi(CDMTypeOpts,CDMType{1})};

end

function csiFeedbackOpts = getCSIFeedbackOptions(simParameters,snrIdx)
% Create a CSI feedback algorithmic options structure

    csiFeedbackOpts = struct();
    csiFeedbackOpts.CSIReportMode = simParameters.CSIReportMode;
    csiFeedbackOpts.CSIReportPeriod = simParameters.CSIReportConfig.Period;
    csiFeedbackOpts.CSIReportConfig = simParameters.CSIReportConfig;
    csiFeedbackOpts.PerfectChannelEstimator = simParameters.PerfectChannelEstimator;

    if simParameters.CSIReportMode == "AI CSI compression"
        % Copy additional link adaptation configuration for AI CSI compression mode
        csiFeedbackOpts.AINetworkFilename = simParameters.AINetworkFilename;

        % Download and extract a pretrained CSI network for AI CSI compression mode
        displayProgress = (snrIdx==1);
        helperCSINetDownloadData(displayProgress);
    end

end

function panelDimensions = getCSIReportPanelDimensions(antennaArray,codebookType)
% Configure the antenna array dimensions according to TS 38.214 Section
% 5.2.2.2 as a vector [N1,N2] for single-panel arrays and a vector
% [Ng,N1,N2] for multi-panel arrays.

    panelDimensions = antennaArray.PanelDimensions;

    % Add number of panels if codebook type is multi-panel
    if strcmpi(codebookType,'Type1MultiPanel')
        panelDimensions = [antennaArray.NumPanels panelDimensions];
    end
end

function csislot = nextCSISlot(period,nslot)
% Return the slot number of the first slot where CSI feedback can be
% reported according to the CSI report periodicity

    p = period(1); % Slot periodicity
    o = period(2); % Slot offset

    csislot = p*ceil((nslot-o)/p)+o;

end

function [numElemenets,numPol] = numAntennaElements(antArray)
% Calculate number of antenna elements in an antenna array

    numPol = 1 + any(antArray.PanelDimensions ~= 1);
    numElemenets = numPol*antArray.NumPanels*prod(antArray.PanelDimensions);

end

function validateCSIRSConfig(carrier,csirs,nTxAnts)
% validateCSIRSConfig(CARRIER,CSIRS,NTXANTS) validates the CSI-RS
```

```
% configuration, given the carrier specific configuration object CARRIER,
% CSI-RS configuration object CSIRS, and the number of transmit antennas
% NTXANTS.

    % Validate the number of CSI-RS ports
    if ~isscalar(unique(csirs.NumCSIRSPorts))
        error('nr5g:InvalidCSIRSPorts',...
            'All the CSI-RS resources must be configured to have the same number of CSI-RS ports
    end

    % Validate the CSI-RS and TX antenna array configuration
    if any(csirs.Ports_Options(csirs.RowNumber) ~= nTxAnts)
        rn = num2str(find(csirs.Ports_Options == nTxAnts),'%3d,');
        str = 'The number of CSI-RS ports must be equal to the number of Tx antenna elements. ';
        str = [str sprintf('For the Tx antenna array size configured, valid CSI-RS row numbers a
        error(str)
    end

    % Validate the CDM lengths
    if ~iscell(csirs.CDMType)
        cdmType = {csirs.CDMType};
    else
        cdmType = csirs.CDMType;
    end
    if (~all(strcmpi(cdmType,cdmType{1})))
        error('nr5g:InvalidCSIRSCDMTypes',...
            'All the CSI-RS resources must be configured to have the same CDM lengths.');
    end
    if nTxAnts ~= csirs.NumCSIRSPorts(1)
        error('nr5g:InvalidNumTxAnts',['Number of transmit antennas (' num2str(nTxAnts)...
            ') must be equal to the number of CSI-RS ports (' num2str(csirs.NumCSIRSPorts(1)) ')
    end

    % Check for the overlap between the CSI-RS indices
    csirsInd = nrCSIRSIndices(carrier,csirs,"OutputResourceFormat",'cell');
    numRes = numel(csirsInd);
    csirsIndAll = cell(1,numRes);
    ratioVal = csirs.NumCSIRSPorts(1)/prod(getCSIRSCDMLengths(csirs));
    for resIdx = 1:numRes
        if ~isempty(csirsInd{resIdx})
            grid = nrResourceGrid(carrier,csirs.NumCSIRSPorts(1));
            [~,tempInd] = nrExtractResources(csirsInd{resIdx},grid);
            if numel(tempInd)/numel(csirsInd{resIdx}) ~= ratioVal
                error('nr5g:OverlappedCSIRSREsSingleResource',['CSI-RS indices of resource '...
                    num2str(resIdx) ' must be unique. Try changing the symbol or subcarrier loca
            end
            csirsIndAll{resIdx} = tempInd(:);
            for idx = 1:resIdx-1
                overlappedInd = ismember(csirsIndAll{idx},csirsIndAll{resIdx});
                if any(overlappedInd)
                    error('nr5g:OverlappedCSIRSREsMultipleResources',['The resource elements of
                        'configured CSI-RS resources must not overlap. Try changing the symbol o
                        'subcarrier locations of CSI-RS resource ' num2str(idx) ' and resource '
                end
            end
        end
    end
end
```

```
function displaySNRPointProgress(simParameters,snrIdx)
% Print SNR point progress

    str = ['\nSimulating transmission scheme 1 (%dx%d) and '...
          'SCS=%dkHz with %s channel at %gdB SNR for %d 10ms frame(s)\n' ...
          'Using %s as CSI feedback.\n'];

    switch simParameters.CSIReportMode
        case 'RI-PMI-CQI'
            modeText = 'RI, PMI, and CQI';
        case 'AI CSI compression'
            modeText = 'compressed channel estimates';
        otherwise
            modeText = 'channel estimates';
    end

    SNRdB = simParameters.SNRIn(snrIdx);
    fprintf(str,simParameters.NTxAnts,simParameters.NRxAnts,simParameters.Carrier.SubcarrierSpaci
        simParameters.DelayProfile,SNRdB,simParameters.NFrames,modeText);

end

function printSlotInfo(NSlots,carrier,pdsch,pdschextra,blkerr,ECR,csirsTransmission,csiReports,re
% Print information about the current slot transmission

    ncw = pdsch.NumCodewords;
    cwLayers = floor((pdsch.NumLayers + (0:ncw-1)) / ncw);
    infoStr = [];
    for cwIdx = 1:ncw
        if blkerr
            infoStrCW = "Transmission failed";
        else
            infoStrCW = "Transmission succeeded";
        end
        infoStrCW = sprintf("%22s (Layers=%d, Mod=%5s, TCR=%.3f, CR=%.3f).",infoStrCW,cwLayers(cw
        if (ncw>1)
            infoStr = sprintf('%s\n%s%s',infoStr,sprintf('CW%d: %s',cwIdx-1),infoStrCW);
        else
            infoStr = infoStrCW;
        end
    end

    csirsInfoStr = [];
    if csirsTransmission
        csirsInfoStr = "CSI-RS transmission. ";
    end

    csifbInfoStr = [];
    if carrier.NSlot == 0
        csifbInfoStr = 'Using initial CSI.';
    elseif reportIndex > 0
        csifbInfoStr = sprintf("Using CSI from NSlot=%2d.",csiReports(reportIndex).NSlot);
    end

    nslot = carrier.NSlot;
    fprintf('\n(%5.2f%%) NSlot=%2d: %s',100*(nslot+1)/NSlots,nslot,join([infoStr,csifbInfoStr,csi
```

```
end


function plotLayerEVM(NSlots,nslot,pdsch,siz,pdschIndices,pdschSymbols,pdschEqSymbols)
% Plot EVM information

    persistent slotEVM;
    persistent rbEVM
    persistent evmPerSlot;
    persistent numLayers;

    maxNumLayers = 8;
    if (nslot==0)
        slotEVM = comm.EVM;
        rbEVM = comm.EVM;
        evmPerSlot = NaN(NSlots,maxNumLayers);
        numLayers = pdsch.NumLayers;
        figure;
    else
        % Keep the maximum number of layers in the legend
        numLayers = max(numLayers,pdsch.NumLayers);
    end
    [Ns,P] = size(pdschEqSymbols);
    pdschEqSym = zeros(Ns,maxNumLayers);
    pdschSym = zeros(Ns,maxNumLayers);
    pdschEqSym(:,1:P) = pdschEqSymbols;
    pdschSym(:,1:P) = pdschSymbols;
    evmPerSlot(nslot+1,:) = slotEVM(pdschSym,pdschEqSym);
    subplot(2,1,1);
    plot(0:(NSlots-1),evmPerSlot,'o-');
    xlabel('Slot number');
    ylabel('EVM (%)');
    legend("layer " + (1:numLayers),'Location','EastOutside');
    title('EVM per layer per slot');

    subplot(2,1,2);
    [k,~,p] = ind2sub(siz,pdschIndices);
    rbsubs = floor((k-1) / 12);
    NRB = siz(1) / 12;
    evmPerRB = NaN(NRB,maxNumLayers);
    for nu = 1:pdsch.NumLayers
        for rb = unique(rbsubs).'
            this = (rbsubs==rb & p==nu);
            evmPerRB(rb+1,nu) = rbEVM(pdschSym(this),pdschEqSym(this));
        end
    end

    plot(0:(NRB-1),evmPerRB,'x-');
    xlabel('Resource block');
    ylabel('EVM (%)');
    legend("layer " + (1:numLayers),'Location','EastOutside');
    title(['EVM per layer per resource block, slot #' num2str(nslot)]);

    drawnow;

end

function plotCQI(simParameters,CSIReport,perc)
```

```
% Plot CQI median and percentiles

    % Calculate median and percentiles
    med = cellfun(@(x) median([x.CQI]), CSIReport);
    p1 = cellfun(@(x) prctile([x.CQI],50-perc/2), CSIReport);
    p2 = cellfun(@(x) prctile([x.CQI],50+perc/2), CSIReport);

    % Calculate the percentage of CQI not in the set {median CQI -1, median CQI, median CQI +1}
    cqiPerc = cellfun(@(x,y) sum(abs([x.CQI]-y)>1)/length(x),CSIReport,num2cell(med));

    figure;
    subplot(211)
    errorbar(simParameters.SNRIn,med,p2-p1,'o-.')
    ylabel('CQI value')
    title(sprintf('Median CQI and (%g,%g) Percentiles',50-perc/2,50+perc/2));
    grid

    subplot(212)
    plot(simParameters.SNRIn,cqiPerc,'o-.')
    xlabel('SNR (dB)')
    ylabel('%')
    title('CQI not in set \{median CQI -1, median CQI, median CQI +1\}');
    grid

end
```

See Also

Related Examples

- “NR PDSCH Throughput” on page 1-58
- “5G NR Downlink CSI Reporting” on page 5-47
- “SNR Definition Used in Link Simulations” on page 5-86

Uplink Channels

5G NR Uplink Vector Waveform Generation

This example shows how to configure and generate a 5G NR uplink vector waveform with physical uplink shared channel (PUSCH) and sounding reference signal (SRS) for a baseband component carrier by using the `nrWaveformGenerator` function.

Introduction

This example shows how to parameterize and generate a 5G new radio (NR) uplink waveform by using the `nrWaveformGenerator` function. The generated waveform contains these channels and signals.

- PUSCH and its associated demodulation reference signal (DM-RS) and phase-tracking reference signal (PT-RS)
- SRS

The baseband component carrier waveform in this example is characterized by multiple subcarrier spacing (SCS) carriers and bandwidth parts (BWP), and multiple sequences of PUSCH and SRS transmission instances over the different BWPs. The example also shows how to parameterize and generate uplink control information (UCI) on PUSCH with CG-UCI and SRS for positioning.

For an example on how to generate a 5G uplink waveform with physical uplink control channel (PUCCH), see “5G NR Uplink with PUCCH Vector Waveform Generation” on page 2-14.

Waveform and Carrier Configuration

Use the `nrULCarrierConfig` object to parameterize the baseband waveform generation. This object contains a set of additional objects associated with the waveform channels and signals and enables you to set these uplink carrier configuration parameters.

- Label for this UL carrier configuration
- SCS carrier bandwidth in resource blocks
- Carrier cell ID
- Length of the generated waveform in subframes
- Windowing
- Sample rate of the OFDM-modulated waveform
- Carrier frequency for symbol phase compensation

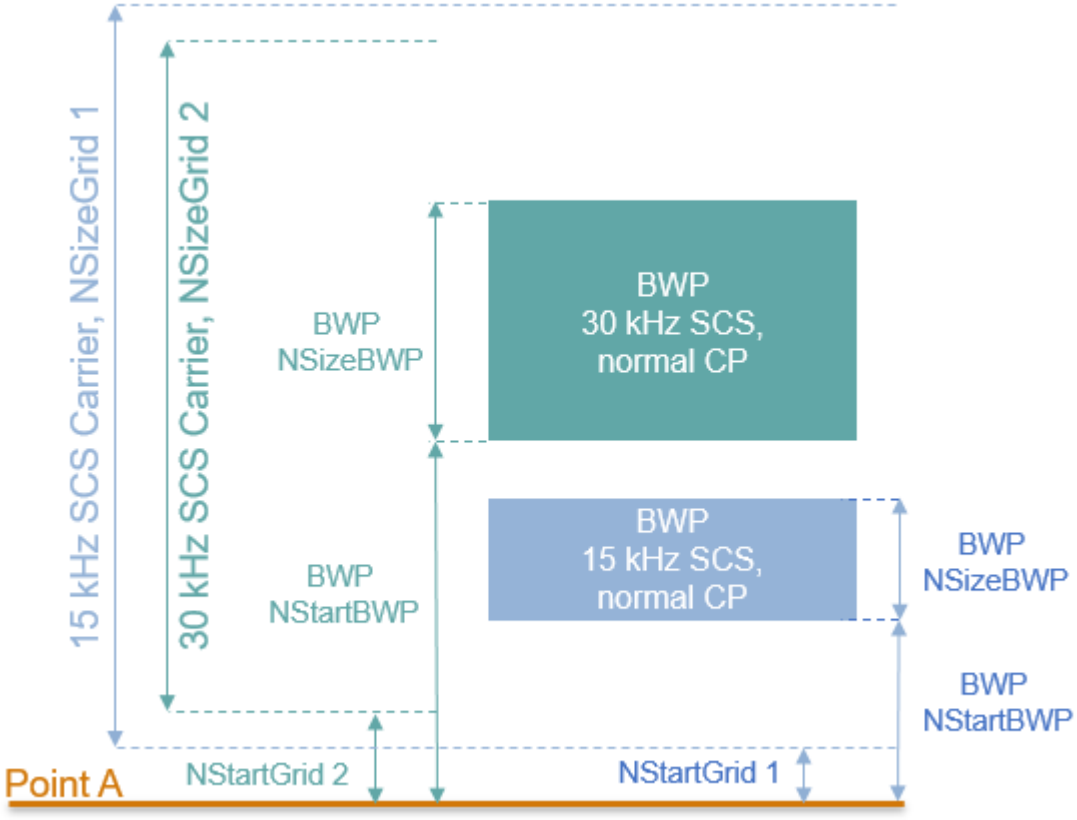
You can control SCS carrier bandwidths and guardbands using the `NStartGrid` and `NSizeGrid` properties of the `nrSCSCarrierConfig` object.

```
waveconfig = nrULCarrierConfig;    % Create an uplink carrier configuration object
waveconfig.Label = 'UL carrier 1'; % Label for this uplink waveform configuration
waveconfig.NCellID = 0;            % Cell identity
waveconfig.ChannelBandwidth = 40;  % Channel bandwidth (MHz)
waveconfig.FrequencyRange = 'FR1'; % 'FR1' or 'FR2'
waveconfig.NumSubframes = 10;      % Number of 1 ms subframes in generated waveform (1, 2, 4, 8 s
waveconfig.WindowingPercent = 0;   % Percentage of windowing relative to FFT length
waveconfig.SampleRate = [];        % Sample rate of the OFDM-modulated waveform
waveconfig.CarrierFrequency = 0;   % Carrier frequency in Hz. This property is used for symbol ph
                                   % compensation before OFDM modulation
```

```
% Define a set of SCS-specific carriers, using the maximum sizes for a
% 40 MHz NR channel. See TS 38.101-1 for more information on defined
% bandwidths and guardband requirements.
scscarriers = {nrSCSCarrierConfig,nrSCSCarrierConfig};
scscarriers{1}.SubcarrierSpacing = 15;
scscarriers{1}.NSizeGrid = 216;
scscarriers{1}.NStartGrid = 0;

scscarriers{2}.SubcarrierSpacing = 30;
scscarriers{2}.NSizeGrid = 106;
scscarriers{2}.NStartGrid = 1;
```

BWPs

A BWP is formed by a set of contiguous resources sharing a numerology on a given SCS carrier. You can define multiple BWPs using a cell array. Each element in the cell array of `nrWavegenBWPConfig` objects defines a BWP. For each BWP, you can specify the SCS, the cyclic prefix (CP) length, and the bandwidth. The `SubcarrierSpacing` property links the BWP to one of the SCS specific carriers defined earlier. The `NStartBWP` property controls the location of the BWP in the carrier, relative to point A. `NStartBWP` is expressed in common resource blocks (CRB) in terms of the BWP numerology. Different BWPs can overlap with each other.




```
% BWP configurations
bwp = {nrWavegenBWPConfig,nrWavegenBWPConfig};
bwp{1}.BandwidthPartID = 1;        % BWP ID
```

```
bwp{1}.Label = 'BWP 1 @ 15 kHz';   % Label for this BWP
bwp{1}.SubcarrierSpacing = 15;     % BWP subcarrier spacing
bwp{1}.CyclicPrefix = 'Normal';    % BWP cyclic prefix for 15 kHz
bwp{1}.NSizeBWP = 25;              % Size of BWP in PRBs
bwp{1}.NStartBWP = 10;             % Position of BWP, relative to point A, in CRBs

bwp{2}.BandwidthPartID = 2;        % BWP ID
bwp{2}.Label = 'BWP 2 @ 30 kHz';   % Label for this BWP
bwp{2}.SubcarrierSpacing = 30;     % BWP subcarrier spacing
bwp{2}.CyclicPrefix = 'Normal';    % BWP cyclic prefix for 30 kHz
bwp{2}.NSizeBWP = 51;              % Size of BWP in PRBs
bwp{2}.NStartBWP = 40;             % Position of BWP, relative to point A, in CRBs
```

PUSCH Instances Configuration

Specify the set of PUSCH transmission instances in the waveform by using a cell array. Each element in the cell array of `nrWavegenPUSCHConfig` objects defines a sequence of PUSCH transmission instances. This example defines two PUSCH sequences that model two user equipment (UE) transmissions.

General Parameters

Set these parameters for each PUSCH sequence.

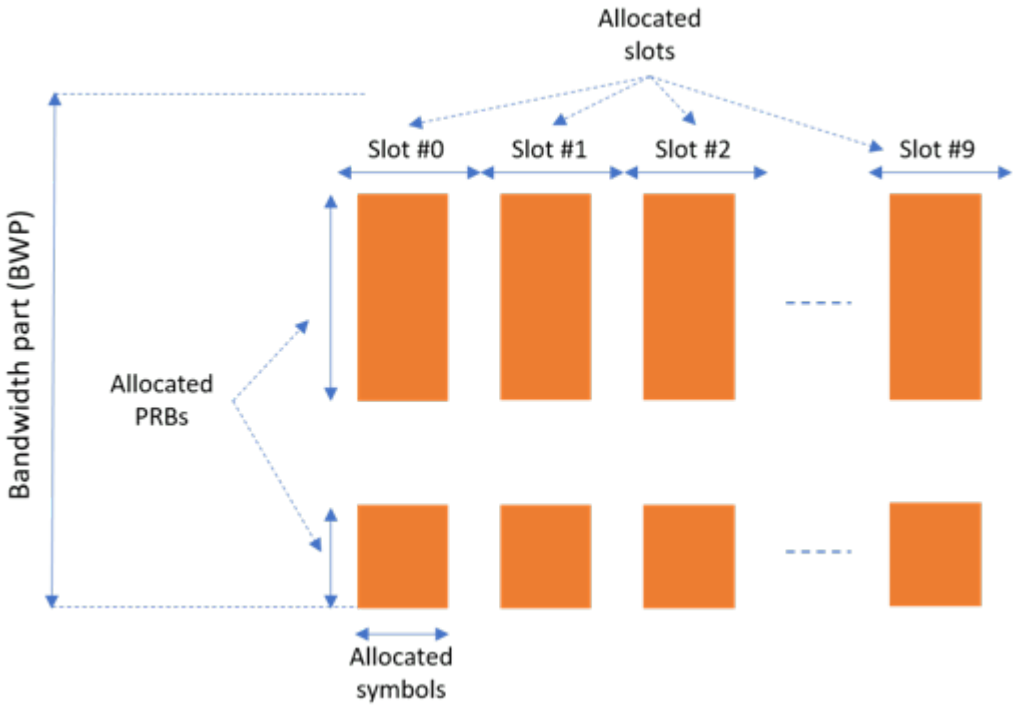
- Enable or disable this PUSCH sequence.
- Specify a label for this PUSCH sequence.
- Specify the BWP carrying the PUSCH. The PUSCH uses the SCS specified for this BWP.
- Power scaling in dB.
- Enable or disable the UL-SCH transport channel coding.
- RNTI.
- NID for scrambling the PUSCH bits.
- Transform precoding. When transform precoding is `true`, the transform precoding is enabled and the resultant waveform is DFT-s-OFDM. When transform precoding is `false`, the resultant waveform is CP-OFDM.
- Target code rate used to calculate the transport block sizes.
- Overhead parameter.
- Transmission scheme. When the transmission scheme is `'codebook'`, the MIMO precoding is enabled and a precoding matrix is selected based on the number of layers, number of antenna ports and the transmitted precoding matrix indicator. When the transmission is set to `'nonCodebook'`, an identity matrix is used, leading to no MIMO precoding.
- Symbol modulation.
- Number of layers. The number of layers is restricted to a maximum of 4 in uplink as there is only one code word transmission. Nominally, the number of layers is set to 1 when transform precoding is enabled. This value is ignored, when the `DMRS.PortSet` property is specified.
- Number of antenna ports. It is used when codebook transmission is enabled. The number of antenna ports must be greater than or equal to number of DM-RS ports configured.
- Transmitted precoding matrix indicator.
- Redundancy version (RV) sequence.
- Frequency hopping.

- Resource block offset for second hop.
- Transport block data source. You can use an array of bits or one of these standard PN sequences: `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, `'PN23'`. You can specify the seed for the generator as a cell array in the form `{'PN9', seed}`. If you do not specify a seed, the generator is initialized with all ones.

```
pusch = {nrWavegenPUSCHConfig};           % Create a PUSCH configuration object for the first UE
pusch{1}.Enable = 1;                      % Enable PUSCH sequence
pusch{1}.Label = 'UE 1 - PUSCH @ 15 kHz'; % Label for this PUSCH sequence
pusch{1}.BandwidthPartID = 1;             % BWP of PUSCH transmission
pusch{1}.Power = 0;                       % Power scaling in dB
pusch{1}.Coding = 1;                      % Enable the UL-SCH transport channel coding
pusch{1}.NID = 1;                         % Scrambling for data part
pusch{1}.RNTI = 11;                       % RNTI for the first UE
pusch{1}.TransformPrecoding = false;      % Transform precoding
pusch{1}.TargetCodeRate = 0.47;           % Code rate used to calculate transport block sizes
pusch{1}.XOverhead = 0;                   % Rate matching overhead

% Transmission settings
pusch{1}.TransmissionScheme = 'codebook'; % 'codebook','nonCodebook'
pusch{1}.Modulation = 'QPSK';             % 'pi/2-BPSK','QPSK','16QAM','64QAM','256QAM'
pusch{1}.NumLayers = 2;                   % Number of PUSCH layers
pusch{1}.NumAntennaPorts = 4;             % Number of antenna ports
pusch{1}.TPMI = 0;                        % Transmitted precoding matrix indicator (0...27)
pusch{1}.RVSequence = [0 2 3 1];          % RV sequence to be applied cyclically across the PUSCH
pusch{1}.FrequencyHopping = 'interSlot';  % Frequency hopping configuration
pusch{1}.SecondHopStartPRB = 10;          % Resource block offset for second hop

% Data source
pusch{1}.DataSource = 'PN9';              % Channel data source
```

Allocation

This figure shows the parameters of the PUSCH allocation.



You can set these parameters to control the PUSCH allocation. These parameters are relative to the BWP.

- PUSCH mapping type.
- Symbols in a slot allocated to each PUSCH instance. For PUSCH mapping type `'A'`, the start symbol within a slot must be zero and the length can be from 4 to 14 (for normal CP) and up to 12 (for extended CP). For PUSCH mapping type `'B'`, the start symbol can be from any symbol in the slot
- Slots in a frame used for the sequence of PUSCH.
- Period of the allocation in slots. Empty period indicates no repetition of the slot pattern.
- The allocated PRBs relative to the BWP.

```
pusch{1}.MappingType = 'A';          % PUSCH mapping type ('A'(slot-wise),'B'(non slot-wise))
pusch{1}.SymbolAllocation = [0 14]; % First symbol and length
pusch{1}.SlotAllocation = [0 1];     % Allocated slots indices for PUSCH sequence
pusch{1}.Period = 5;                 % Allocation period in slots
pusch{1}.PRBSet = 0:10;              % PRB allocation
```

PUSCH DM-RS Configuration

Set the DM-RS parameters.

```
% Antenna port and DM-RS configuration (TS 38.211 section 6.4.1.1)
pusch{1}.DMRSPower = 0;                    % Additional power boosting in dB

pusch{1}.DMRS.DMRSConfigurationType = 1;   % DM-RS configuration type (1,2)
pusch{1}.DMRS.NumCDMGroupsWithoutData = 2; % Number of DM-RS CDM groups without data. The value c
pusch{1}.DMRS.DMRSPortSet = [0 2];         % DM-RS antenna ports used ([] gives port numbers 0:Nu
pusch{1}.DMRS.DMRSTypeAPosition = 2;       % Mapping type A only. First DM-RS symbol position (2
```

```
pusch{1}.DMRS.DMRSLength = 1;               % Number of front-loaded DM-RS symbols (1(single symbo
pusch{1}.DMRS.DMRSAdditionalPosition = 2;  % Additional DM-RS symbol positions (max range 0...3)
pusch{1}.DMRS.NIDNSCID = 1;                 % Scrambling identity for CP-OFDM (0...65535). Use emp
pusch{1}.DMRS.NSCID = 0;                    % Scrambling initialization for CP-OFDM (0,1)
pusch{1}.DMRS.NRSID = 0;                    % Scrambling identity for DFT-s-OFDM DM-RS (0...1007)

pusch{1}.DMRS.GroupHopping = true;          % Group hopping configuration. This property is used c
pusch{1}.DMRS.SequenceHopping = false;      % Sequence hopping configuration. This property is use
```

The `GroupHopping` property is used in DM-RS sequence generation when transform precoding is enabled. You can set `GroupHopping` to:

- `'enable'` to indicate the presence of group hopping. It is configured by higher-layer parameter *`sequenceGroupHopping`*.
- `'disable'` to indicate the presence of sequence hopping. It is configured by higher-layer parameter *`sequenceHopping`*.
- `'neither'` to indicate both group hopping and sequence hopping are not present.

The number of DM-RS CDM groups without data depends on the configuration type. The maximum number of DM-RS CDM groups can be 2 for DM-RS configuration type 1 and it can be 3 for DM-RS configuration type 2.

PUSCH PT-RS Configuration

Set the PT-RS parameters.

```
% PT-RS configuration (TS 38.211 section 6.4.1.2)
pusch{1}.EnablePTRS = 0;            % Enable or disable the PT-RS (1 or 0)
pusch{1}.PTRSPower = 0;             % Additional PT-RS power boosting in dB for CP-OFDM

pusch{1}.PTRS.TimeDensity = 1;      % Time density (L_PT-RS) of PT-RS (1,2,4)
pusch{1}.PTRS.FrequencyDensity = 2; % Frequency density (K_PT-RS) of PT-RS for CP-OFDM (2,4)
pusch{1}.PTRS.NumPTRSSamples = 2;   % Number of PT-RS samples (NGroupSamp) for DFT-s-OFDM (2,4)
pusch{1}.PTRS.NumPTRSGroups = 2;    % Number of PT-RS groups (NPTRSGroup) for DFT-s-OFDM (2,4,8)
pusch{1}.PTRS.REOffset = '00';      % PT-RS resource element offset for CP-OFDM ('00','01','10',
pusch{1}.PTRS.PTRSPortSet = 0;      % PT-RS antenna ports must be a subset of DM-RS ports for CP
pusch{1}.PTRS.NID = 0;              % PT-RS scrambling identity for DFT-s-OFDM (0...1007)
```

When PT-RS is enabled for CP-OFDM, the DM-RS ports must be in the range from 0 to 3 for DM-RS configuration type 1, and in the range from 0 to 5 for DM-RS configuration type 2. When PT-RS is enabled for DFT-s-OFDM and the number of PT-RS groups is set to 8, the number of PT-RS samples must be set to 4.

UCI on PUSCH

You can set these parameters to configure the transmission of UCI on PUSCH.

- Enable or disable the transmission of HARQ-ACK, CSI part 1, CSI part2, and CG-UCI
- Number of HARQ-ACK, CSI part 1, CSI part 2, and CG-UCI bits.
- `BetaOffsetACK`, `BetaOffsetCSI1`, `BetaOffsetCSI2`, and `BetaOffsetCGUCI` can be set from the tables 9.3-1 and 9.3-2 of TS 38.213.
- Data source for HARQ-ACK, CSI part 1, CSI part 2, and CG-UCI. You can use an array of bits or one of these standard PN sequences: `'PN9-ITU'`, `'PN9'`, `'PN11'`, `'PN15'`, `'PN23'`. You can specify the seed for the generator as a cell array in the form `{'PN9', seed}`. If you do not specify a seed, the generator is initialized with all ones.

- Enable UL-SCH transmission with UCI.
- `UCIScaling` is provided by higher layer parameter *`scaling`*, as per TS 38.212, Section 6.3.2.4.

```
pusch{1}.EnableACK = true;         % Enable or disable HARQ-ACK
pusch{1}.NumACKBits = 5;           % Number of HARQ-ACK bits
pusch{1}.BetaOffsetACK = 1;        % Power factor of HARQ-ACK
pusch{1}.DataSourceACK = 'PN9';    % HARQ-ACK data source
pusch{1}.EnableCSI1 = true;        % Enable or disable CSI part 1
pusch{1}.NumCSI1Bits = 10;         % Number of CSI part 1 bits
pusch{1}.BetaOffsetCSI1 = 2;       % Power factor of CSI part 1
pusch{1}.DataSourceCSI1 = 'PN9';   % CSI part 1 data source
pusch{1}.EnableCSI2 = true;        % Enable or disable CSI part 2
pusch{1}.NumCSI2Bits = 10;         % Number of CSI part 2 bits
pusch{1}.BetaOffsetCSI2 = 2;       % Power factor of CSI part 2
pusch{1}.DataSourceCSI2 = 'PN9';   % CSI part 2 data source
pusch{1}.EnableCGUCI = false;      % Enable or disable CG-UCI
pusch{1}.NumCGUCIBits = 10;        % Number of CG-UCI bits
pusch{1}.BetaOffsetCGUCI = 2;      % Power factor of CG-UCI
pusch{1}.DataSourceCGUCI = 'PN9'; % CG-UCI data source
pusch{1}.EnableULSCH = true;       % Enable or disable UL-SCH when there is UCI transmission on PU
pusch{1}.UCIScaling = 1;           % Scaling factor (0.5, 0.65, 0.8, 1)
```

When both HARQ-ACK and CG-UCI are enabled, Section 6.3.2.1.4 of TS 38.212 specifies the UCI bit sequence as the union of the CG-UCI bits and the HARQ-ACK bits. Therefore, the processing of UCI on PUSCH considers any active CG-UCI source as an extension to HARQ-ACK and only the value of `BetaOffsetACK` is used in this case.

Specifying Multiple PUSCH Sequences

Specify the second PUSCH sequence for the second BWP.

```
pusch{2} = pusch{1};                % Create a PUSCH configuration object for the second UE
pusch{2}.Enable = 1;
pusch{2}.Label = 'UE 2 - PUSCH @ 30 kHz';
pusch{2}.BandwidthPartID = 2;       % PUSCH mapped to the second BWP
pusch{2}.RNTI = 12;                 % RNTI for the second UE
pusch{2}.SymbolAllocation = [0 12];
pusch{2}.SlotAllocation = [5 6 7 8];
pusch{2}.PRBSet = 5:10;             % PRB allocation, relative to BWP
pusch{2}.Period = 10;
pusch{2}.TransformPrecoding = 1;
pusch{2}.FrequencyHopping = 'interSlot';
pusch{2}.NumLayers = 1;
pusch{2}.RNTI = 1;

pusch{2}.DMRS.GroupHopping = false;
pusch{2}.DMRS.DMRSPortSet = 1;
```

SRS Instances Configuration

Specify SRS in the waveform. Each element in the cell array of `nrWavegenSRSConfig` objects defines a sequence of SRS instances associated with a BWP. Define two disabled SRS sequences.

General Parameters

Set these parameters for each SRS sequence.

- Enable or disable this SRS sequence.
- Specify a label for this SRS sequence.
- Specify the BWP carrying this SRS sequence. The SRS sequence configuration uses the SCS specified for this BWP.
- Specify the power scaling in dB.

```
srs = {nrWavegenSRSConfig};
srs{1}.Enable = 0;
srs{1}.Label = 'SRS @ 15 kHz';
srs{1}.BandwidthPartID = 1;
srs{1}.Power = 3; % Power scaling in dB
```

SRS Configuration

You can configure these parameters for each SRS sequence.

- Number of SRS antenna ports.
- Symbols in a slot allocated to each SRS sequence.
- Slots within a period used for SRS transmission.
- Period of the allocation in slots. Empty period indicates no repetition of the slot pattern.
- Starting position of the SRS sequence in the BWP in RBs.
- Additional frequency offset from the starting position in 4-PRB blocks.
- Bandwidth and frequency hopping configuration. The occupied bandwidth depends on the properties `CSRS`, `BSRS`, and `BHop`. Set `BHop < BSRS` to enable frequency hopping.
- Transmission comb to specify the SRS frequency density in subcarriers.
- Offset of the transmission comb in subcarriers.
- Cyclic shift rotating the low-PAPR base sequence. The maximum number of cyclic shifts, 8 or 12, depends on the transmission comb number, 2 or 4. For 4 SRS antenna ports, the subcarrier set allocated to the SRS in the first and third antenna ports depends on the cyclic shift.
- Number of repeated SRS symbols within a slot. It disables frequency hopping in blocks of `Repetition` symbols. Set `Repetition = 1` for no repetition.
- Group or sequence hopping. It can be `'neither'`, `'groupHopping'` or `'sequenceHopping'`.
- Scrambling identity. It initializes the pseudo-random binary sequence when group or sequence hopping are enabled.

```
srs{1}.NumSRSPorts = 1;             % Number of SRS ports (1,2,4)
srs{1}.NumSRSSymbols = 4;           % Number of SRS symbols in a slot (1,2,4)
srs{1}.SymbolStart = 10;            % Time-domain position of the SRS in the slot. (8...13) for r
srs{1}.SlotAllocation = 2;          % Allocated slots indices
srs{1}.Period = 5;                  % Allocation period in slots
srs{1}.FrequencyStart = 0;          % Frequency position of the SRS in BWP in RBs
srs{1}.NRRC = 0;                    % Additional offset from FreqStart specified in blocks of 4 R
srs{1}.CSRS = 13;                   % Bandwidth configuration C_SRS (0...63). It controls the al
srs{1}.BSRS = 2;                    % Bandwidth configuration B_SRS (0...3). It controls the allo
srs{1}.BHop = 1;                    % Frequency hopping configuration (0...3). Set BHop < BSRS to
srs{1}.KTC = 2;                     % Comb number (2,4). It indicates the allocation of the SRS e
srs{1}.KBarTC = 0;                  % Subcarrier offset of the SRS sequence (0...KTC-1)
srs{1}.CyclicShift = 0;             % Cyclic shift number (0...NCSmax-1). NCSmax = 8 for KTC = 2
srs{1}.Repetition = 1;              % Repetition factor (1,2,4). It indicates the number of equal
srs{1}.GroupSeqHopping = 'neither'; % Group or sequence hopping ('neither', 'groupHopping', 'sequ
```

```
srs{1}.NSRSID = 0;                  % Scrambling identity (0...1023)
srs{1}.SRSPositioning = false;      % Enable SRS for user positioning
```

Specifying Multiple SRS Sequences

Specify the second SRS sequence for the second BWP.

```
srs{2} = srs{1};
srs{2}.Enable = 0;
srs{2}.Label = 'SRS @ 30 kHz';
srs{2}.BandwidthPartID = 2;
srs{2}.NumSRSSymbols = 2;
srs{2}.SymbolStart = 12;
srs{2}.SlotAllocation = [5 6 7 8];
srs{2}.Period = 10;
srs{2}.BSRS = 0;
srs{2}.BHop = 0;
```

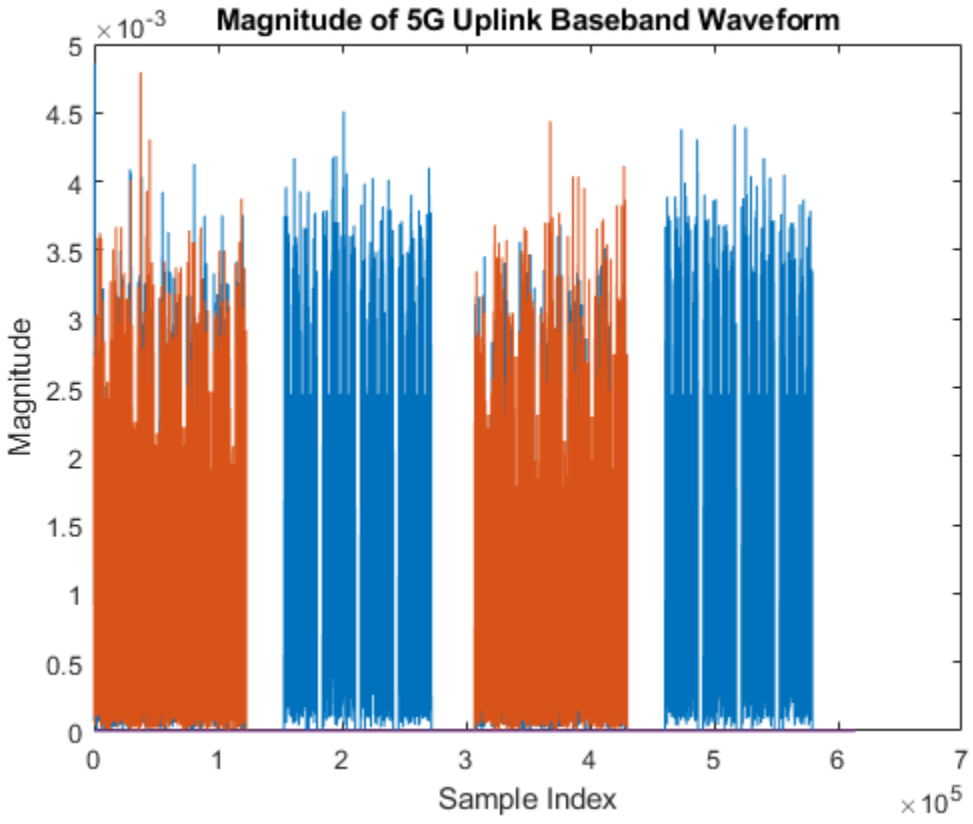
Waveform Generation

Assign all the channel and signal parameters to the main carrier configuration object `nrULCarrierConfig`, then generate and plot the waveform.

```
waveconfig.SCSCarriers = scscarriers;
waveconfig.BandwidthParts = bwp;
waveconfig.PUSCH = pusch;
waveconfig.SRS = srs;

% Generate complex baseband waveform
[waveform,info] = nrWaveformGenerator(waveconfig);
```

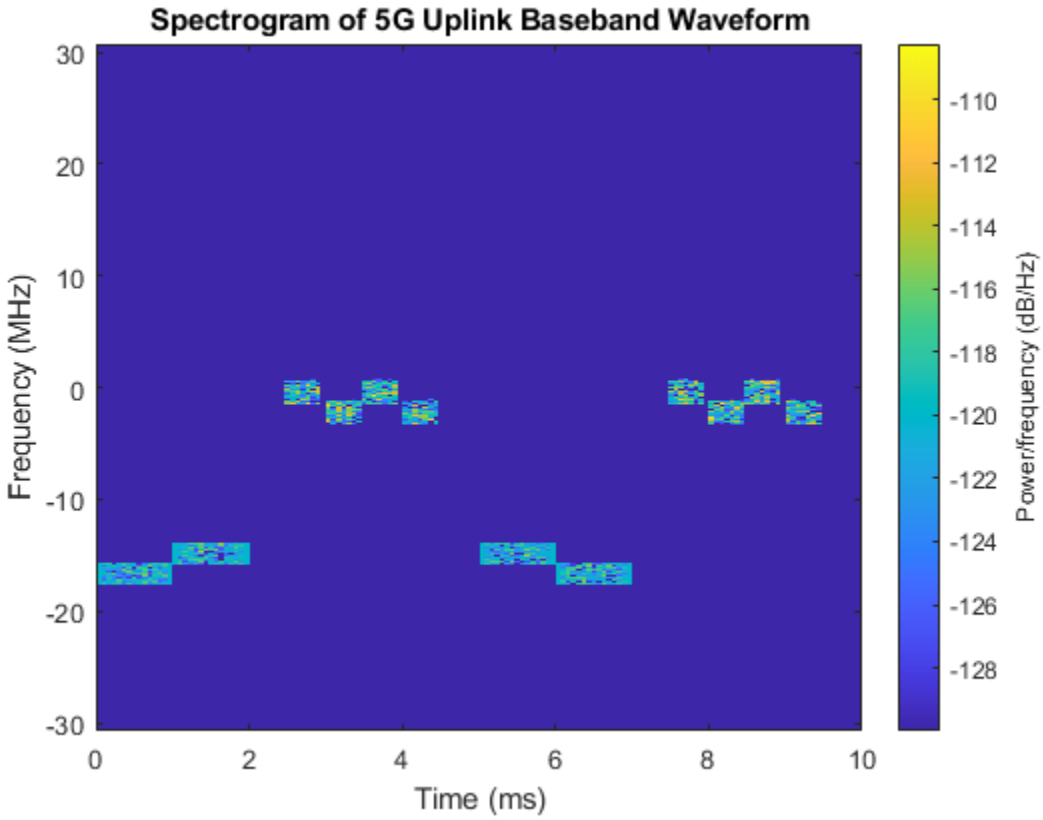
Plot the magnitude of the baseband waveform for the set of antenna ports defined.

```
figure;
plot(abs(waveform));
title('Magnitude of 5G Uplink Baseband Waveform');
xlabel('Sample Index');
ylabel('Magnitude');
```



Plot the spectogram of the waveform for the first antenna port.

```
samplerate = info.ResourceGrids(1).Info.SampleRate;
nfft = info.ResourceGrids(1).Info.Nfft;
figure;
spectrogram(waveform(:,1),ones(nfft,1),0,nfft,'centered',samplerate,'yaxis','MinThreshold',-130)
title('Spectrogram of 5G Uplink Baseband Waveform');
```



The waveform generator function returns the time-domain waveform and a structure `info`. The `info` structure contains the underlying resource element grid and a breakdown of the resources that all the PUSCH and SRS instances use in the waveform.

The `ResourceGrids` field is a structure array, which contains these fields.

- The resource grid corresponding to each BWP.
- The resource grid of the overall bandwidth containing the channels and signals in each BWP.
- An info structure with information corresponding to each BWP. For example, display the information of the first BWP.

```
disp('Modulation information associated with BWP 1:')
disp(info.ResourceGrids(1).Info)
```

```
Modulation information associated with BWP 1:
                  Nfft: 4096
            SampleRate: 61440000
    CyclicPrefixLengths: [320 288 288 288 288 288 288 320 288 288 288 ... ]
          SymbolLengths: [4416 4384 4384 4384 4384 4384 4384 4416 4384 ... ]
              Windowing: 0
           SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         SymbolsPerSlot: 14
       SlotsPerSubframe: 1
          SlotsPerFrame: 10
                     k0: 0
```

The generated resource grid is a 3-D matrix. The different planes in the grid represent the antenna ports in increasing port number order.

See Also

Functions
`nrWaveformGenerator`

Objects
`nrWavegenBWPConfig` | `nrSCSCarrierConfig` | `nrULCarrierConfig` | `nrWavegenPUSCHConfig`

More About

- “5G NR Downlink Vector Waveform Generation” on page 1-2
- “5G NR Uplink with PUCCH Vector Waveform Generation” on page 2-14

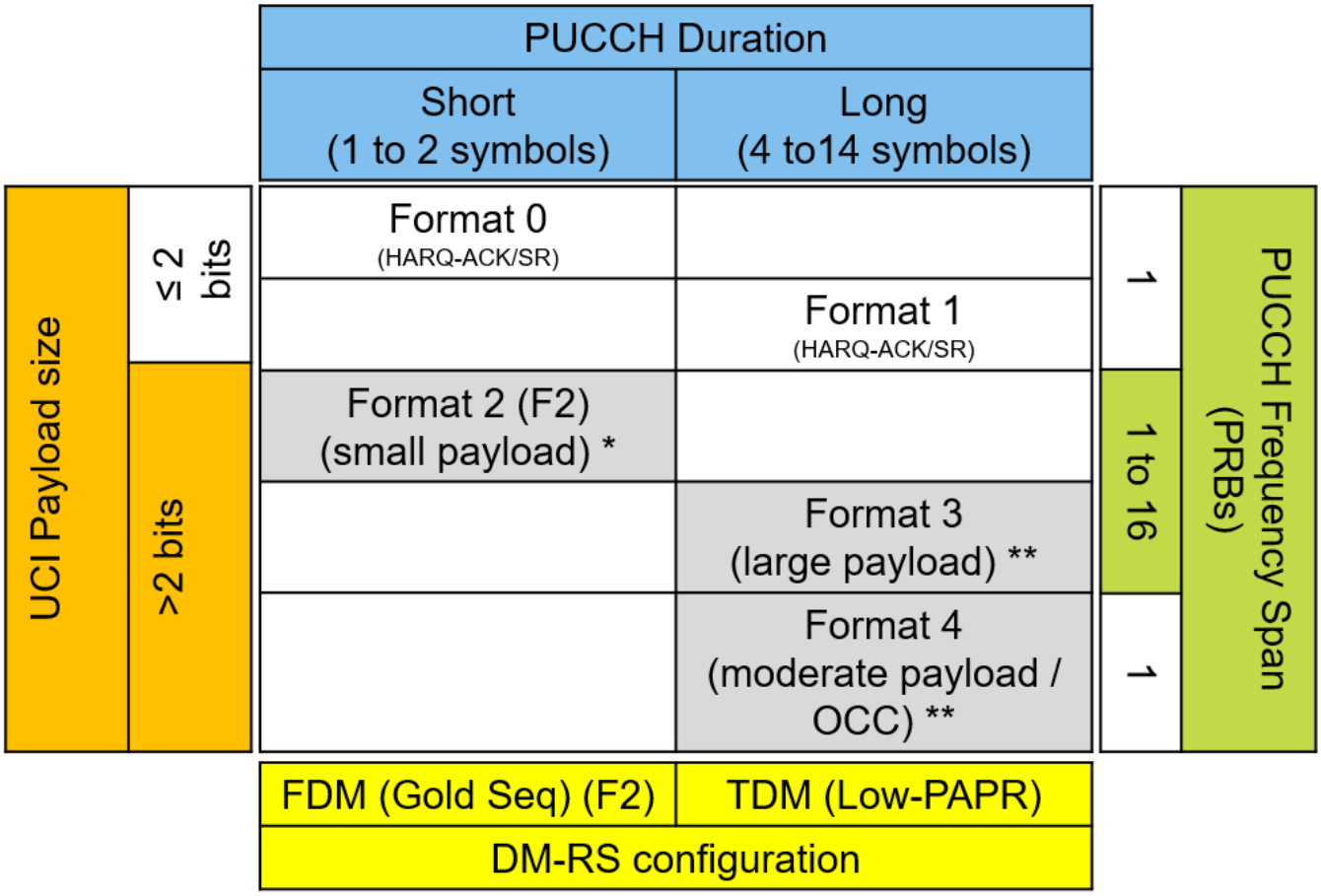
5G NR Uplink with PUCCH Vector Waveform Generation

This example shows how to configure and generate a 5G NR uplink vector waveform with physical uplink control channel (PUCCH) for a baseband component carrier by using the `nrWaveformGenerator` function.

Introduction

This example shows how to parameterize and generate a 5G new radio (NR) waveform for multiple user equipment (UE) transmissions of uplink control by using the `nrWaveformGenerator` function. The baseband component carrier waveform in this example is characterized by multiple subcarrier spacing (SCS) carriers and bandwidth parts (BWP), and multiple sequences of PUCCH transmission instances and their demodulation reference signals (DM-RS) over the different BWPs. Each sequence of PUCCH models a separate UE transmission. For an example on how to generate a 5G uplink waveform with physical uplink shared channel (PUSCH) and sounding reference signal (SRS), including Release 16 CG-UCI and SRS for positioning, see “5G NR Uplink Vector Waveform Generation” on page 2-2.

The example configures multiple sequences of PUCCH for several formats. This figure shows the characteristics of each PUCCH format, as defined in TS 38.211 Section 6.3.2.



Waveform and Carrier Configuration

Use the `nrULCarrierConfig` object to parameterize baseband waveform generation. This object contains a set of additional objects associated with the waveform channels and signals and enables you to set these uplink carrier configuration parameters.

- Label for this UL carrier configuration
- SCS carrier bandwidth in resource blocks
- Carrier cell ID
- Length of the generated waveform in subframes
- Windowing
- Sample rate of the OFDM-modulated waveform
- Carrier frequency for symbol phase compensation

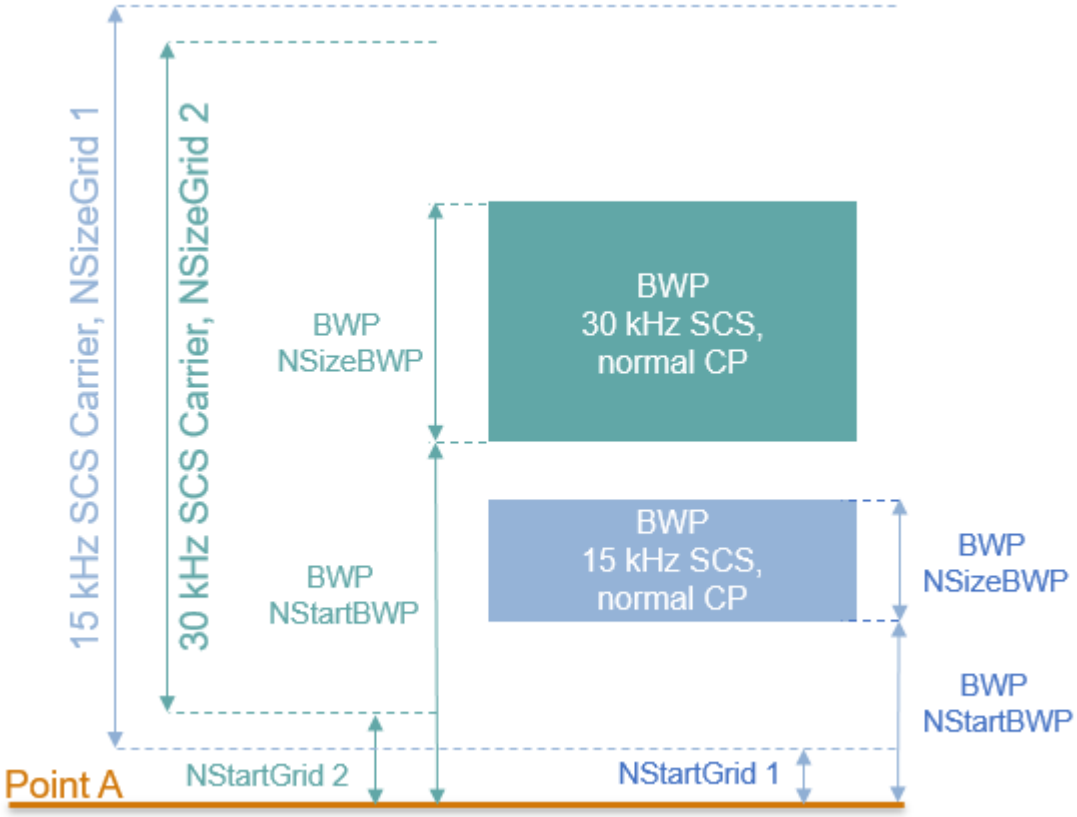
You can control SCS carrier bandwidths and guardbands using the `NStartGrid` and `NSizeGrid` properties of the `nrSCSCarrierConfig` object.

```
waveconfig = nrULCarrierConfig;    % Create an uplink carrier configuration object
waveconfig.Label = 'UL carrier 1'; % Label for this uplink waveform configuration
```

```
waveconfig.NCellID = 0;            % Cell identity
waveconfig.ChannelBandwidth = 40;  % Channel bandwidth (MHz)
waveconfig.FrequencyRange = 'FR1'; % 'FR1' or 'FR2'
waveconfig.NumSubframes = 10;      % Number of 1 ms subframes in generated waveform (1, 2, 4, 8 s
waveconfig.WindowingPercent = 0;   % Percentage of windowing relative to FFT length
waveconfig.SampleRate = [];        % Sample rate of the OFDM-modulated waveform
waveconfig.CarrierFrequency = 0;   % Carrier frequency in Hz. This property is used for symbol ph
                                   % compensation before OFDM modulation

% Define a set of SCS specific carriers, using the maximum sizes for a
% 40 MHz NR channel. See TS 38.101-1 for more information on defined
% bandwidths and guardband requirements.
scscarriers = {nrSCSCarrierConfig,nrSCSCarrierConfig};
scscarriers{1}.SubcarrierSpacing = 15;
scscarriers{1}.NSizeGrid = 216;
scscarriers{1}.NStartGrid = 0;

scscarriers{2}.SubcarrierSpacing = 30;
scscarriers{2}.NSizeGrid = 106;
scscarriers{2}.NStartGrid = 1;
```

BWPs

A BWP is formed by a set of contiguous resources sharing a numerology on a given SCS carrier. You can define multiple BWPs using a cell array. Each element in the cell array of `nrWavegenBWPConfig` objects defines a BWP. For each BWP, you can specify the SCS, the cyclic prefix (CP) length, and the bandwidth. The `SubcarrierSpacing` property links the BWP to one of the SCS-specific carriers defined earlier. The `NStartBWP` property controls the location of the BWP in the carrier, relative to point A. `NStartBWP` is expressed in common resource blocks (CRB) in terms of the BWP numerology. Different BWPs can overlap with each other.




```
% BWP configurations
bwp = {nrWavegenBWPConfig,nrWavegenBWPConfig};
bwp{1}.BandwidthPartID = 1;         % BWP ID
bwp{1}.Label = 'BWP 1 @ 15 kHz';    % Label for this BWP
bwp{1}.SubcarrierSpacing = 15;      % BWP subcarrier spacing
bwp{1}.CyclicPrefix = 'Normal';     % BWP cyclic prefix for 15 kHz
bwp{1}.NSizeBWP = 25;               % Size of BWP in PRBs
bwp{1}.NStartBWP = 10;              % Position of BWP, relative to point A, in CRBs

bwp{2}.BandwidthPartID = 2;         % BWP ID
bwp{2}.Label = 'BWP 2 @ 30 kHz';    % Label for this BWP
bwp{2}.SubcarrierSpacing = 30;      % BWP subcarrier spacing
bwp{2}.CyclicPrefix = 'Normal';     % BWP cyclic prefix for 30 kHz
bwp{2}.NSizeBWP = 51;               % Size of BWP in PRBs
bwp{2}.NStartBWP = 40;              % Position of BWP, relative to point A, in CRBs
```

PUCCH Instances Configuration

There are five different formats of PUCCH, each one for different control purposes. Use these format-specific configuration objects to define a sequence of PUCCH instances.

- `nrWavegenPUCCH0Config`
- `nrWavegenPUCCH1Config`
- `nrWavegenPUCCH2Config`

- `nrWavegenPUCCH3Config`
- `nrWavegenPUCCH4Config`

This section specifies the set of PUCCH transmission instances in the waveform by using a cell array. Each element in the cell array defines a sequence of PUCCH transmission instances and must be one of the objects listed above. This example defines three PUCCH sequences that model three UE transmissions: a PUCCH format 3, a PUCCH format 2, and a PUCCH format 0. Some of the properties assigned in this section only apply to certain PUCCH formats. For a list of all properties for a specific format, see the documentation of the PUCCH configuration object for that format.

General Parameters

Set these parameters, common to all formats, for each PUCCH sequence.

- Enable or disable this PUCCH sequence
- Specify a label for this PUCCH sequence
- Specify the BWP carrying the PUCCH. The PUCCH uses the SCS specified for this BWP
- Power scaling in dB

```
pucch = {nrWavegenPUCCH3Config};                    % Create a PUCCH format 3 configuration objec
pucch{1}.Enable = 1;                                % Enable PUCCH sequence
pucch{1}.Label = 'UE 1 - PUCCH Format 3 @ 15 kHz'; % Label for this PUCCH sequence
pucch{1}.BandwidthPartID = 1;                       % BWP of PUCCH transmission
pucch{1}.Power = 0;                                 % Power scaling in dB
```

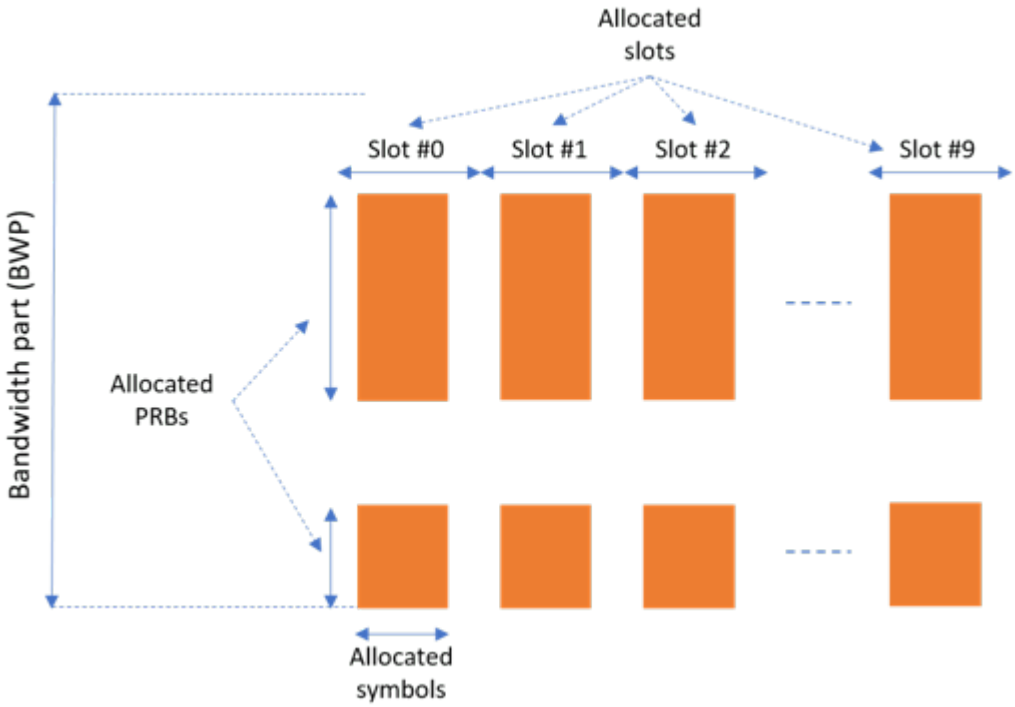
Set these format-specific parameters for each PUCCH sequence.

- Modulation scheme.
- Frequency hopping configuration.
- Resource block offset for second hop.
- Group hopping configuration.
- PUCCH hopping identity. The value is used in sequence generation for format 0, both sequence and DM-RS generation for format 1, and only for DM-RS generation for formats 3 and 4.
- RNTI.
- NID for scrambling the uplink control information (UCI) bits.

```
pucch{1}.Modulation = 'QPSK';            % 'pi/2-BPSK','QPSK'
pucch{1}.FrequencyHopping = 'intraSlot'; % Frequency hopping configuration
pucch{1}.SecondHopStartPRB = 10;         % Resource block offset for second hop
pucch{1}.GroupHopping = 'enable';        % Group hopping configuration
pucch{1}.HoppingID = 1;                  % Hopping identity
pucch{1}.RNTI = 11;                      % RNTI for the first UE
pucch{1}.NID = 0;                        % Scrambling identity
```

Allocation

This figure shows the parameters used in the PUCCH allocation.



You can set these parameters to control the PUCCH allocation. These parameters are relative to the BWP.

- Symbols in a slot allocated to each PUCCH instance. For PUCCH formats 0 and 2, you can only allocate 1 or 2 symbols. For PUCCH formats 1, 3, and 4, you must allocate at least 4 symbols in a slot.
- Slots in a frame used for the sequence of PUCCH.
- Period of the allocation in slots. Empty period indicates no repetition of the slot pattern.
- The allocated PRBs relative to the BWP. For formats 0, 1, and 4, you can only allocate a single PRB.

```
pucch{1}.SymbolAllocation = [3 11]; % First symbol and length
pucch{1}.SlotAllocation = [3 4];    % Allocated slots indices for PUCCH sequence
pucch{1}.Period = 6;                % Allocation period in slots
pucch{1}.PRBSet = 0:9;              % PRB allocation
```

PUCCH DM-RS Configuration

You can set these parameters to control the PUCCH DM-RS for each PUCCH sequence.

- Presence of additional DM-RS
- Additional power boosting for DM-RS

```
pucch{1}.AdditionalDMRS = 1;        % Additional DM-RS
pucch{1}.DMRSPower = 1;             % Additional power boosting for DM-RS in dB
```

UCI Payload Configuration

Set these parameters for the UCI payload configuration.

- Enable or disable the UCI coding.
- Target code rate used to calculate the transport block sizes when both UCI part 1 and UCI part 2 are present.
- Number of UCI (HARQ-ACK, SR, and CSI part 1) bits.
- Number of UCI part 2 (CSI part 2) bits.
- Data source for UCI and UCI part 2. You can use an array of bits or one of these standard PN sequences: 'PN9-ITU', 'PN9', 'PN11', 'PN15', 'PN23'. You can specify the seed for the generator as a cell array in the form {'PN9', seed}. If you do not specify a seed, the generator is initialized with all ones.

```
pucch{1}.Coding = 1;
pucch{1}.TargetCodeRate = 0.15;
pucch{1}.NumUCIBits = 20;
pucch{1}.NumUCI2Bits = 10;
pucch{1}.DataSourceUCI = 'PN9';
pucch{1}.DataSourceUCI2 = 'PN9';
```

Specifying Multiple PUCCH Instances

Specify two additional PUCCH sequences for the second BWP. The first one is a PUCCH format 2 that is allocated in the lower end of the second BWP, with no hopping and no repetition. The second sequence is a PUCCH format 0 that is allocated in the top half part of the second BWP and characterized by interslot hopping, 2 UCI bits containing HARQ-ACK, and one scheduling resource (SR) bit.

```
pucch{2} = nrWavegenPUCCH2Config;                    % Create a PUCCH format 2 configuration objec
pucch{2}.Label = 'UE 2 - PUCCH Format 2 @ 30 kHz'; % Label for this PUCCH sequence
pucch{2}.BandwidthPartID = 2;                        % PUCCH mapped to 2nd BWP
pucch{2}.SymbolAllocation = [10 2];                  % Symbol allocation
pucch{2}.SlotAllocation = 0:2;                       % Slot allocation
pucch{2}.Period = [];                                % Specify no repetitions of the slot pattern
pucch{2}.RNTI = 12;                                  % RNTI for the second UE
pucch{2}.NID0 = 0;                                   % DM-RS scrambling identity

pucch{3} = nrWavegenPUCCH0Config;                    % Create a PUCCH format 0 configuration objec
pucch{3}.Label = 'UE 3 - PUCCH Format 0 @ 30 kHz'; % Label for this PUCCH sequence
pucch{3}.BandwidthPartID = 2;                        % PUCCH mapped to 2nd BWP
pucch{3}.SymbolAllocation = [1 2];                   % Symbol allocation
pucch{3}.PRBSet = 40;                                % PRB allocation
pucch{3}.FrequencyHopping = 'interSlot';             % Frequency hopping
pucch{3}.SecondHopStartPRB = 30;                     % Resource block offset for second hop
pucch{3}.InitialCyclicShift = 3;                     % Initial cyclic shift
pucch{3}.NumUCIBits = 2;                             % Number of UCI bits containing HARQ-ACK
pucch{3}.DataSourceSR = 1;                           % SR data source
```

PUSCH Instances Configuration

Specify the set of PUSCH instances in the waveform by using a cell array. Each element in the cell array of `nrWavegenPUSCHConfig` objects defines a sequence of PUSCH instances. Disable the PUSCH sequence in the first BWP.

```
pusch = {nrWavegenPUSCHConfig};
pusch{1}.Enable = 0;
pusch{1}.Label = 'PUSCH @ 15 kHz';
pusch{1}.BandwidthPartID = 1;
```

SRS Instances Configuration

Specify SRS in the waveform. Each element in the cell array of `nrWavegenSRSConfig` objects defines a sequence of SRS instances associated with a BWP. Disable the SRS sequence in the first BWP.

```
srs = {nrWavegenSRSConfig};
srs{1}.Enable = 0;
srs{1}.Label = 'SRS @ 15 kHz';
srs{1}.BandwidthPartID = 1;
```

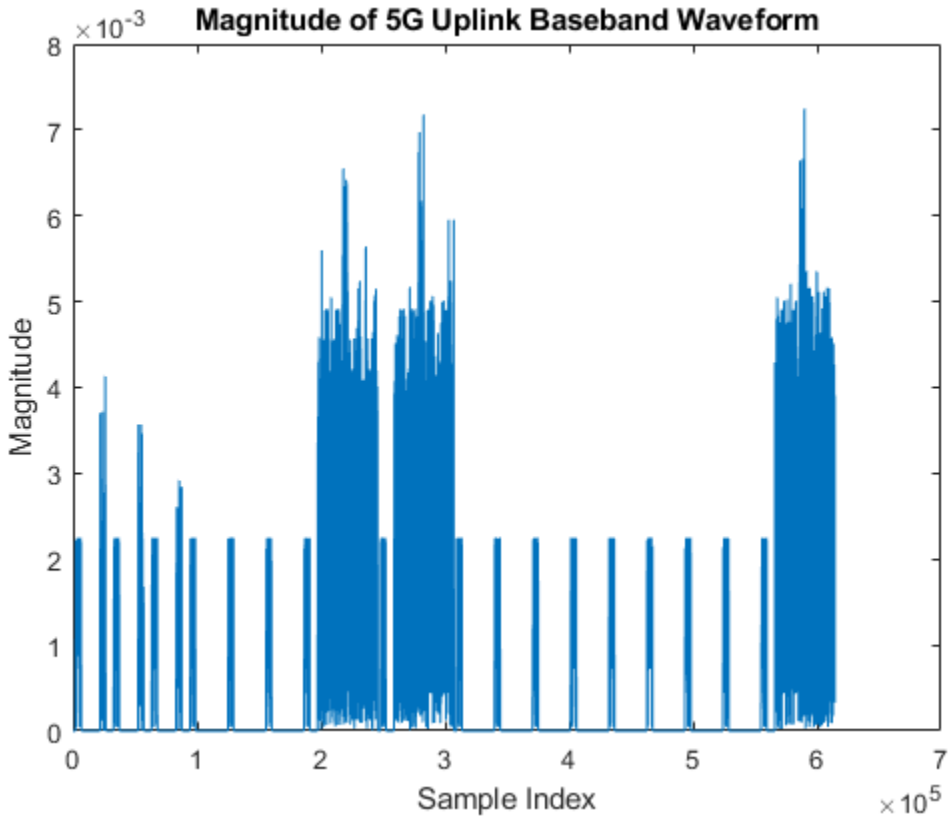
Waveform Generation

Assign all the channel and signal parameters to the main carrier configuration object `nrULCarrierConfig`, then generate and plot the waveform.

```
waveconfig.SCSCarriers = scscarriers;
waveconfig.BandwidthParts = bwp;
waveconfig.PUCCH = pucch;
waveconfig.PUSCH = pusch;
waveconfig.SRS = srs;

% Generate complex baseband waveform
[waveform,info] = nrWaveformGenerator(waveconfig);
```

Plot the magnitude of the baseband waveform.

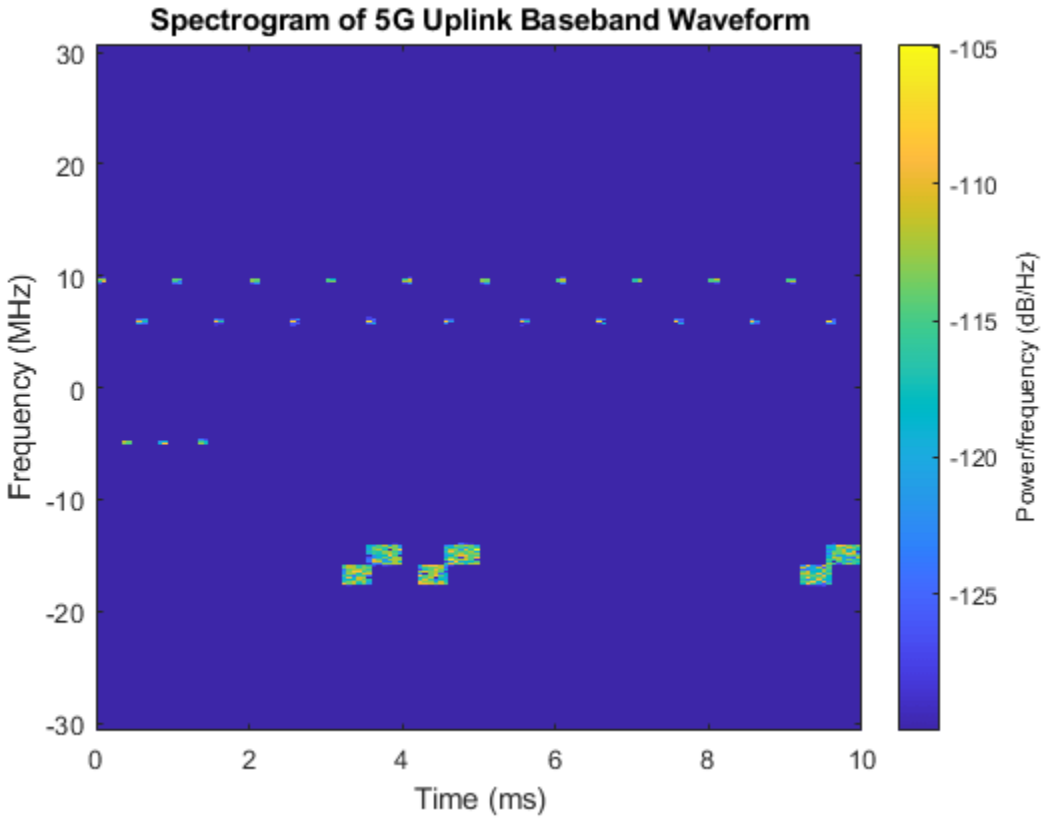
```
figure;
plot(abs(waveform));
title('Magnitude of 5G Uplink Baseband Waveform');
xlabel('Sample Index');
ylabel('Magnitude');
```



Plot the spectogram of the baseband waveform. The plot shows the resource allocation of the three PUCCH sequences.

- PUCCH format 3, in the first BWP, is in the lower part of the spectrogram. The plot shows intraslot frequency hopping of this PUCCH.
- PUCCH format 2, in the second BWP, is around -10 MHz.
- PUCCH format 0, in the second BWP, is the central part of the spectrogram. The plot shows interslot frequency hopping of this PUCCH.

```
samplerate = info.ResourceGrids(1).Info.SampleRate;
nfft = info.ResourceGrids(1).Info.Nfft;
figure;
spectrogram(waveform(:,1),ones(nfft,1),0,nfft,'centered',samplerate,'yaxis','MinThreshold',-130)
title('Spectrogram of 5G Uplink Baseband Waveform');
```



The waveform generator function returns the time-domain waveform and a structure `info`. The `info` structure contains the underlying resource element grid and a breakdown of the resources that all the PUCCH, PUSCH, and SRS instances use in the waveform.

For example, display the high-level information of the first PUCCH.

```
disp('Information associated with the first PUCCH:')
disp(info.WaveformResources.PUCCH(1))
```

```
Information associated with the first PUCCH:
          Name: 'UE 1 - PUCCH Format 3 @ 15 kHz'
        Format: 3
    CDMLengths: [1 1]
     Resources: [1x3 struct]
```

The `ResourceGrids` field is a structure array, which contains these fields.

- The resource grid corresponding to each BWP.
- The resource grid of the overall bandwidth containing the channels and signals in each BWP.
- An info structure with information corresponding to each BWP. For example, display the information for the first BWP.

```
disp('Modulation information associated with BWP 1:')
disp(info.ResourceGrids(1).Info)
```

```
Modulation information associated with BWP 1:
                  Nfft: 4096
```

```
         SampleRate: 61440000
CyclicPrefixLengths: [320 288 288 288 288 288 288 320 288 288 288 ... ]
      SymbolLengths: [4416 4384 4384 4384 4384 4384 4384 4416 4384 ... ]
          Windowing: 0
       SymbolPhases: [0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     SymbolsPerSlot: 14
   SlotsPerSubframe: 1
      SlotsPerFrame: 10
                 k0: 0
```

See Also

Functions

nrPUSCH | nrPUCCH0 | nrPUCCH1 | nrPUCCH2 | nrPUCCH3 | nrPUCCH4 | nrULSCH

More About

- “5G NR Uplink Vector Waveform Generation” on page 2-2

NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals

This example shows the time-frequency aspects of the new radio (NR) physical uplink shared channel (PUSCH), the associated demodulation reference signal (DM-RS), and phase tracking reference signal (PT-RS). The example shows how PUSCH resource allocation affects the time-frequency structure of DM-RS and PT-RS.

Introduction

In 5G NR, PUSCH is the physical uplink channel that carries user data. DM-RS and PT-RS are the reference signals associated with PUSCH. DM-RS is used for channel estimation as part of coherent demodulation of PUSCH. To compensate for the common phase error (CPE), 3GPP 5G NR introduced PT-RS. Phase noise produced in local oscillators introduces a significant degradation at mmWave frequencies. It produces CPE and inter-carrier interference (ICI). CPE leads to an identical rotation of a received symbol in each subcarrier. ICI leads to a loss of orthogonality between the subcarriers. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance.

The time-frequency structure of reference signals depends on the type of waveform configured for PUSCH, as defined in TS 38.211 Sections 6.4.1.1 and 6.4.1.2 [1] on page 2-41. When transform precoding is disabled, the waveform configured is cyclic-prefix-orthogonal frequency division multiplexing (CP-OFDM). When transform precoding is enabled, the waveform configured is discrete-fourier-transform-spread orthogonal frequency division multiplexing (DFT-s-OFDM).

The 5G Toolbox™ provides the functions for physical (PHY) layer modeling with varying levels of granularity. The levels of granularity range from PHY channel level functions that perform the transport and physical channel processing to individual channel processing stage functions performing cyclic redundancy check (CRC) coding, code block segmentation, low density parity check (LDPC) channel coding, and so on. The toolbox provides reference signals functionality associated with the PUSCH as functions `nrPUSCHDMRS`, `nrPUSCHDMRSIndices`, `nrPUSCHPTRS`, and `nrPUSCHPTRSIndices`.

PUSCH

PUSCH is the physical channel that carries the user data. The resources allocated for PUSCH are within the bandwidth part (BWP) of the carrier, as defined in TS 38.214 Section 6.1.2 [2] on page 2-41. The resources in time domain for PUSCH transmission are scheduled by downlink control information (DCI) in the field *Time domain resource assignment.* This field indicates the slot offset K_0, starting symbol **S**, the allocation length **L**, and the mapping type of PUSCH. The valid combinations of **S** and **L** are shown in Table 1.

PUSCH Mapping Type	Normal Cyclic Prefix			Extended Cyclic Prefix		
	S	*L*	*S+L*	*S*	*L*	*S+L*
Type A	0	{4,...,14}	{4,...,14}	0	{4,...,12}	{4,...,12}
Type B	{0,...,13}	{1,...,14}	{1,...,14}	{0,...,11}	{1,...,12}	{1,...,12}

Table 1: Valid *S* and *L* Combinations

The resources in the frequency domain for PUSCH transmission are scheduled by a DCI in the field *Frequency domain resource assignment*. This field indicates whether the resource allocation of resource blocks (RBs) is contiguous or noncontiguous, based on the allocation type. The RBs allocated are within the BWP.

The 5G Toolbox™ provides the `nrCarrierConfig` and `nrPUSCHConfig` objects to set the parameters related to the PUSCH within the BWP.

```
% Setup the carrier with 15 kHz subcarrier spacing and 10 MHz bandwidth
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 52;
carrier.NStartGrid = 0;

% Configure the physical uplink shared channel parameters
pusch = nrPUSCHConfig;
pusch.NSizeBWP = [];   % Empty implies that the value is equal to NSizeGrid
pusch.NStartBWP = [];  % Empty implies that the value is equal to NStartGrid
pusch.PRBSet = 0:25;   % Allocate half of the carrier bandwidth
pusch.SymbolAllocation = [0 14]; % Symbol allocation [S L]
pusch.MappingType = 'A'; % PUSCH mapping type ('A' or 'B')
pusch.TransmissionScheme = 'nonCodebook'; % ('codebook' or 'nonCodebook')
% The following parameters are applicable when TransmissionScheme is set
% to 'codebook'
pusch.NumAntennaPorts = 4;
pusch.TPMI = 0;
```

DM-RS for CP-OFDM

DM-RS is used to estimate the radio channel. DM-RS is present only in the RBs scheduled for PUSCH. The DM-RS structure is designed to support different deployment scenarios and use cases.

Parameters That Control Time Resources

The parameters that control the time resources of DM-RS are:

- PUSCH symbol allocation
- Mapping type
- Intra-slot frequency hopping
- DM-RS type A position

- DM-RS length
- DM-RS additional position

Symbol allocation of PUSCH indicates the OFDM symbol locations allocated for the PUSCH transmission in a slot. The mapping type indicates the first DM-RS OFDM symbol location and the duration of OFDM symbols (l_d). For mapping type A, l_d is the duration between the first OFDM symbol of the slot and the last OFDM symbol of the allocated PUSCH resources. For mapping type B, l_d is the duration of the allocated PUSCH resources. When intra-slot frequency hopping is enabled, l_d is the duration per hop. The DM-RS symbols are present in each hop when intra-slot frequency hopping is enabled. When intra-slot frequency hopping is enabled, DM-RS is single-symbol with the maximum number of additional positions either 0 or 1. The DM-RS symbol locations is given by TS 38.211 Tables 6.4.1.1.3-3, 6.4.1.1.3-4, and 6.4.1.1.3-6. Figure 1 shows the DM-RS symbol locations for PUSCH occupying 14 symbols with PUSCH mapping type A, intra-slot frequency hopping enabled, and number of DM-RS additional positions as 1. The figure shows DM-RS is present in each hop. The locations of DM-RS symbols in each hop depends on the number of OFDM symbols allocated for PUSCH in each hop.

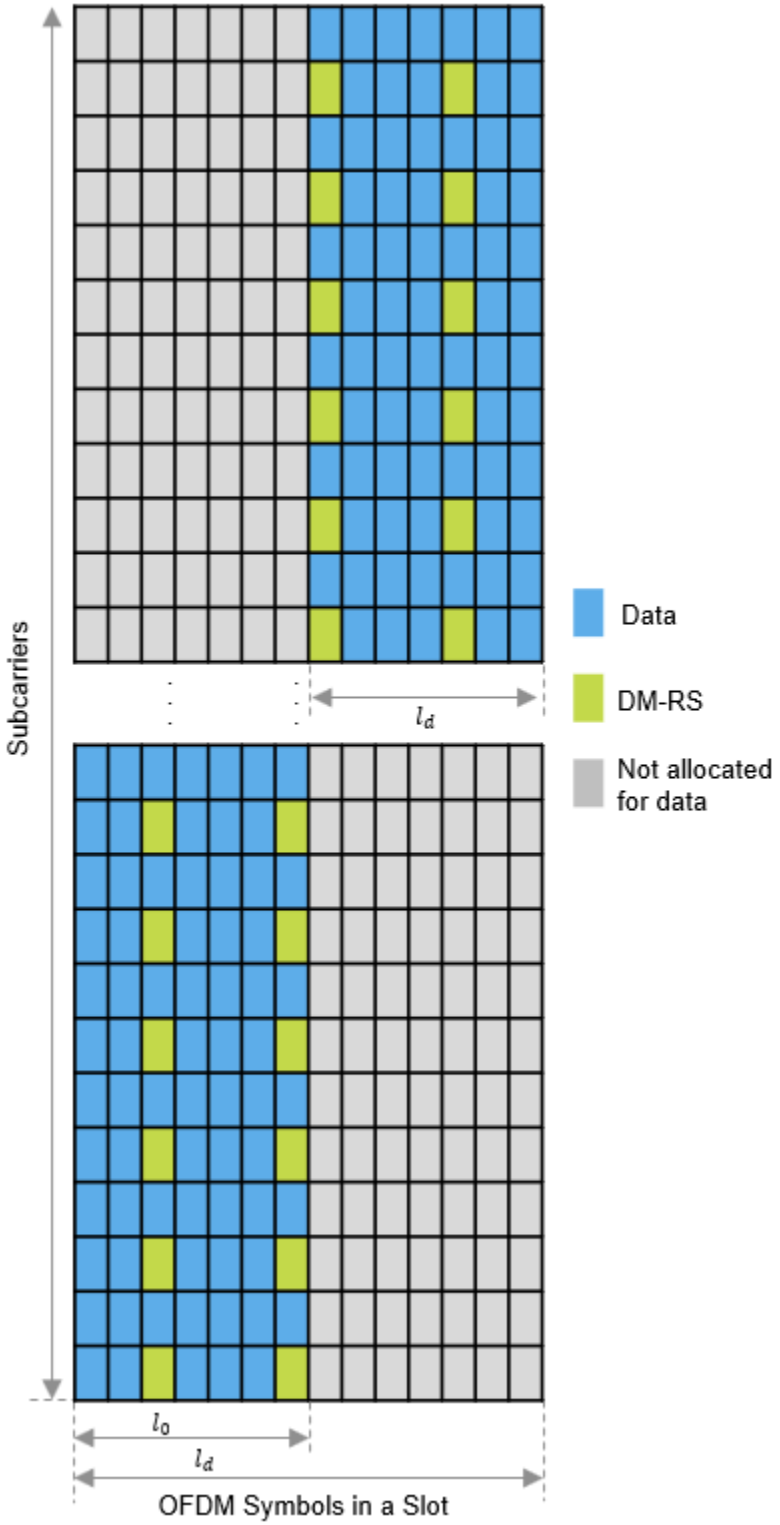


Figure 1: DM-RS Symbol Locations Based on Intra-Slot Frequency Hopping

For details on other DM-RS parameters, see “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-15.

```
% Assign intra-slot frequency hopping for PUSCH
pusch.FrequencyHopping = 'intraSlot'; % 'neither', 'intraSlot', 'interSlot'
pusch.SecondHopStartPRB = 26;

% Set the parameters that control the time resources of DM-RS
pusch.DMRS.DMRSTypeAPosition = 2;      % 2 or 3
pusch.DMRS.DMRSLength = 1;             % 1 or 2 (single-symbol or double-symbol)
pusch.DMRS.DMRSAdditionalPosition = 1; % 0...3 (Number of additional DM-RS positions)
```

Parameters That Control Frequency Resources

The parameters that control the frequency resources of DM-RS are:

- DM-RS configuration type
- DM-RS antenna ports

The configuration type indicates the frequency density of DM-RS and is signaled by the RRC message *dmrs-Type*. Configuration type 1 defines six subcarriers per physical resource block (PRB) per antenna port, comprising alternate subcarriers. Configuration type 2 defines four subcarriers per PRB per antenna port, consisting of two groups of two consecutive subcarriers. Different delta shifts are applied to the sets of subcarriers used, depending on the associated antenna port or code division multiplexing (CDM) group. For configuration type 1, there are two possible CDM groups/shifts across eight possible antenna ports (p=0...7). For configuration type 2, there are three possible CDM groups/shifts across twelve antenna ports (p=0...11). For more details, see “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-15.

In the case of codebook-based PUSCH processing, the union of DM-RS subcarrier locations present in each layer are projected to all the antenna ports.

```
% Set the parameters that control the frequency resources of DM-RS
pusch.DMRS.DMRSConfigurationType = 1; % 1 or 2
pusch.DMRS.DMRSPortSet = 0;

% The read-only properties DeltaShifts and DMRSSubcarrierLocations of DMRS
% property of pusch object provides the values of delta shift(s) and DM-RS
% subcarrier locations in an RB for each antenna port configured.
pusch.DMRS.DeltaShifts

ans = 0

pusch.DMRS.DMRSSubcarrierLocations

ans = 6×1

     0
     2
     4
     6
     8
    10
```

Sequence Generation

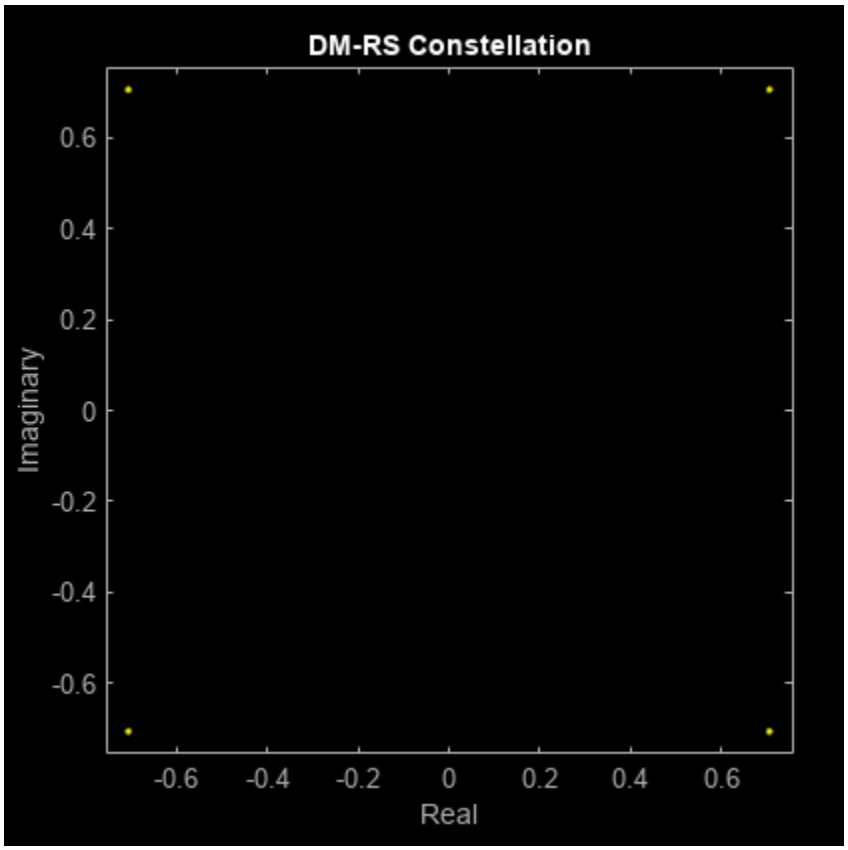
The pseudorandom sequence used for DM-RS is $2^{31} - 1$length gold sequence. The sequence is generated across all the common resource blocks (CRBs) and is transmitted only in the RBs allocated for data because the sequence is not required to estimate the channel outside the frequency region in which data is not transmitted. Generating the reference signal sequence across all the CRBs ensures that the same underlying pseudorandom sequence is used for multiple UEs on overlapping time-frequency resources in the case of a multi-user MIMO. The parameters that control the sequence generation are:

- DM-RS scrambling identity ($N_{\mathrm{ID}}^{n\mathrm{SCID}}$)
- DM-RS scrambling initialization (n_{SCID})
- Number of OFDM symbols in a slot
- Slot number in a radio frame
- DM-RS symbol locations
- PRBs allocation

The CyclicPrefix property of the carrier object controls the number of OFDM symbols in a slot. The NSlot property of the carrier object controls the slot number.

In the case of codebook-based PUSCH processing, the sequence is multiplied with a precoder matrix, which depends on the number of layers, number of antenna ports, and the transmitted precoder matrix indicator (TPMI).

```
% Set the parameters that only control the DM-RS sequence generation
pusch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pusch.DMRS.NSCID = 0;    % 0 or 1

% Generate DM-RS symbols
pusch.NumLayers = numel(pusch.DMRS.DMRSPortSet);
dmrsSymbols = nrPUSCHDMRS(carrier,pusch);

% Plot the constellation
scatterplot(dmrsSymbols)
title('DM-RS Constellation')
xlabel('Real')
ylabel('Imaginary')
```




```
% The read-only properties TimeWeights and FrequencyWeights of DMRS
% property of pusch object provides the values of time and frequency
% weights applied to the DM-RS symbols.
pusch.DMRS.TimeWeights
```

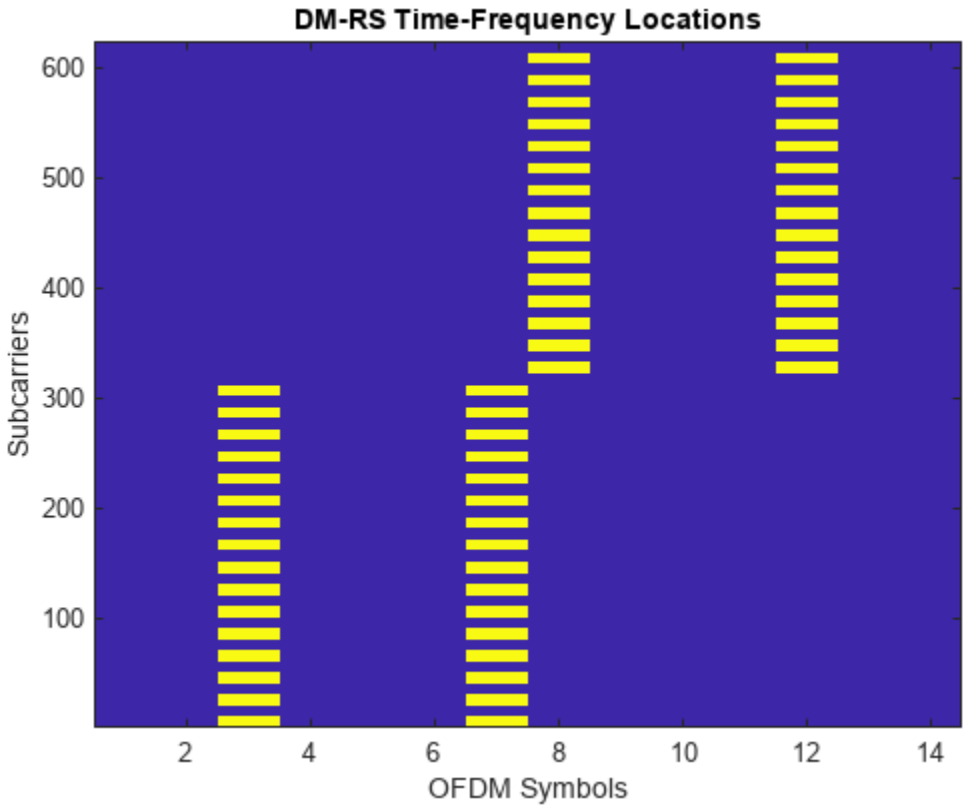
```
ans = 2×1

     1
     1
```

```
pusch.DMRS.FrequencyWeights
```

```
ans = 2×1

     1
     1
```

```
% Generate DM-RS indices
dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);

% Map the DM-RS symbols to the grid with the help of DM-RS indices
if strcmpi(pusch.TransmissionScheme,'codebook')
    nports = pusch.NumAntennaPorts;
else
    nports = pusch.NumLayers;
end
```

```
grid = zeros([12*carrier.NSizeGrid carrier.SymbolsPerSlot nports]);
grid(dmrsIndices) = dmrsSymbols;
figure
imagesc(abs(grid(:,:,1)));
axis xy;
xlabel('OFDM Symbols');
ylabel('Subcarriers');
title('DM-RS Time-Frequency Locations');
```



PT-RS for CP-OFDM

PT-RS is the phase tracking reference signal. PT-RS is used mainly to estimate and minimize the effect of CPE on system performance. Due to the phase noise properties, the PT-RS signal has low density in the frequency domain and high density in the time domain. PT-RS always occurs in combination with DM-RS and only when the network has configured PT-RS to be present.

Parameters That Control Time Resources

PT-RS is configured through the higher layer parameter *DMRS-UplinkConfig* for uplink. The parameters that control the time resources of PT-RS are:

- DM-RS symbol locations
- Time density of PT-RS (L_{PT-RS})

L_{PT-RS} depends on the scheduled modulation and coding scheme. The value must be one of {1, 2, 4}. For the parameters that control DM-RS symbol locations, refer to Parameters that Control DM-RS Time Resources (CP-OFDM) on page 2-26.

```
% Set the EnablePTRS property in pusch to 1
pusch.EnablePTRS = 1;

% Set the parameters that control the time resources of PT-RS
pusch.PTRS.TimeDensity = 2;
```

Parameters That Control Frequency Resources

PT-RS occupies only one subcarrier in an RB for one OFDM symbol. The parameters that control the frequency resources of PT-RS are:

- PRB allocation
- DM-RS configuration type
- Frequency density of PT-RS (K_{PT-RS})
- Radio network temporary identifier (n_{RNTI})
- Resource element offset
- PT-RS antenna ports

K_{PT-RS} depends on the scheduled bandwidth. The value is either 2 or 4. The value indicates whether PT-RS is present in every two RBs or every four RBs.

For more details, see “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-15.

```
% Set the parameters that control PT-RS subcarrier locations
pusch.RNTI = 1;
pusch.DMRS.DMRSConfigurationType = 1;
pusch.DMRS.DMRSPortSet = 0;
% Set the PT-RS parameters
pusch.PTRS.FrequencyDensity = 2; % 2 or 4
pusch.PTRS.REOffset = '10';      % '00', '01', '10', '11'
pusch.PTRS.PTRSPortSet = min(pusch.DMRS.DMRSPortSet);
```

Sequence Generation

The sequence used for generating PT-RS is the same pseudorandom sequence used for the DM-RS sequence generation. In the absence of intra-slot frequency hopping, the values of PT-RS sequence depend on the first DM-RS symbol position. In the presence of intra-slot frequency hopping, the values of PT-RS sequence depend on first DM-RS symbol positions in each hop. For more details, refer the section DM-RS Sequence Generation (CP-OFDM) on page 2-30.

In the case of codebook-based PUSCH processing, the sequence is multiplied with a precoder matrix, which depends on the number of layers, number of antenna ports, and the transmitted precoder matrix indicator (TPMI).

```
% Set the parameters that control the PT-RS sequence generation
pusch.DMRS.NIDNSCID = 1; % Use empty to set it to NCellID of the carrier
pusch.DMRS.NSCID = 0;    % 0 or 1
```

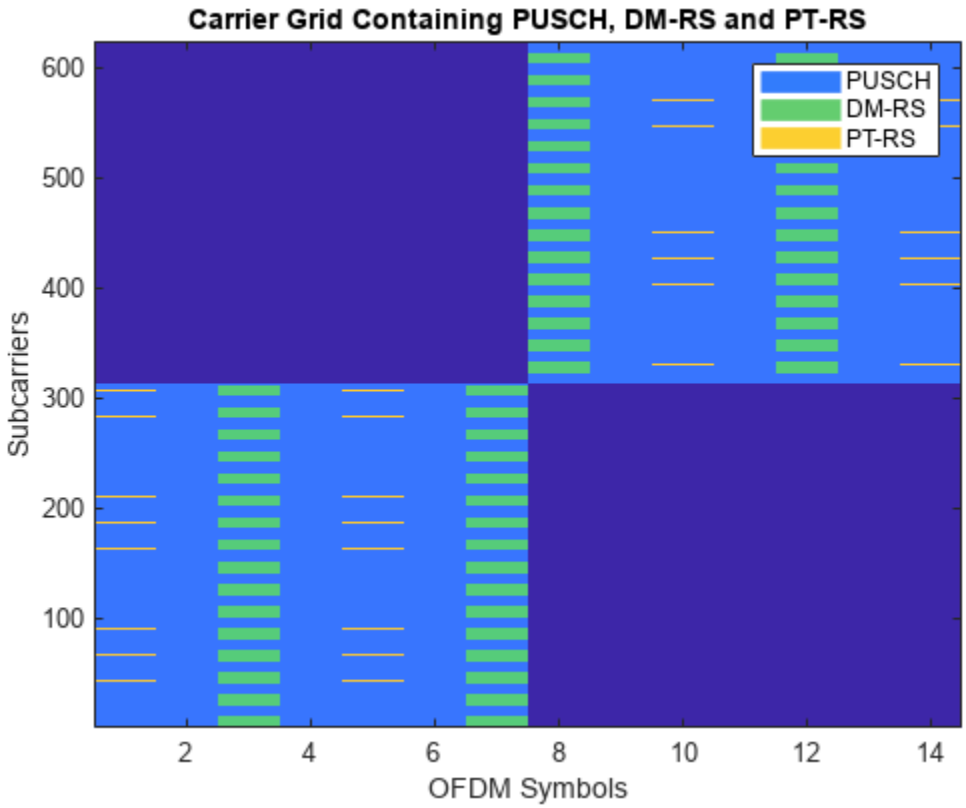
Generate the resource element (RE) indices of PUSCH, DM-RS, and PT-RS. Also, generate DM-RS and PT-RS symbols.

```
% Control the resource elements available for data in DM-RS OFDM symbol
% locations
```

```
pusch.DMRS.NumCDMGroupsWithoutData = 1;

% PUSCH, DM-RS and PT-RS indices
pusch.NumLayers = numel(pusch.DMRS.DMRSPortSet);
[puschIndices, puschInfo] = nrPUSCHIndices(carrier,pusch);
dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);
ptrsIndices = nrPUSCHPTRSIndices(carrier,pusch);

% DM-RS and PT-RS symbols
dmrsSymbols = nrPUSCHDMRS(carrier,pusch);
ptrsSymbols = nrPUSCHPTRS(carrier,pusch);
```

Map PUSCH, DM-RS, and PT-RS RE indices to the grid with scaled values to visualize the respective locations on the grid.

```
chpLevel = struct;
chpLevel.PUSCH = 0.4;
chpLevel.DMRS = 1;
chpLevel.PTRS = 1.4;
gridCPOFDM = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot nports]));
gridCPOFDM(puschIndices) = chpLevel.PUSCH;
dmrsFactor = chpLevel.DMRS*(1/(max(abs(dmrsSymbols))));
gridCPOFDM(dmrsIndices) = dmrsFactor*dmrsSymbols;
ptrsFactor = chpLevel.PTRS*(1/(max(abs(ptrsSymbols))));
gridCPOFDM(ptrsIndices) = ptrsFactor*ptrsSymbols;
plotGrid(gridCPOFDM,1,chpLevel)
```



In the preceding figure, PT-RS is located from the start of the OFDM symbol in the physical uplink shared channel allocation. The symbols are present at every L_{PT-RS} hop interval from each other or from DM-RS symbols. The difference in consecutive subcarrier locations of PT-RS is 24, which is the number of subcarriers in an RB (12) times the frequency density of PT-RS (2).

DM-RS for DFT-s-OFDM

DFT-s-OFDM supports only single layer transmission and is primarily used for low coverage scenarios. The time-frequency resources of DM-RS in DFT-s-OFDM are structured in a way to achieve low cubic metric and high power amplifier efficiency. The transmission of a reference signal frequency multiplexed with other uplink data transmissions highly impacts the power amplifier efficiency due to the increased cubic metric. The reference signals are time-multiplexed with uplink transmissions, thereby blocking all the resource elements for data transmission in the OFDM symbols carrying DM-RS.

Parameters That Control Time Resources

The parameters that control the time resources of DM-RS in DFT-s-OFDM are:

- PUSCH symbol allocation
- Mapping type
- Intra-slot frequency hopping
- DM-RS type A position
- DM-RS length
- DM-RS additional position

These parameters are the same parameters that control the time resources of DM-RS in CP-OFDM. For more details, refer to Parameters that Control DM-RS Time Resources (CP-OFDM) on page 2-26.

```
% Set the TransformPrecoding property in pusch to 1
pusch.TransformPrecoding = 1;

% Parameters that control the time resources
pusch.DMRS.DMRSTypeAPosition = 2;
pusch.DMRS.DMRSLength = 1;
pusch.DMRS.DMRSAdditionalPosition = 0;
```

Parameters That Control Frequency Resources

The parameters that control the frequency resources of DM-RS in DFT-s-OFDM are:

- DM-RS configuration type
- DM-RS antenna port

These two parameters are the same as the parameters of CP-OFDM. The DM-RS configuration type is always set to 1. The DM-RS antenna port is nominally a scalar with value 0.

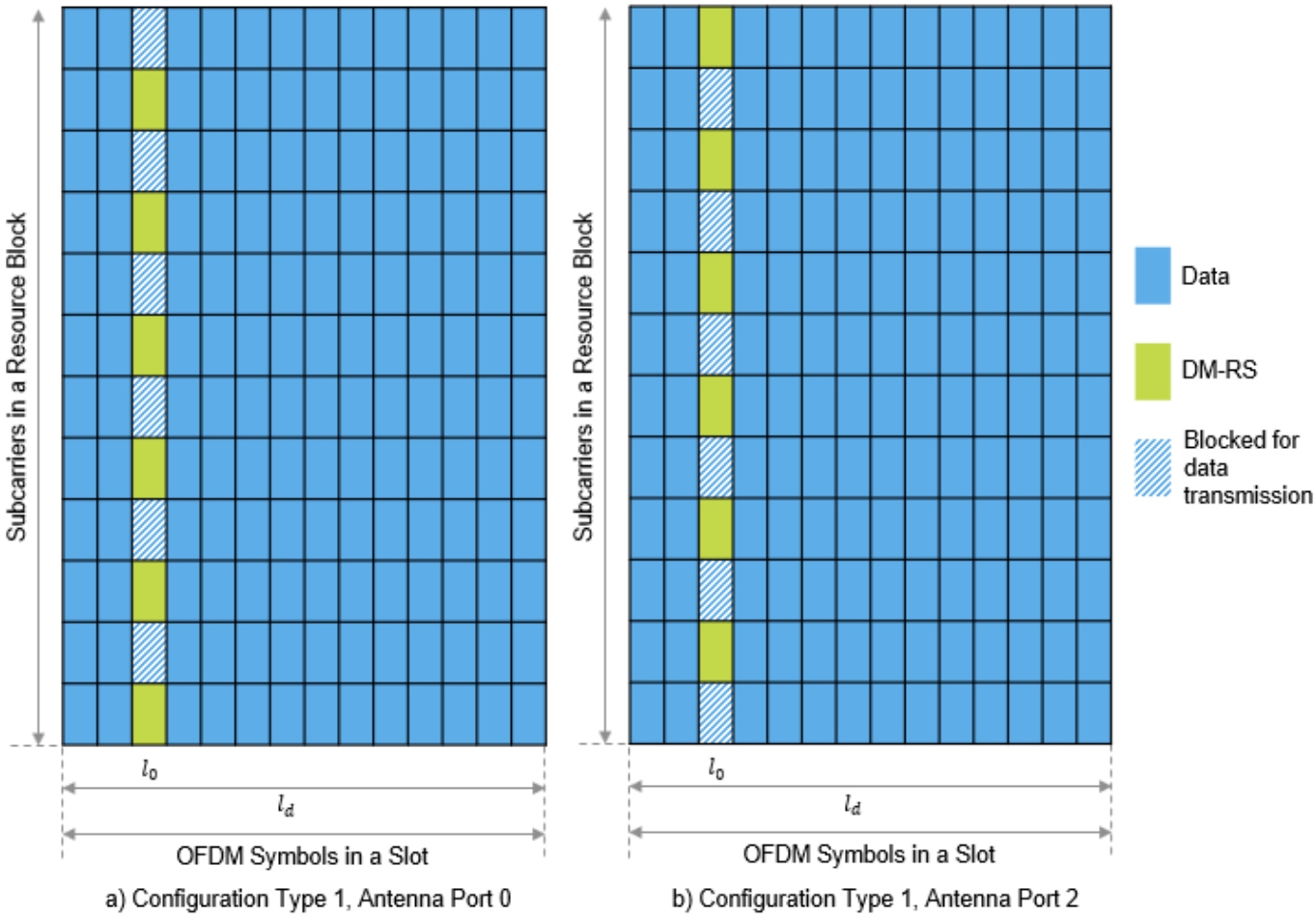


Figure 2: DM-RS Subcarrier Locations for DFT-s-OFDM

There is no need to support multi-user MIMO situations because DFT-s-OFDM is for coverage-limited scenarios. With no MIMO situations, the reference signal is generated only for the transmitted PRBs rather than CRBs as in OFDM. Due to the single layer and single configuration type allowed in DFT-s-OFDM, the number of subcarrier locations used for DM-RS in an RB is constant. Figure 2 illustrates the DM-RS subcarrier locations in DFT-s-OFDM for mapping type A with OFDM symbols allocated for PUSCH spanning over the complete slot.

```
% Set the DM-RS antenna port
pusch.DMRS.DMRSPortSet = 0;
```

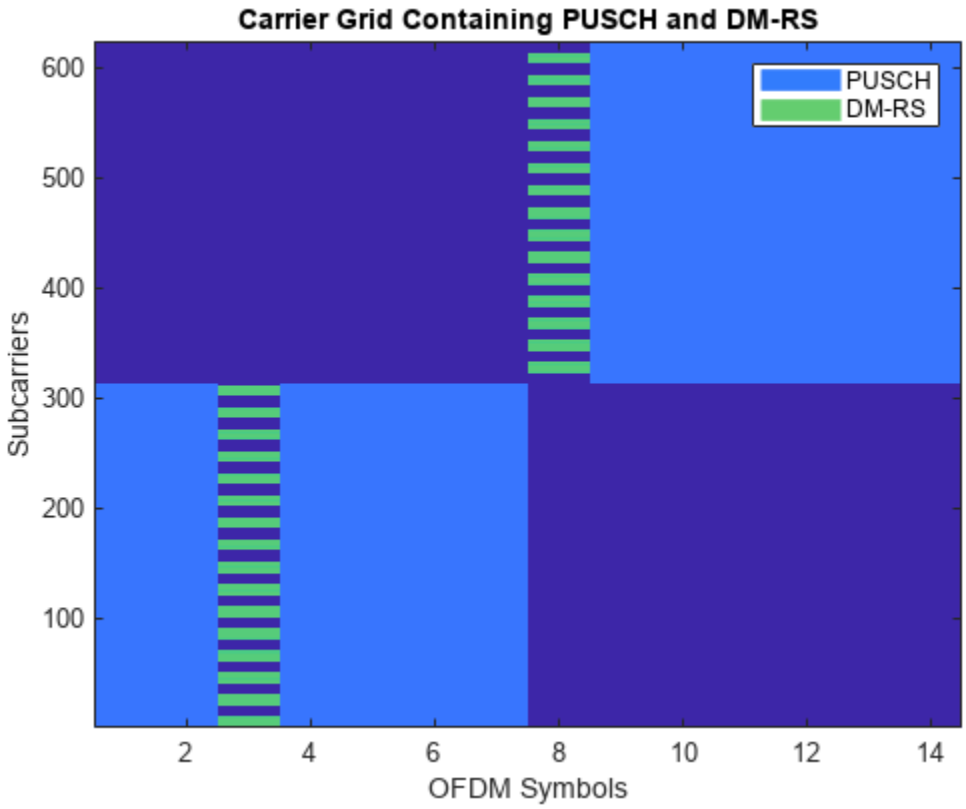
Sequence Generation

The DM-RS sequence is the ZadoffChu sequence in DFT-s-OFDM. The orthogonal sequences are generated with different cyclic shifts for a group number and sequence number. The parameters that control the sequence generation are:

- PRB allocation
- Group hopping
- Sequence hopping

- DM-RS scrambling identity (N_{ID}^{RS})
- DM-RS symbol locations

```
% Parameters that control the sequence generation
pusch.DMRS.SequenceHopping = 0; % Sequence hopping (0 or 1)
pusch.DMRS.GroupHopping = 1;    % Group hopping (0 or 1)
pusch.DMRS.NRSID = 1;           % Use empty to set it to NCellID of carrier

% Generate the DM-RS symbols and indices
pusch.NumLayers = numel(pusch.DMRS.DMRSPortSet);
dmrsSymbols = nrPUSCHDMRS(carrier,pusch);
dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);
dmrsFactor = chpLevel.DMRS*(1/(max(abs(dmrsSymbols))));
% Map DM-RS onto the grid
grid = complex(zeros([12*carrier.NSizeGrid carrier.SymbolsPerSlot nports]));
grid(dmrsIndices) = dmrsFactor*dmrsSymbols;

% Generate PUSCH indices and map onto the grid
puschIndices = nrPUSCHIndices(carrier,pusch);
grid(puschIndices) = chpLevel.PUSCH;

% Plot the grid
titleText = 'Carrier Grid Containing PUSCH and DM-RS';
plotGrid(grid,1,struct('PUSCH',chpLevel.PUSCH,'DMRS',chpLevel.DMRS),titleText,{'PUSCH','DM-RS'})
```



The subcarrier locations in the OFDM symbols occupying DM-RS are not allocated for PUSCH.

PT-RS for DFT-s-OFDM

PT-RS in DFT-s-OFDM is inserted with data in the transform precoding stage.

Parameters That Control Time Resources

The parameters that control the time resources of PT-RS in DFT-s-OFDM are same as the parameters that control the time resources of PT-RS in CP-OFDM. The value of $L_{\mathrm{PT-RS}}$ is either 1 or 2 in DFT-s-OFDM. For more details, refer to Parameters that Control PT-RS Time Resources (CP-OFDM) on page 2-32.

```
% Generate a grid with shared channel allocation for an RB in a single slot
% with complete symbol allocation of 14 symbols for a single layer

% Set the carrier resource grid with one RB
carrier.NSizeGrid = 1;

% Configure PUSCH with DFT-s-OFDM and no frequency hopping
pusch.TransformPrecoding = 1;
pusch.FrequencyHopping = 'neither';

% Set the parameter that control PT-RS time resources
pusch.EnablePTRS = 1;
pusch.PTRS.TimeDensity = 2;
```

Parameters That Control Frequency Resources

The PT-RS pattern in the frequency domain is quite different from CP-OFDM. The PT-RS samples are inserted as chunks or groups ($N_{\mathrm{group}}^{\mathrm{PT-RS}}$). Each group consists of a finite number of samples ($N_{\mathrm{samp}}^{\mathrm{group}}$) in the scheduled bandwidth for each OFDM symbol where PT-RS is present.

The parameters that control the frequency resources of PT-RS in DFT-s-OFDM are:

- PRB allocation
- Number of PT-RS samples in a group ($N_{\mathrm{samp}}^{\mathrm{group}}$)
- Number of PT-RS groups ($N_{\mathrm{group}}^{\mathrm{PT-RS}}$)

The valid combinations of PT-RS sample density ($[N_{\mathrm{samp}}^{\mathrm{group}} N_{\mathrm{group}}^{\mathrm{PT-RS}}]$) are {[2 2], [2 4], [4 2], [4 4], [4 8]}. The number of PT-RS samples in an OFDM symbol is fixed in DFT-s-OFDM, based on the number of PT-RS samples in all the PT-RS groups. This number is different from CP-OFDM in which the number of PT-RS samples increase based on the number of RBs in PUSCH.

Figure 3 shows the subcarrier locations of PT-RS symbols for an RB with the number of PT-RS samples set to 2 and the number of PT-RS groups set to 2 for an OFDM symbol carrying PT-RS.

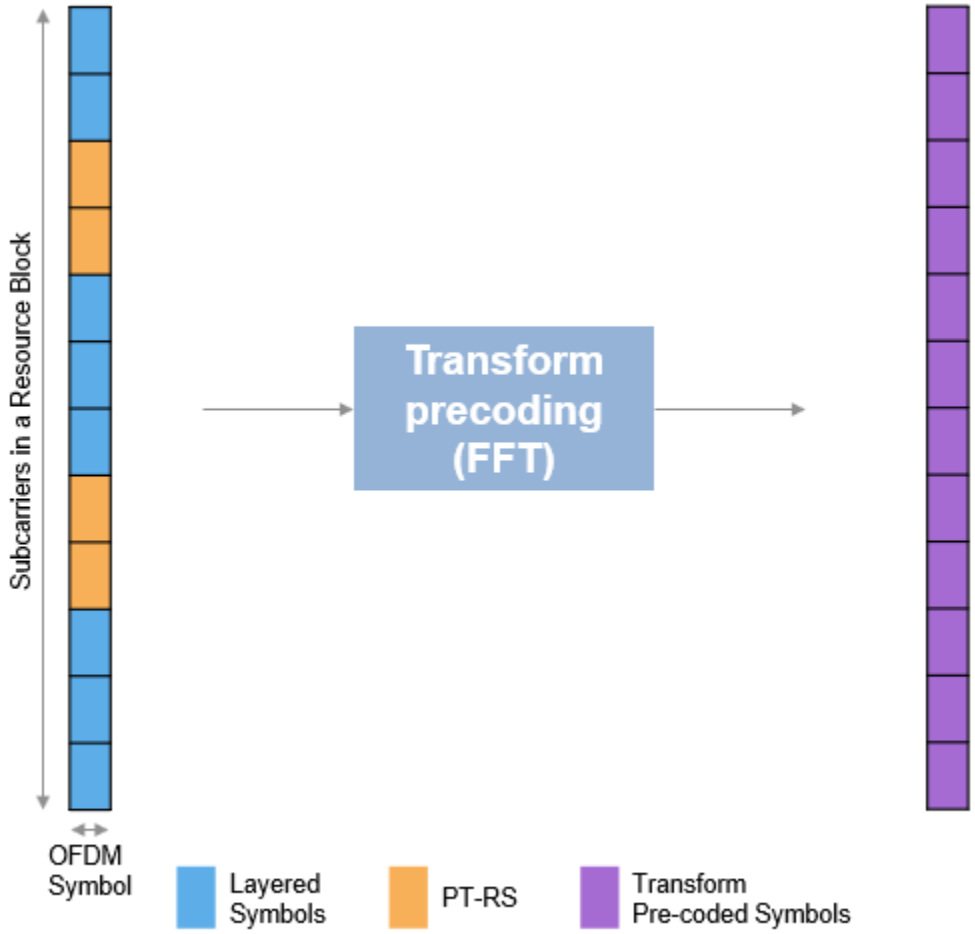


Figure 3: PT-RS Insertion in DFT-s-OFDM

PT-RS sample density [2 2] implies that there are two PT-RS groups in a scheduled bandwidth with two symbols each.

PT-RS is inserted with layered symbols at the input of transform precoding. After transform precoding, both layered symbols and PT-RS are treated as data. Therefore, the PT-RS is not visible in the grid directly.

```
% Set the parameters that control PT-RS frequency resources
pusch.PRBSet = 0:carrier.NSizeGrid-1;
pusch.PTRS.NumPTRSSamples = 2; % 2, 4
pusch.PTRS.NumPTRSGroups = 2;  % 2, 4, 8
```

Sequence Generation

The PT-RS sequence in DFT-s-OFDM is a modified pi/2-BPSK sequence. The parameters that control the sequence generation are:

- Starting OFDM symbol of PUSCH allocation
- Number of OFDM symbols in a slot

- Slot number in a radio frame
- PT-RS scrambling identity (N_{ID})
- PT-RS subcarrier locations

```
% Set the parameters that control PT-RS sequence generation
pusch.DMRS.NRSID = 1;
pusch.PTRS.NID = 10; % Use empty to set it to NRSID of DMRS configuration
```

Generate PUSCH and PT-RS RE indices.

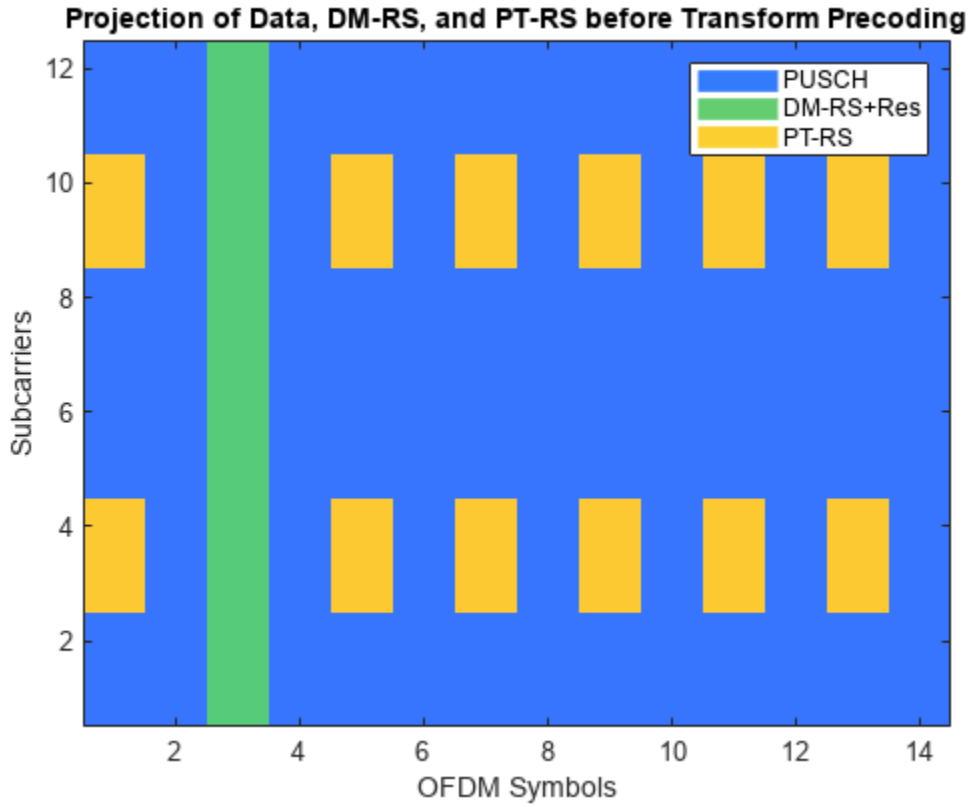
```
% PUSCH, and PT-RS indices
[puschIndices, puschInfoDFTsOFDM] = nrPUSCHIndices(carrier,pusch);
ptrsIndices = nrPUSCHPTRSIndices(carrier,pusch);
```

Set PUSCH and PT-RS resource elements to constant values.

```
% Insert PT-RS along with the PUSCH data
GdPTRS = size(reshape(ptrsIndices,[],pusch.NumLayers),1);
dataWithPTRS = chpLevel.PUSCH*ones(puschInfoDFTsOFDM.Gd+GdPTRS,1);
dataWithPTRS(ptrsIndices(:,1)) = chpLevel.PTRS;
```

Plot PT-RS projections onto the grid.

```
gridDFTsOFDM = zeros(numel(pusch.PRBSet)*12, carrier.SymbolsPerSlot);
% Map the grid with data and reference signals
gridDFTsOFDM(:,puschInfoDFTsOFDM.DMRSSymbolSet+1) = chpLevel.DMRS;
gridDFTsOFDM(~(gridDFTsOFDM==chpLevel.DMRS)) = dataWithPTRS;

% Plot the projections of data, DM-RS and PT-RS on grid before transform
% precoding
fNames = {'PUSCH','DM-RS+Res','PT-RS'};
titleText = 'Projection of Data, DM-RS, and PT-RS before Transform Precoding';
plotGrid(gridDFTsOFDM,1,chpLevel,titleText,fNames)
```



Further Exploration

You can try changing the parameters that affect the time and frequency resources of reference signals and observe the variations in the RE positions for the respective signals.

Try changing the number of antenna ports configured for DM-RS and PT-RS, then observe the variations of reference signals and data across the ports. For example, try to configure DM-RS for two antenna ports 0 and 2, configuration type 1, and PT-RS for antenna port 0. Generate the PUSCH indices, DM-RS signal (indices and symbols), and PT-RS signal (indices and symbols). Map them to a grid and visualize the grid for both the ports.

Try performing channel estimation and phase tracking using the PT-RS symbols and indices. Compute the throughput by following the steps outlined in “NR PUSCH Throughput” on page 2-43.

This example shows how to generate the DM-RS and PT-RS sequences and how to map the sequences to the OFDM carrier resource grid. It highlights the properties that control the time-frequency structure of reference signals for different waveforms. For example, the time-frequency pattern for reference signals in CP-OFDM and DFT-s-OFDM and the variation in the sequences generated for reference signals in different waveforms.

References

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

2 3GPP TS 38.214. "NR; Physical layer procedures for data" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

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

Local Functions

```
function plotGrid(grid,nLayer,chpLevel,titleText,names)
% plotGrid Display resource grid GRID of the layer number NLAYER with the
% legend containing physical channel and associated reference signals, at
% different power levels CHPLEVEL with title TITLETEXT. Legend is created
% using a cell array of character vectors NAMES.

    if nargin < 4
        titleText = 'Carrier Grid Containing PUSCH, DM-RS and PT-RS';
    end

    if nargin < 5
        names = {'PUSCH', 'DM-RS', 'PT-RS'};
    end

    map = parula(64);
    cscaling = 40;
    im = image(1:size(grid,2),1:size(grid,1),cscaling*abs(grid(:,:,nLayer)));
    colormap(im.Parent,map);

    % Add legend to the image
    chpval = struct2cell(chpLevel);
    clevels = cscaling*[chpval{:}];
    N = length(clevels);
    L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
    % Index the color map and associated the selected colors with the lines
    set(L,{'color'},mat2cell(map( min(1+clevels,length(map) ),:),ones(1,N),3)); % Set the colors
    % Create legend
    legend(names{:});
    axis xy;
    ylabel('Subcarriers');
    xlabel('OFDM Symbols');
    title(titleText);
end
```

See Also

Functions

`nrPUSCHDMRS` | `nrPUSCHDMRSIndices` | `nrPUSCHPTRS` | `nrPUSCHPTRSIndices` | `nrPUSCHIndices`

Objects

`nrPUSCHConfig` | `nrCarrierConfig`

Related Examples

- “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-15

NR PUSCH Throughput

This reference simulation shows how to measure the physical uplink shared channel (PUSCH) throughput of a 5G New Radio (NR) link, as defined by the 3GPP NR standard. The example implements PUSCH and uplink transport channel (UL-SCH). The transmitter model includes PUSCH demodulation reference signals (DM-RS). The example supports both clustered delay line (CDL) and tapped delay line (TDL) propagation channels. You can perform perfect or practical synchronization and channel estimation. To reduce the total simulation time, you can execute the SNR points in the SNR loop in parallel by using the Parallel Computing Toolbox™.

Introduction

This example measures the PUSCH throughput of a 5G link, as defined by the 3GPP NR standard [1], [2], [3], [4].

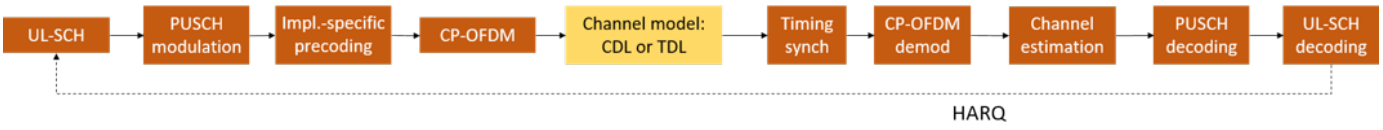
The example models these 5G NR features:

- UL-SCH transport channel coding
- PUSCH and PUSCH DM-RS generation
- Variable subcarrier spacing and frame numerologies (2^n * 15 kHz)
- Normal and extended cyclic prefix
- TDL and CDL propagation channel models

Other features of the simulation are:

- Codebook and non-codebook based PUSCH transmission schemes
- Optional PUSCH transform precoding
- Slot wise and non slot wise PUSCH and DM-RS mapping
- Perfect or practical synchronization and channel estimation
- HARQ operation with 16 processes

The figure shows the implemented processing chain. For clarity, the DM-RS generation is omitted.



Note that this example does not include closed-loop adaptation of the MIMO precoding according to channel conditions. The PUSCH MIMO precoding used in the example is as follows:

- For codebook based transmission, the MIMO precoding matrix used inside the PUSCH modulation can be selected using the TPMI parameter.
- The implementation-specific MIMO precoding matrix (for non-codebook based transmission, or MIMO precoding between transmission antenna ports and antennas for codebook based transmission) is an identity matrix.

To reduce the total simulation time, you can use the Parallel Computing Toolbox to execute the SNR points of the SNR loop in parallel.

Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10ms frames. A large number of NFrames should be used to produce meaningful throughput results. Set the SNR points to simulate. The SNR is defined per RE and applies to each receive antenna. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
simParameters = struct();       % Clear simParameters variable to contain all key simulation para
simParameters.NFrames = 2;      % Number of 10 ms frames
simParameters.SNRIn = [-5 0 5]; % SNR range (dB)
```

Channel Estimator Configuration

The logical variable `PerfectChannelEstimator` controls channel estimation and synchronization behavior. When set to `true`, perfect channel estimation and synchronization is used. Otherwise, practical channel estimation and synchronization is used, based on the values of the received PUSCH DM-RS.

```
simParameters.PerfectChannelEstimator = true;
```

Simulation Diagnostics

The variable `DisplaySimulationInformation` controls the display of simulation information such as the HARQ process ID used for each subframe. In case of CRC error, the value of the index to the RV sequence is also displayed.

```
simParameters.DisplaySimulationInformation = true;
```

The `DisplayDiagnostics` flag enables the plotting of the EVM per layer. This plot monitors the quality of the received signal after equalization. The EVM per layer figure shows:

- The EVM per layer per slot, which shows the EVM evolving with time.
- The EVM per layer per resource block, which shows the EVM in frequency.

This figure evolves with the simulation and is updated with each slot. Typically, low SNR or channel fades can result in decreased signal quality (high EVM). The channel affects each layer differently, therefore, the EVM values may differ across layers.

In some cases, some layers can have a much higher EVM than others. These low-quality layers can result in CRC errors. This behavior may be caused by low SNR or by using too many layers for the channel conditions. You can avoid this situation by a combination of higher SNR, lower number of layers, higher number of antennas, and more robust transmission (lower modulation scheme and target code rate).

```
simParameters.DisplayDiagnostics = false;
```

Carrier and PUSCH Configuration

Set the key parameters of the simulation. These include:

- The bandwidth in resource blocks (12 subcarriers per resource block)
- Subcarrier spacing: 15, 30, 60, 120 (kHz)
- Cyclic prefix length: normal or extended
- Cell ID

- Number of transmit and receive antennas

A substructure containing the UL-SCH and PUSCH parameters is also specified. This includes:

- Target code rate
- Allocated resource blocks (PRBSet)
- Modulation scheme: 'pi/2-BPSK', 'QPSK', '16QAM', '64QAM', '256QAM'
- Number of layers
- Transform precoding (enable/disable)
- PUSCH transmission scheme and MIMO precoding matrix indication (TPMI)
- Number of antenna ports
- PUSCH mapping type
- DM-RS configuration parameters

Other simulation wide parameters are:

- Propagation channel model delay profile (TDL or CDL)

Note that if transform precoding is enabled, the number of layers should be set to 1.

```
% Set waveform type and PUSCH numerology (SCS and CP type)
simParameters.Carrier = nrCarrierConfig;        % Carrier resource grid configuration
simParameters.Carrier.NSizeGrid = 52;           % Bandwidth in number of resource blocks (52 RBs
simParameters.Carrier.SubcarrierSpacing = 15;   % 15, 30, 60, 120 (kHz)
simParameters.Carrier.CyclicPrefix = 'Normal';  % 'Normal' or 'Extended' (Extended CP is relevant
simParameters.Carrier.NCellID = 0;              % Cell identity

% PUSCH/UL-SCH parameters
simParameters.PUSCH = nrPUSCHConfig;      % This PUSCH definition is the basis for all PUSCH tran
simParameters.PUSCHExtension = struct();  % This structure is to hold additional simulation param

% Define PUSCH time-frequency resource allocation per slot to be full grid (single full grid BWP)
simParameters.PUSCH.PRBSet =  0:simParameters.Carrier.NSizeGrid-1; % PUSCH PRB allocation
simParameters.PUSCH.SymbolAllocation = [0,simParameters.Carrier.SymbolsPerSlot]; % PUSCH symbol a
simParameters.PUSCH.MappingType = 'A'; % PUSCH mapping type ('A'(slot-wise),'B'(non slot-wise))

% Scrambling identifiers
simParameters.PUSCH.NID = simParameters.Carrier.NCellID;
simParameters.PUSCH.RNTI = 1;

% Define the transform precoding enabling, layering and transmission scheme
simParameters.PUSCH.TransformPrecoding = false; % Enable/disable transform precoding
simParameters.PUSCH.NumLayers = 1;              % Number of PUSCH transmission layers
simParameters.PUSCH.TransmissionScheme = 'nonCodebook'; % Transmission scheme ('nonCodebook','cod
simParameters.PUSCH.NumAntennaPorts = 1;        % Number of antenna ports for codebook based prec
simParameters.PUSCH.TPMI = 0;                   % Precoding matrix indicator for codebook based p

% Define codeword modulation
simParameters.PUSCH.Modulation = 'QPSK'; % 'pi/2-BPSK', 'QPSK', '16QAM', '64QAM', '256QAM'

% PUSCH DM-RS configuration
simParameters.PUSCH.DMRS.DMRSTypeAPosition = 2;       % Mapping type A only. First DM-RS symbol p
simParameters.PUSCH.DMRS.DMRSLength = 1;              % Number of front-loaded DM-RS symbols (1(s
simParameters.PUSCH.DMRS.DMRSAdditionalPosition = 1;  % Additional DM-RS symbol positions (max ra
```

```
simParameters.PUSCH.DMRS.DMRSConfigurationType = 1;   % DM-RS configuration type (1,2)
simParameters.PUSCH.DMRS.NumCDMGroupsWithoutData = 2; % Number of CDM groups without data
simParameters.PUSCH.DMRS.NIDNSCID = 0;                % Scrambling identity (0...65535)
simParameters.PUSCH.DMRS.NSCID = 0;                   % Scrambling initialization (0,1)
simParameters.PUSCH.DMRS.NRSID = 0;                   % Scrambling ID for low-PAPR sequences (0.
simParameters.PUSCH.DMRS.GroupHopping = 0;            % Group hopping (0,1)
simParameters.PUSCH.DMRS.SequenceHopping = 0;         % Sequence hopping (0,1)

% Additional simulation and UL-SCH related parameters
%
% Target code rate
simParameters.PUSCHExtension.TargetCodeRate = 193 / 1024; % Code rate used to calculate transport
%
% HARQ process and rate matching/TBS parameters
simParameters.PUSCHExtension.XOverhead = 0;       % Set PUSCH rate matching overhead for TBS (Xoh
simParameters.PUSCHExtension.NHARQProcesses = 16; % Number of parallel HARQ processes to use
simParameters.PUSCHExtension.EnableHARQ = true;   % Enable retransmissions for each process, usin

% LDPC decoder parameters
% Available algorithms: 'Belief propagation', 'Layered belief propagation', 'Normalized min-sum',
simParameters.PUSCHExtension.LDPCDecodingAlgorithm = 'Normalized min-sum';
simParameters.PUSCHExtension.MaximumLDPCIterationCount = 6;

% Define the overall transmission antenna geometry at end-points
% If using a CDL propagation channel then the integer number of antenna elements is
% turned into an antenna panel configured when the channel model object is created
simParameters.NTxAnts = 1; % Number of transmit antennas
simParameters.NRxAnts = 2; % Number of receive antennas

% Define the general CDL/TDL propagation channel parameters
simParameters.DelayProfile = 'TDL-A'; % Use TDL-A model (Indoor hotspot model)
simParameters.DelaySpread = 30e-9;
simParameters.MaximumDopplerShift = 10;

% Cross-check the PUSCH layering against the channel geometry
validateNumLayers(simParameters);
```

The simulation relies on various pieces of information about the baseband waveform, such as sample rate.

```
waveformInfo = nrOFDMInfo(simParameters.Carrier); % Get information about the baseband waveform a
```

Propagation Channel Model Construction

Create the channel model object for the simulation. Both CDL and TDL channel models are supported [5].

```
% Constructed the CDL or TDL channel model object
if contains(simParameters.DelayProfile,'CDL','IgnoreCase',true)

    channel = nrCDLChannel; % CDL channel object

    % Turn the number of antennas into antenna panel array layouts. If
    % NRxAnts is not one of (1,2,4,8,16,32,64,128,256,512,1024), its value
    % is rounded up to the nearest value in the set. If NTxAnts is not 1 or
    % even, its value is rounded up to the nearest even number.
    channel = hArrayGeometry(channel,simParameters.NTxAnts,simParameters.NRxAnts,'uplink');
    simParameters.NTxAnts = prod(channel.TransmitAntennaArray.Size);
```

```
    simParameters.NRxAnts = prod(channel.ReceiveAntennaArray.Size);

    % Configure antenna elements
    channel.TransmitAntennaArray.Element = 'isotropic';
    channel.ReceiveAntennaArray.Element = '38.901';
else
    channel = nrTDLChannel; % TDL channel object

    % Set the channel geometry
    channel.NumTransmitAntennas = simParameters.NTxAnts;
    channel.NumReceiveAntennas = simParameters.NRxAnts;
end

% Assign simulation channel parameters and waveform sample rate to the object
channel.DelayProfile = simParameters.DelayProfile;
channel.DelaySpread = simParameters.DelaySpread;
channel.MaximumDopplerShift = simParameters.MaximumDopplerShift;
channel.SampleRate = waveformInfo.SampleRate;
```

Get the maximum number of delayed samples by a channel multipath component. This is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This is required later to flush the channel filter to obtain the received signal.

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

Processing Loop

To determine the throughput at each SNR point, the PUSCH data is analyzed per transmission instance using the following steps:

- *Update current HARQ process.* Check the transmission status for the given HARQ process to determine whether a retransmission is required. If that is not the case then generate new data.
- *Generate resource grid.* Channel coding is performed by `nrULSCH`. It operates on the input transport block provided. Internally, it keeps a copy of the transport block in case a retransmission is required. The coded bits are modulated by `nrPUSCH`. Implementation-specific MIMO precoding is applied to the resulting signal. Note that if `TxScheme='codebook'`, codebook based MIMO precoding will already have been applied inside `nrPUSCH` and the implementation-specific MIMO precoding is an additional stage of MIMO precoding.
- *Generate waveform.* The generated grid is then OFDM modulated.
- *Model noisy channel.* The waveform is passed through a CDL or TDL fading channel. AWGN is added. The SNR for each layer is defined per RE and per receive antenna.
- *Perform synchronization and OFDM demodulation.* For perfect synchronization, the channel impulse response is reconstructed and used to synchronize the received waveform. For practical synchronization, the received waveform is correlated with the PUSCH DM-RS. The synchronized signal is then OFDM demodulated.
- *Perform channel estimation.* If perfect channel estimation is used, the channel impulse response is reconstructed and OFDM demodulated to provide a channel estimate. For practical channel estimation, the PUSCH DM-RS is used.
- *Extract PUSCH and perform equalization.* The resource elements corresponding to the PUSCH allocation are extracted from the received OFDM resource grid and the channel estimate using `nrExtractResources`. The received PUSCH resource elements are then MMSE equalized using `nrEqualizeMMSE`.

- *Decode the PUSCH.* The equalized PUSCH symbols, along with a noise estimate, are demodulated and descrambled by `nrPUSCHDecode` to obtain an estimate of the received codewords.
- *Decode the Uplink Shared Channel (UL-SCH) and update HARQ process with the block CRC error.* The vector of decoded soft bits is passed to `nrULSCHDecoder` which decodes the codeword and returns the block CRC error used to determine the throughput of the system.

```
% Array to store the maximum throughput for all SNR points
maxThroughput = zeros(length(simParameters.SNRIn),1);
% Array to store the simulation throughput for all SNR points
simThroughput = zeros(length(simParameters.SNRIn),1);

% Set up redundancy version (RV) sequence for all HARQ processes
if simParameters.PUSCHExtension.EnableHARQ
    % From PUSCH demodulation requirements in RAN WG4 meeting #88bis (R4-1814062)
    rvSeq = [0 2 3 1];
else
    % HARQ disabled - single transmission with RV=0, no retransmissions
    rvSeq = 0;
end

% Create UL-SCH encoder System object to perform transport channel encoding
encodeULSCH = nrULSCH;
encodeULSCH.MultipleHARQProcesses = true;
encodeULSCH.TargetCodeRate = simParameters.PUSCHExtension.TargetCodeRate;

% Create UL-SCH decoder System object to perform transport channel decoding
% Use layered belief propagation for LDPC decoding, with half the number of
% iterations as compared to the default for belief propagation decoding
decodeULSCH = nrULSCHDecoder;
decodeULSCH.MultipleHARQProcesses = true;
decodeULSCH.TargetCodeRate = simParameters.PUSCHExtension.TargetCodeRate;
decodeULSCH.LDPCDecodingAlgorithm = simParameters.PUSCHExtension.LDPCDecodingAlgorithm;
decodeULSCH.MaximumLDPCIterationCount = simParameters.PUSCHExtension.MaximumLDPCIterationCount;

for snrIdx = 1:numel(simParameters.SNRIn)    % comment out for parallel computing
% parfor snrIdx = 1:numel(simParameters.SNRIn) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'
% statement and uncomment the 'parfor' statement. If the Parallel Computing
% Toolbox is not installed, 'parfor' defaults to normal 'for' statement.
% Because parfor-loop iterations are executed in parallel in a
% nondeterministic order, the simulation information displayed for each SNR
% point can be intertwined. To switch off simulation information display,
% set the 'displaySimulationInformation' variable above to false

    % Reset the random number generator so that each SNR point will
    % experience the same noise realization
    rng('default');

    % Take full copies of the simulation-level parameter structures so that they are not
    % PCT broadcast variables when using parfor
    simLocal = simParameters;
    waveinfoLocal = waveformInfo;

    % Take copies of channel-level parameters to simplify subsequent parameter referencing
    carrier = simLocal.Carrier;
    pusch = simLocal.PUSCH;
```

```
puschextra = simLocal.PUSCHExtension;
decodeULSCHLocal = decodeULSCH;  % Copy of the decoder handle to help PCT classification of v
decodeULSCHLocal.reset();        % Reset decoder at the start of each SNR point
pathFilters = [];

% Create PUSCH object configured for the non-codebook transmission
% scheme, used for receiver operations that are performed with respect
% to the PUSCH layers
puschNonCodebook = pusch;
puschNonCodebook.TransmissionScheme = 'nonCodebook';

% Prepare simulation for new SNR point
SNRdB = simLocal.SNRIn(snrIdx);
fprintf('\nSimulating transmission scheme 1 (%dx%d) and SCS=%dkHz with %s channel at %gdB SNR
    simLocal.NTxAnts,simLocal.NRxAnts,carrier.SubcarrierSpacing, ...
    simLocal.DelayProfile,SNRdB,simLocal.NFrames);

% Specify the fixed order in which we cycle through the HARQ process IDs
harqSequence = 0:puschextra.NHARQProcesses-1;

% Initialize the state of all HARQ processes
harqEntity = HARQEntity(harqSequence,rvSeq);

% Reset the channel so that each SNR point will experience the same
% channel realization
reset(channel);

% Total number of slots in the simulation period
NSlots = simLocal.NFrames * carrier.SlotsPerFrame;

% Timing offset, updated in every slot for perfect synchronization and
% when the correlation is strong for practical synchronization
offset = 0;

% Loop over the entire waveform length
for nslot = 0:NSlots-1

    % Update the carrier slot numbers for new slot
    carrier.NSlot = nslot;

    % Calculate the transport block size for the transmission in the slot
    [puschIndices,puschIndicesInfo] = nrPUSCHIndices(carrier,pusch);
    MRB = numel(pusch.PRBSet);
    trBlkSize = nrTBS(pusch.Modulation,pusch.NumLayers,MRB,puschIndicesInfo.NREPerPRB,puschex

    % HARQ processing
    % If new data for current process then create a new UL-SCH transport block
    if harqEntity.NewData
        trBlk = randi([0 1],trBlkSize,1);
        setTransportBlock(encodeULSCH,trBlk,harqEntity.HARQProcessID);
        % If new data because of previous RV sequence time out then flush decoder soft buffe
        if harqEntity.SequenceTimeout
            resetSoftBuffer(decodeULSCHLocal,harqEntity.HARQProcessID);
        end
    end

    % Encode the UL-SCH transport block
    codedTrBlock = encodeULSCH(pusch.Modulation,pusch.NumLayers, ...
```

```
    puschIndicesInfo.G,harqEntity.RedundancyVersion,harqEntity.HARQProcessID);

% Create resource grid for a slot
puschGrid = nrResourceGrid(carrier,simLocal.NTxAnts);

% PUSCH modulation, including codebook based MIMO precoding if TxScheme = 'codebook'
puschSymbols = nrPUSCH(carrier,pusch,codedTrBlock);

% Implementation-specific PUSCH MIMO precoding and mapping. This
% MIMO precoding step is in addition to any codebook based
% MIMO precoding done during PUSCH modulation above
if (strcmpi(pusch.TransmissionScheme,'codebook'))
    % Codebook based MIMO precoding, F precodes between PUSCH
    % transmit antenna ports and transmit antennas
    F = eye(pusch.NumAntennaPorts,simLocal.NTxAnts);
else
    % Non-codebook based MIMO precoding, F precodes between PUSCH
    % layers and transmit antennas
    F = eye(pusch.NumLayers,simLocal.NTxAnts);
end
[~,puschAntIndices] = nrExtractResources(puschIndices,puschGrid);
puschGrid(puschAntIndices) = puschSymbols * F;

% Implementation-specific PUSCH DM-RS MIMO precoding and mapping.
% The first DM-RS creation includes codebook based MIMO precoding if applicable
dmrsSymbols = nrPUSCHDMRS(carrier,pusch);
dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);
for p = 1:size(dmrsSymbols,2)
    [~,dmrsAntIndices] = nrExtractResources(dmrsIndices(:,p),puschGrid);
    puschGrid(dmrsAntIndices) = puschGrid(dmrsAntIndices) + dmrsSymbols(:,p) * F(p,:);
end

% OFDM modulation
txWaveform = nrOFDMModulate(carrier,puschGrid);

% Pass data through channel model. Append zeros at the end of the
% transmitted waveform to flush channel content. These zeros take
% into account any delay introduced in the channel. This is a mix
% of multipath delay and implementation delay. This value may
% change depending on the sampling rate, delay profile and delay
% spread
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))]; %#ok<AGROW>
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

% Add AWGN to the received time domain waveform
% Normalize noise power by the IFFT size used in OFDM modulation,
% as the OFDM modulator applies this normalization to the
% transmitted waveform. Also normalize by the number of receive
% antennas, as the channel model applies this normalization to the
% received waveform, by default
SNR = 10^(SNRdB/10);
N0 = 1/sqrt(2.0*simLocal.NRxAnts*double(waveinfoLocal.Nfft)*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

if (simLocal.PerfectChannelEstimator)
    % Perfect synchronization. Use information provided by the
    % channel to find the strongest multipath component
```

```
        pathFilters = getPathFilters(channel);
        [offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
    else
        % Practical synchronization. Correlate the received waveform
        % with the PUSCH DM-RS to give timing offset estimate 't' and
        % correlation magnitude 'mag'. The function
        % hSkipWeakTimingOffset is used to update the receiver timing
        % offset. If the correlation peak in 'mag' is weak, the current
        % timing estimate 't' is ignored and the previous estimate
        % 'offset' is used
        [t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols);
        offset = hSkipWeakTimingOffset(offset,t,mag);
        % Display a warning if the estimated timing offset exceeds the
        % maximum channel delay
        if offset > maxChDelay
            warning(['Estimated timing offset (%d) is greater than the maximum channel delay
                ' This will result in a decoding failure. This may be caused by low SNR,' ..
                ' or not enough DM-RS symbols to synchronize successfully.'],offset,maxChDela
        end
    end
    rxWaveform = rxWaveform(1+offset:end,:);

    % Perform OFDM demodulation on the received data to recreate the
    % resource grid, including padding in the event that practical
    % synchronization results in an incomplete slot being demodulated
    rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
    [K,L,R] = size(rxGrid);
    if (L < carrier.SymbolsPerSlot)
        rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
    end

    if (simLocal.PerfectChannelEstimator)
        % Perfect channel estimation, use the value of the path gains
        % provided by the channel
        estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sample

        % Get perfect noise estimate (from the noise realization)
        noiseGrid = nrOFDMDemodulate(carrier,noise(1+offset:end,:));
        noiseEst = var(noiseGrid(:));

        % Apply MIMO deprecoding to estChannelGrid to give an estimate
        % per transmission layer
        K = size(estChannelGrid,1);
        estChannelGrid = reshape(estChannelGrid,K*carrier.SymbolsPerSlot*simLocal.NRxAnts,si
        estChannelGrid = estChannelGrid * F.';
        if (strcmpi(pusch.TransmissionScheme,'codebook'))
            W = nrPUSCHCodebook(pusch.NumLayers,pusch.NumAntennaPorts,pusch.TPMI,pusch.Trans
            estChannelGrid = estChannelGrid * W.';
        end
        estChannelGrid = reshape(estChannelGrid,K,carrier.SymbolsPerSlot,simLocal.NRxAnts,[]
    else
        % Practical channel estimation between the received grid and
        % each transmission layer, using the PUSCH DM-RS for each layer
        % which are created by specifying the non-codebook transmission
        % scheme
        dmrsLayerSymbols = nrPUSCHDMRS(carrier,puschNonCodebook);
        dmrsLayerIndices = nrPUSCHDMRSIndices(carrier,puschNonCodebook);
        [estChannelGrid,noiseEst] = nrChannelEstimate(carrier,rxGrid,dmrsLayerIndices,dmrsLay
```

```
            end

            % Get PUSCH resource elements from the received grid
            [puschRx,puschHest] = nrExtractResources(puschIndices,rxGrid,estChannelGrid);

            % Equalization
            [puschEq,csi] = nrEqualizeMMSE(puschRx,puschHest,noiseEst);

            % Decode PUSCH physical channel
            [ulschLLRs,rxSymbols] = nrPUSCHDecode(carrier,puschNonCodebook,puschEq,noiseEst);

            % Display EVM per layer, per slot and per RB. Reference symbols for
            % each layer are created by specifying the non-codebook
            % transmission scheme
            if (simLocal.DisplayDiagnostics)
                refSymbols = nrPUSCH(carrier,puschNonCodebook,codedTrBlock);
                plotLayerEVM(NSlots,nslot,puschNonCodebook,size(puschGrid),puschIndices,refSymbols,pu
            end

            % Apply channel state information (CSI) produced by the equalizer,
            % including the effect of transform precoding if enabled
            if (pusch.TransformPrecoding)
                MSC = MRB * 12;
                csi = nrTransformDeprecode(csi,MRB) / sqrt(MSC);
                csi = repmat(csi((1:MSC:end).'),1,MSC).';
                csi = reshape(csi,size(rxSymbols));
            end
            csi = nrLayerDemap(csi);
            Qm = length(ulschLLRs) / length(rxSymbols);
            csi = reshape(repmat(csi{1}.',Qm,1),[],1);
            ulschLLRs = ulschLLRs .* csi;

            % Decode the UL-SCH transport channel
            decodeULSCHLocal.TransportBlockLength = trBlkSize;
            [decbits,blkerr] = decodeULSCHLocal(ulschLLRs,pusch.Modulation,pusch.NumLayers,harqEntity

            % Store values to calculate throughput
            simThroughput(snrIdx) = simThroughput(snrIdx) + (~blkerr * trBlkSize);
            maxThroughput(snrIdx) = maxThroughput(snrIdx) + trBlkSize;

            % Update current process with CRC error and advance to next process
            procstatus = updateAndAdvance(harqEntity,blkerr,trBlkSize,puschIndicesInfo.G);
            if (simLocal.DisplaySimulationInformation)
                fprintf('\n(%3.2f%%) NSlot=%d, %s',100*(nslot+1)/NSlots,nslot,procstatus);
            end

        end

        % Display the results dynamically in the command window
        if (simLocal.DisplaySimulationInformation)
            fprintf('\n');
        end
        fprintf('\nThroughput(Mbps) for %d frame(s) = %.4f\n',simLocal.NFrames,1e-6*simThroughput(sn
        fprintf('Throughput(%%) for %d frame(s) = %.4f\n',simLocal.NFrames,simThroughput(snrIdx)*100/

    end
```

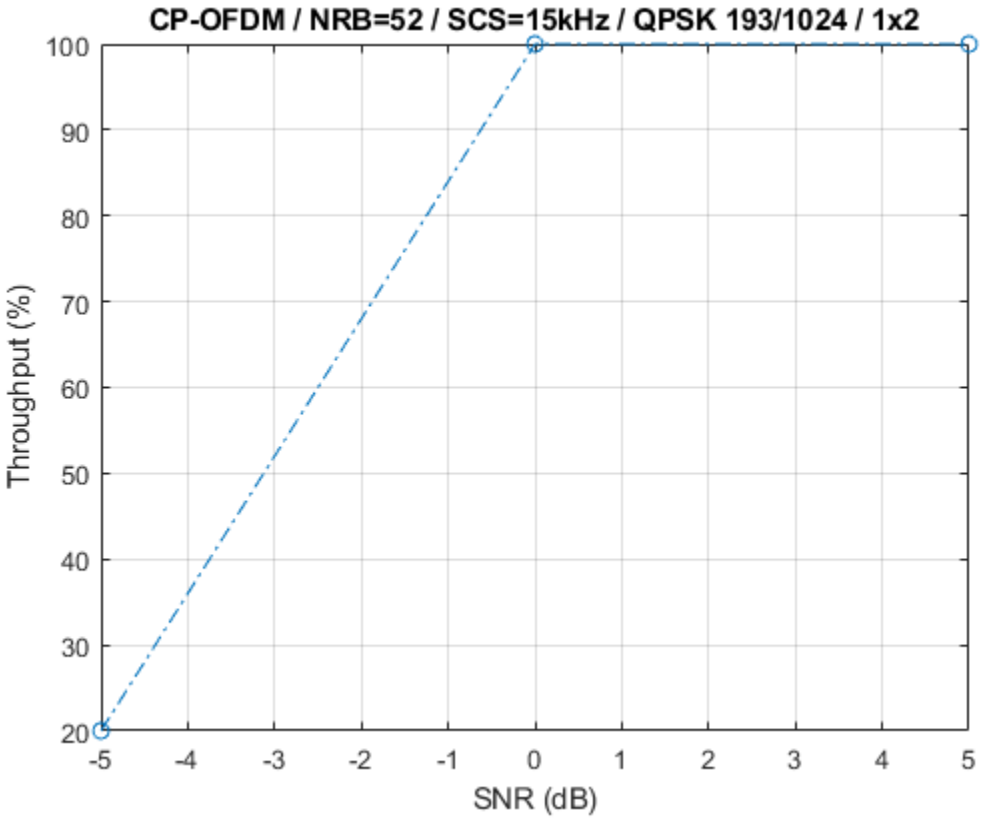
```
Simulating transmission scheme 1 (1x2) and SCS=15kHz with TDL-A channel at -5dB SNR for 2 10ms fr

(5.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.190705).
(10.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.190705).
(15.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.190705).
(20.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.190705).
(25.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.190705).
(30.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.190705).
(35.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.190705).
(40.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.190705).
(45.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.190705).
(50.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.190705).
(55.00%) NSlot=10, HARQ Proc 10: CW0: Initial transmission failed (RV=0,CR=0.190705).
(60.00%) NSlot=11, HARQ Proc 11: CW0: Initial transmission failed (RV=0,CR=0.190705).
(65.00%) NSlot=12, HARQ Proc 12: CW0: Initial transmission failed (RV=0,CR=0.190705).
(70.00%) NSlot=13, HARQ Proc 13: CW0: Initial transmission failed (RV=0,CR=0.190705).
(75.00%) NSlot=14, HARQ Proc 14: CW0: Initial transmission failed (RV=0,CR=0.190705).
(80.00%) NSlot=15, HARQ Proc 15: CW0: Initial transmission failed (RV=0,CR=0.190705).
(85.00%) NSlot=16, HARQ Proc 0: CW0: Retransmission #1 passed (RV=2,CR=0.190705).
(90.00%) NSlot=17, HARQ Proc 1: CW0: Retransmission #1 passed (RV=2,CR=0.190705).
(95.00%) NSlot=18, HARQ Proc 2: CW0: Retransmission #1 passed (RV=2,CR=0.190705).
(100.00%) NSlot=19, HARQ Proc 3: CW0: Retransmission #1 passed (RV=2,CR=0.190705).

Throughput(Mbps) for 2 frame(s) = 0.5712
Throughput(%) for 2 frame(s) = 20.0000

Simulating transmission scheme 1 (1x2) and SCS=15kHz with TDL-A channel at 0dB SNR for 2 10ms fra

(5.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.190705).
(10.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.190705).
(15.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.190705).
(20.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.190705).
(25.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.190705).
(30.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.190705).
(35.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.190705).
(40.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.190705).
(45.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.190705).
(50.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.190705).
(55.00%) NSlot=10, HARQ Proc 10: CW0: Initial transmission passed (RV=0,CR=0.190705).
(60.00%) NSlot=11, HARQ Proc 11: CW0: Initial transmission passed (RV=0,CR=0.190705).
(65.00%) NSlot=12, HARQ Proc 12: CW0: Initial transmission passed (RV=0,CR=0.190705).
(70.00%) NSlot=13, HARQ Proc 13: CW0: Initial transmission passed (RV=0,CR=0.190705).
(75.00%) NSlot=14, HARQ Proc 14: CW0: Initial transmission passed (RV=0,CR=0.190705).
(80.00%) NSlot=15, HARQ Proc 15: CW0: Initial transmission passed (RV=0,CR=0.190705).
(85.00%) NSlot=16, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.190705).
(90.00%) NSlot=17, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.190705).
(95.00%) NSlot=18, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.190705).
(100.00%) NSlot=19, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.190705).

Throughput(Mbps) for 2 frame(s) = 2.8560
Throughput(%) for 2 frame(s) = 100.0000

Simulating transmission scheme 1 (1x2) and SCS=15kHz with TDL-A channel at 5dB SNR for 2 10ms fra

(5.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.190705).
(10.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.190705).
(15.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.190705).
```

```
(20.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.190705).
(25.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.190705).
(30.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.190705).
(35.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.190705).
(40.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.190705).
(45.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.190705).
(50.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.190705).
(55.00%) NSlot=10, HARQ Proc 10: CW0: Initial transmission passed (RV=0,CR=0.190705).
(60.00%) NSlot=11, HARQ Proc 11: CW0: Initial transmission passed (RV=0,CR=0.190705).
(65.00%) NSlot=12, HARQ Proc 12: CW0: Initial transmission passed (RV=0,CR=0.190705).
(70.00%) NSlot=13, HARQ Proc 13: CW0: Initial transmission passed (RV=0,CR=0.190705).
(75.00%) NSlot=14, HARQ Proc 14: CW0: Initial transmission passed (RV=0,CR=0.190705).
(80.00%) NSlot=15, HARQ Proc 15: CW0: Initial transmission passed (RV=0,CR=0.190705).
(85.00%) NSlot=16, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.190705).
(90.00%) NSlot=17, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.190705).
(95.00%) NSlot=18, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.190705).
(100.00%) NSlot=19, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.190705).

Throughput(Mbps) for 2 frame(s) = 2.8560
Throughput(%) for 2 frame(s) = 100.0000
```

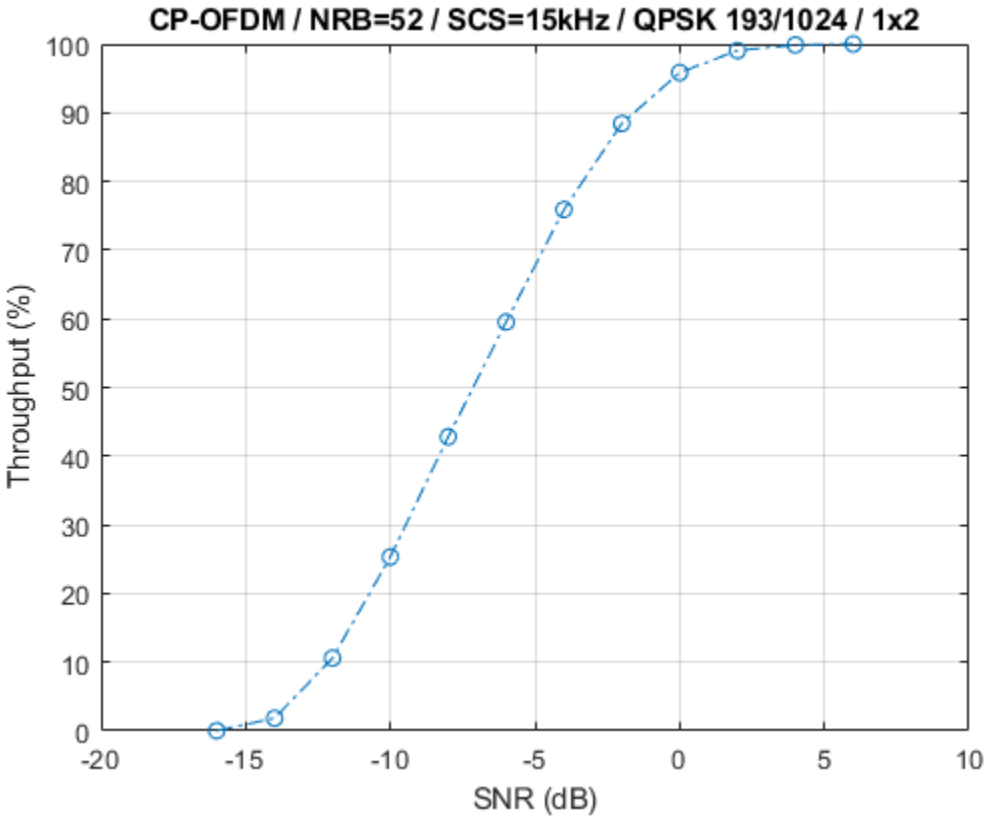
Results

Display the measured throughput. This is calculated as the percentage of the maximum possible throughput of the link given the available resources for data transmission.

```
figure;
plot(simParameters.SNRIn,simThroughput*100./maxThroughput,'o-.')
xlabel('SNR (dB)'); ylabel('Throughput (%)'); grid on;
if (simParameters.PUSCH.TransformPrecoding)
    ofdmType = 'DFT-s-OFDM';
else
    ofdmType = 'CP-OFDM';
end
title(sprintf('%s / NRB=%d / SCS=%dkHz / %s %d/1024 / %dx%d', ...
    ofdmType,simParameters.Carrier.NSizeGrid,simParameters.Carrier.SubcarrierSpacing, ...
    simParameters.PUSCH.Modulation, ...
    round(simParameters.PUSCHExtension.TargetCodeRate*1024),simParameters.NTxAnts,simParameters.N

% Bundle key parameters and results into a combined structure for recording
simResults.simParameters = simParameters;
simResults.simThroughput = simThroughput;
simResults.maxThroughput = maxThroughput;
```



The figure below shows throughput results obtained simulating 10000 subframes (`NFrames = 1000`, `SNRIn = -16:2:6`).



Selected Bibliography

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.
3. 3GPP TS 38.213. "NR; Physical layer procedures for control." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
4. 3GPP TS 38.214. "NR; Physical layer procedures for data." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
5. 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.

Local Functions

```
function validateNumLayers(simParameters)
% Validate the number of layers, relative to the antenna geometry

    numlayers = simParameters.PUSCH.NumLayers;
    ntxants = simParameters.NTxAnts;
    nrxants = simParameters.NRxAnts;
    antennaDescription = sprintf('min(NTxAnts,NRxAnts) = min(%d,%d) = %d',ntxants,nrxants,min(nt
    if numlayers > min(ntxants,nrxants)
        error('The number of layers (%d) must satisfy NumLayers <= %s', ...
            numlayers,antennaDescription);
```

```
    end

    % Display a warning if the maximum possible rank of the channel equals
    % the number of layers
    if (numlayers > 2) && (numlayers == min(ntxants,nrxants))
        warning(['The maximum possible rank of the channel, given by %s, is equal to NumLayers (%
            ' This may result in a decoding failure under some channel conditions.' ...
            ' Try decreasing the number of layers or increasing the channel rank' ...
            ' (use more transmit or receive antennas).'],antennaDescription,numlayers); %#ok<SPWH
    end

end

function plotLayerEVM(NSlots,nslot,pusch,siz,puschIndices,puschSymbols,puschEq)
% Plot EVM information

    persistent slotEVM;
    persistent rbEVM
    persistent evmPerSlot;

    if (nslot==0)
        slotEVM = comm.EVM;
        rbEVM = comm.EVM;
        evmPerSlot = NaN(NSlots,pusch.NumLayers);
        figure;
    end
    evmPerSlot(nslot+1,:) = slotEVM(puschSymbols,puschEq);
    subplot(2,1,1);
    plot(0:(NSlots-1),evmPerSlot,'o-');
    xlabel('Slot number');
    ylabel('EVM (%)');
    legend("layer " + (1:pusch.NumLayers),'Location','EastOutside');
    title('EVM per layer per slot');

    subplot(2,1,2);
    [k,~,p] = ind2sub(siz,puschIndices);
    rbsubs = floor((k-1) / 12);
    NRB = siz(1) / 12;
    evmPerRB = NaN(NRB,pusch.NumLayers);
    for nu = 1:pusch.NumLayers
        for rb = unique(rbsubs).'
            this = (rbsubs==rb & p==nu);
            evmPerRB(rb+1,nu) = rbEVM(puschSymbols(this),puschEq(this));
        end
    end
    plot(0:(NRB-1),evmPerRB,'x-');
    xlabel('Resource block');
    ylabel('EVM (%)');
    legend("layer " + (1:pusch.NumLayers),'Location','EastOutside');
    title(['EVM per layer per resource block, slot #' num2str(nslot)]);

    drawnow;
```

```
end
```

See Also

Objects
`nrULSCH` | `nrULSCHDecoder` | `nrTDLChannel` | `nrCDLChannel`

Functions
`nrPUSCHDecode` | `nrPUSCH` | `nrULSCHInfo` | `parfor`

More About

- “SNR Definition Used in Link Simulations” on page 5-86

NR SRS Configuration

This example shows how to generate sounding reference signals (SRS), as defined in TS 38.211 Section 6.4.1.4 [1 on page 2-73], including SRS configuration, symbol and indices generation, OFDM resource grid mapping, and SRS waveform generation. The example demonstrates how to select the appropriate parameters to position the SRS in frequency and to setup:

- *Full-band transmissions*: Generate an SRS spanning all the available bandwidth.
- *Frequency-hopping transmissions*: Generate periodic and aperiodic SRS transmissions with different frequency-hopping patterns.
- *Multi-user transmissions*: Generate orthogonal SRS using time, frequency, and cyclic shifts.

Introduction

Sounding reference signals are uplink physical signals employed by user equipment (UE) for uplink channel sounding, including channel quality estimation and synchronization. Unlike demodulation reference signals (DM-RS), SRS are not associated to any physical uplink channels and they support uplink channel-dependent scheduling and link adaptation. SRS assist in:

- Codebook-based closed-loop spatial multiplexing.
- Control uplink transmit timing.
- Reciprocity-based downlink precoding in multi-user MIMO setups.
- Quasi co-location of physical channels and reference signals.

Frequency Positioning

This section shows how to configure the SRS position in frequency domain and determine the bandwidth allocated to the SRS.

Configure a 15 MHz bandwidth carrier with 15 kHz subcarrier spacing (SCS).

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 79; % Bandwidth in RB
carrier.SubcarrierSpacing = 15 kHz;
```

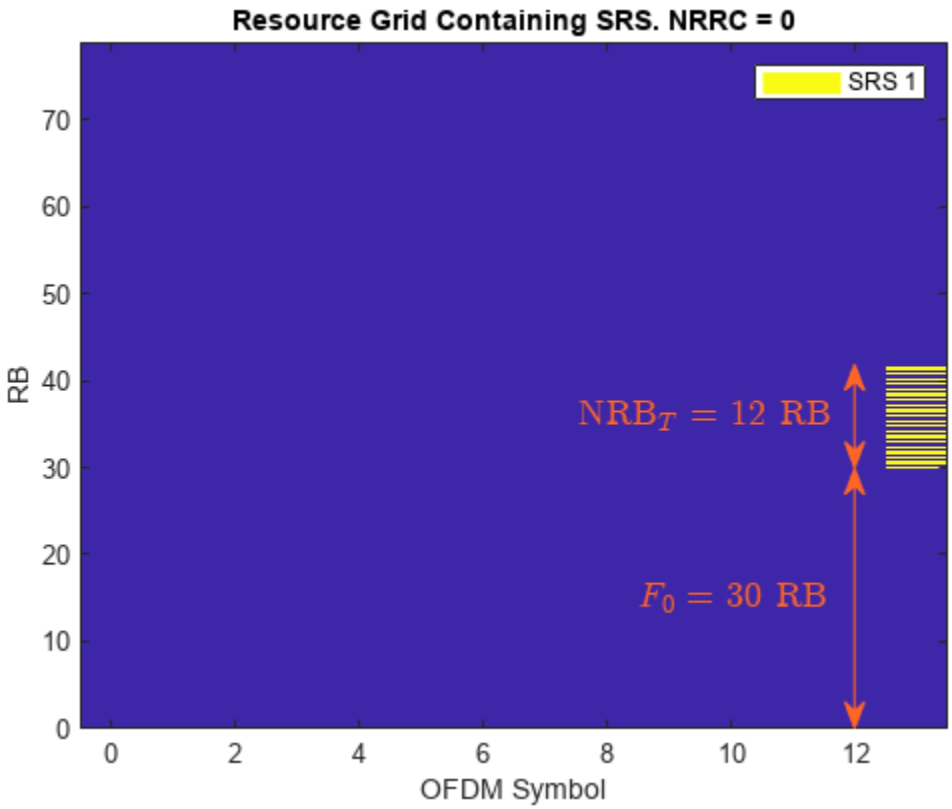
The bandwidth configuration parameters `CSRS` and `BSRS` control the bandwidth allocated to the SRS, which normally increases with `CSRS` and decreases with `BSRS`. Use these interactive controls to configure the SRS bandwidth.

```
srs = nrSRSConfig;
srs.CSRS = 10;  % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 1;  % Bandwidth configuration B_SRS (0...3)
```

To modify the frequency position of the SRS, change the values of `FrequencyStart` and `NRRC`. `FrequencyStart` specifies the frequency origin of the SRS in RBs with respect to the carrier origin when `NRRC = 0`.

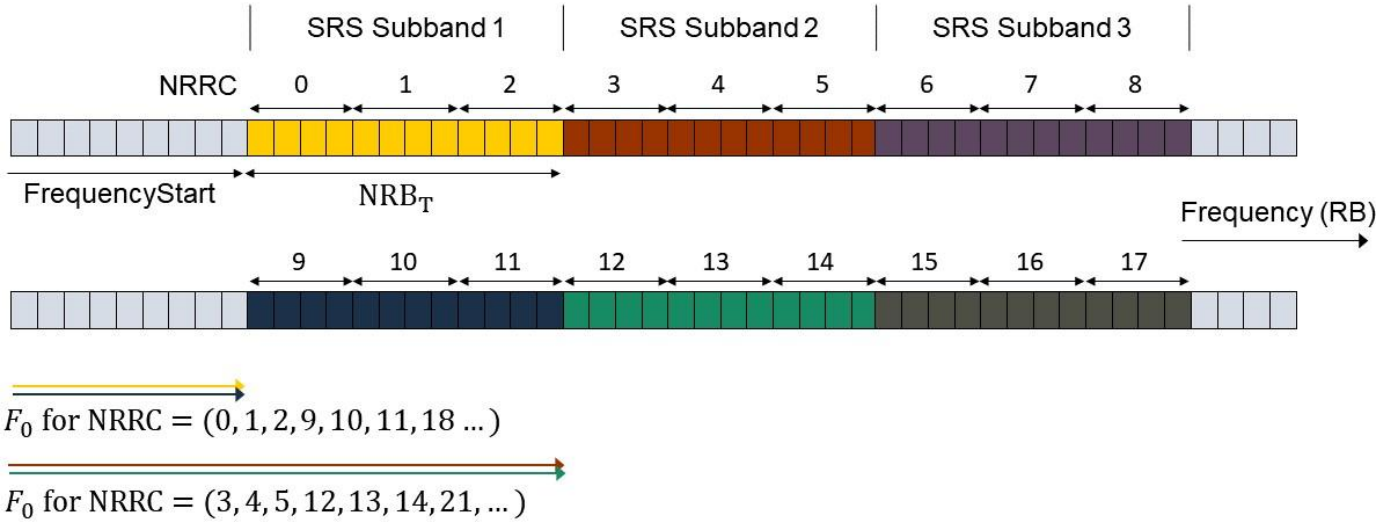
```
srs.FrequencyStart = 30; % Frequency position of the SRS in carrier in RB
          srs.NRRC = 0; % Frequency domain position in blocks of 4 PRB (
```

```
hSRSGrid(carrier,srs,1,true); % Create and display a single-slot resource grid containing SRS
title(['Resource Grid Containing SRS. NRRC = ' num2str(srs.NRRC)]);
hSRSAnnotations(carrier,srs);
```



This figure displays a single-slot OFDM resource grid containing an SRS. For `CSRS = 10` and `BSRS = 1`, the frequency position of the SRS (F_0) shifts by NRB_T when `NRRC` is in the range (3...5) and by $2NRB_T$ when `NRRC` is in the range (6...8). The SRS returns to the initial position (`FrequencyStart`) when `NRRC` is 9. NRB_T is the number of resource blocks (RBs) allocated to the SRS per transmission.

This figure illustrates the concepts introduced above for `CSRS = 10` and `BSRS = 1`.



`NRRC` is an additional frequency offset specified as a number of 4 RBs and it corresponds to the higher layer parameter *freqDomainPosition* (see TS 38.331 Section 6.3.2 SRS-Config [2 on page 2-73]). For values of `NRRC` between $k \cdot (\mathrm{NRB}_T/4)$ and $(k+1) \cdot (\mathrm{NRB}_T/4) - 1$, the frequency position of the SRS is shifted by $k \cdot \mathrm{NRB}_T$, where k is an integer. TS 38.211 Section 6.4.1.4 refers to NRB_T as $m_{\mathrm{SRS},b}$ with b = BSRS. For more information, see the `NRBPerTransmission` property of the `nrSRSConfig` configuration object.

This equation determines the origin of the SRS in frequency:

$$F_0 = \mathrm{FrequencyStart} + \mathrm{NRB}_T \cdot \mathrm{mod}\left(\left\lfloor 4\frac{\mathrm{NRRC}}{\mathrm{NRB}_T}\right\rfloor, \mathrm{NSB}\right)$$

NSB denotes the number of SRS subbands (frequency bands of size NRB_T) where the SRS can be positioned through the parameter `NRRC`. To calculate NSB, you can use the SRS bandwidth configuration table (see TS 38.211 Table 6.4.1.4.3-1). You can also access this table through the `BandwidthConfigurationTable` property of the `nrSRSConfig` object.

```
csrs = srs.CSRS;
disp(nrSRSConfig.BandwidthConfigurationTable(csrs+(0:2) + (csrs==0),:));
```

C_SRS	m_SRS_0	N_0	m_SRS_1	N_1	m_SRS_2	N_2	m_SRS_3	N_3
9	32	1	16	2	8	2	4	2
10	36	1	12	3	4	3	4	1
11	40	1	20	2	4	5	4	1

The first column contains possible values of the parameter `CSRS`. For `CSRS = 10` and `BSRS = 1`, the number of unique SRS subbands is NSB = $N_0 \cdots N_{B_{\mathrm{SRS}}} = 3$, where $N_0 = 1$ and $N_1 = 3$.

```
% Calculate and display the number of SRS subbands configurable by NRRC
NSBTable = hSRSNumberOfSubbandsOrHoppingPatterns(srs);

fprintf('Number of SRS subbands (NRRC < %d):  %d', NSBTable*srs.NRBPerTransmission/4,NSBTable);
```

```
Number of SRS subbands (NRRC < 9):  3
```

```
% Calculate the frequency origin of the first SRS symbol
f0 = hSRSFrequencyOrigin(srs);
fprintf('The frequency origin of the SRS is F0 = %d RB.', f0);
```

```
The frequency origin of the SRS is F0 = 30 RB.
```

Confirm the frequency position of the SRS using the `info` output of nrSRSIndices

```
[~,info] = nrSRSIndices(carrier,srs);
display(info.PRBSet(1))
```

```
    30
```

For a given value of `CSRS`, the frequency band where the SRS can be allocated using `NRRC` is FrequencyStart + (0...NSB · NRB_T − 1). For `CSRS = 10`, the SRS wraps around when NRRC = NSB · $\mathrm{NRB}_T/4 = N_0 \cdots N_{B_{\mathrm{SRS}}} \cdot m_{\mathrm{SRS},b}/4 = m_{\mathrm{SRS},0}/4 = 9$, which results into the same frequency position as that of NRRC = 0.

```
fprintf('The SRS frequency range is limited to the range (%d,%d) RB for the current value of CSRS
```

```
The SRS frequency range is limited to the range (30,66) RB for the current value of CSRS (10).
```

When intra-slot frequency hopping is enabled, the calculation of `FrequencyOrigin` is only valid for the first SRS symbol in the slot. For frequency-hopping SRS, use the outputs `info.PRBset` or `info.SubcarrierOffset` of the `nrSRSIndices` function to determine the frequency position of subsequent symbols.

```
fprintf('The frequency origin of the SRS is F0 = %d RB.',info.PRBSet(1));
```

```
The frequency origin of the SRS is F0 = 30 RB.
```

Full-Bandwidth Configuration

This section shows how to configure and generate a full-band SRS transmission by calculating the appropriate SRS bandwidth parameters for a carrier.

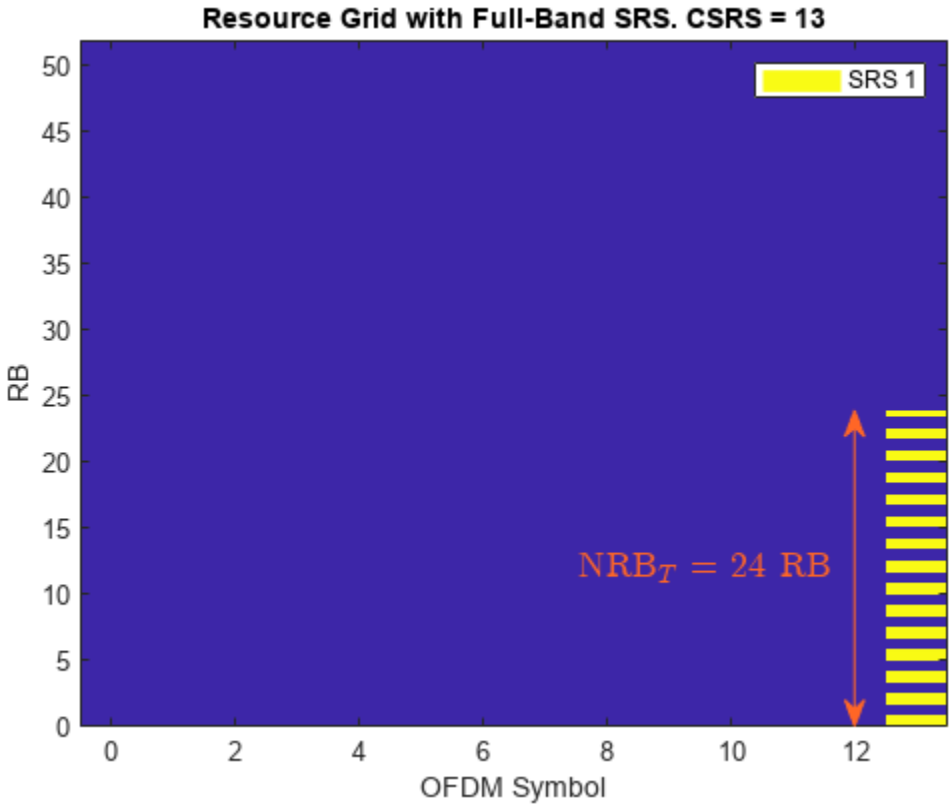
Configure a 10 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15 kHz;
```

The bandwidth configuration parameters `CSRS`, `BSRS`, and `BHop` control the bandwidth allocated to the SRS. To configure a full-band SRS, set `CSRS = 14` and `BSRS = 0`.

```
srs = nrSRSConfig;
srs.CSRS = 13; % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 1; % Bandwidth configuration B_SRS (0...3)
```

```
hSRSGrid(carrier,srs, 1, true); % Create and display a single-slot resource grid containing SRS
title(['Resource Grid with Full-Band SRS. CSRS = ' num2str(srs.CSRS)]);
hSRSAnnotations(carrier,srs);
```



For an SRS frequency allocation, you can find the appropriate values of `CSRS`, `BSRS`, and `BHop` in the SRS bandwidth configuration table (see TS 38.211 Table 6.4.1.4.3-1). Alternatively, `nrSRSConfig.BandwidthConfigurationTable` provides an easy way to access and display this table.

```
% To display relevant rows of the bandwidth configuration table, calculate the value of CSRS for
[csrs,bsrs] = hSRSBandwidthConfiguration(srs,carrier.NSizeGrid);

% Display bandwidth configuration table
disp(nrSRSConfig.BandwidthConfigurationTable(csrs+(0:2) + 1*(csrs==0),:));
```

C_SRS	m_SRS_0	N_0	m_SRS_1	N_1	m_SRS_2	N_2	m_SRS_3	N_3
13	48	1	24	2	12	2	4	3
14	52	1	4	13	4	1	4	1
15	56	1	28	2	4	7	4	1

The columns labeled as `m_SRS_b` with `b = 0...3` contain the number of RBs allocated to the SRS for the parameter `b = BSRS` and non-hopping configurations. The row corresponding to `C_SRS = 14` contains a number of RBs `m_SRS_0 = 52`, which is the closest value to the carrier bandwidth. Therefore, the parameters `CSRS = 14` and `BSRS = 0` configure a full-band SRS transmission for the current carrier configuration. This example uses the bandwidth configuration table to calculate the values of `CSRS` and `BSRS` that maximize the SRS bandwidth within the carrier.

```
fprintf('For a full-band SRS in a carrier bandwidth of %d RB, set CSRS = %d and BSRS = %d.',carri
```

```
For a full-band SRS in a carrier bandwidth of 52 RB, set CSRS = 14 and BSRS = 0.
```

You can use the SRS read-only property `NRBPerTransmission` to confirm that the generated SRS fits into the carrier bandwidth.

```
fprintf('The SRS bandwidth (%d RB) is lower than or equal to the carrier bandwidth (%d RB).',srs
```

```
The SRS bandwidth (24 RB) is lower than or equal to the carrier bandwidth (52 RB).
```

Frequency-Hopping Configuration

You can configure intra-slot and inter-slot frequency hopping for multi-symbol and multi-slot SRS transmissions, respectively. The instantaneous bandwidth per OFDM symbol is constant across SRS OFDM symbols and is smaller than the bandwidth over which the SRS hops.

Configure a 15 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 79;
carrier.SubcarrierSpacing = 15 kHz;
```

Create a four-symbol SRS located at the end of the slot. Select the repetition factor to indicate the number of equal consecutive SRS transmissions (OFDM symbols) in a slot. For frequency-hopping configurations, `Repetition` must be lower than the number of SRS symbols.

```
srs = nrSRSConfig;
srs.NumSRSSymbols = 4;
   srs.Repetition = 1;
  srs.SymbolStart = 10 ; % Time-domain position of the SRS in the slot. (8
```

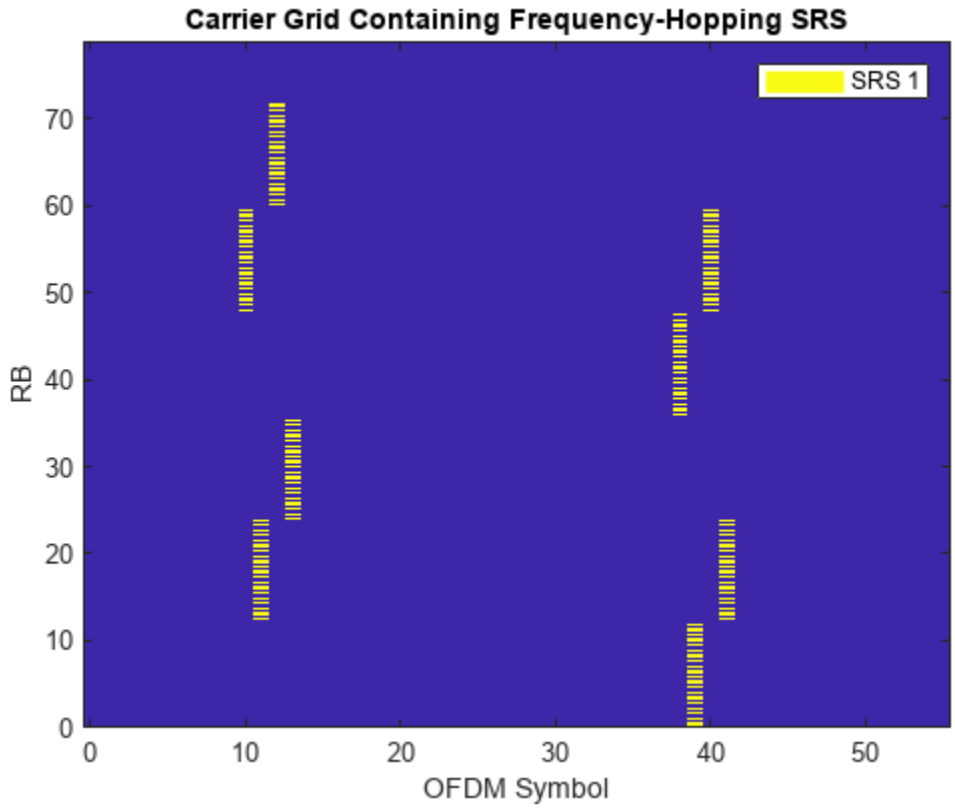
Downlink control information (DCI) can trigger aperiodic SRS transmissions using the higher layer parameter *resourceType* (see TS 38.331 Section 6.3.2 SRS-Config). As the frequency hopping pattern is reset for aperiodic SRS resource types at each slot, select `periodic` or `semi-persistent` SRS resource types to enable inter-slot frequency hopping or `aperiodic` to disable it. You can configure the slot-wise periodicity and offset of the SRS transmissions by using the property `SRSPeriod`. For `aperiodic` resource types, the parameter `SRSPeriod` controls the periodicity and offset of DCI signals triggering aperiodic SRS transmissions.

```
srs.ResourceType = periodic;
   srs.SRSPeriod = [2 0]; % Periodicity in slots (1,2,4,5,8,10,...)
srs.SRSPeriod(2) = 0 ; % Offset in slots must be smaller than the periodic
```

Use these interactive controls to select a hopping configuration and observe the changes to the OFDM resource grid.

```
srs.CSRS = 19 ; % Bandwidth configuration C_SRS (0...63)
srs.BSRS = 2 ;  % Bandwidth configuration B_SRS (0...3)
srs.BHop = 0 ;  % Frequency hopping configuration (0...3). Set BHop >= BSR
srs.NRRC = 14 ; % Frequency domain position in blocks of 4 PRB (0...67)
```

```
% Create and display a multi-slot resource grid containing SRS
duration = 2*srs.SRSPeriod(1); % Transmission length in slots
hSRSGrid(carrier,srs, duration, true);
title('Carrier Grid Containing Frequency-Hopping SRS')
```



The bandwidth over which the SRS symbols hop increases with the value of `CSRS` and decreases with `BHop` (until `BHop = BSRS` disables hopping). Increasing `BSRS` reduces the allocated bandwidth per OFDM symbol (property `NRBPerTransmission`) and can reduce the intra-slot frequency hopping as well. To disable frequency hopping, set `BHop >= BSRS`. For non-hopping configurations, the roles of `CSRS` and `BSRS` are analogous, as the allocated bandwidth increases with `CSRS` and decreases with `BSRS`.

You can use the bandwidth configuration table (TS 38.211 Table 6.4.1.4.3-1) to calculate the number of different frequency-hopping patterns configurable by the SRS parameter `NRRC` as NFHP = $N_{\mathrm{BHop}+1} \cdots N_{B_{\mathrm{SRS}}}$. The frequency-hopping patterns repeat when NRRC > NFHP · $\mathrm{NRB}_T/4$.

```
N = hSRSNumberOfSubbandsOrHoppingPatterns(srs);

if srs.BHop < srs.BSRS && srs(1).NRB > 16 % Number of unique frequency-hopping patterns
    fprintf('Number of unique frequency-hopping patterns (configurable by NRRC < %d): %d.',N*srs
else % Number of unique SRS subbands
    fprintf('Number of unique SRS subbands (configurable by NRRC < %d):  %d.',N*srs.NRBPerTransmi
end
```

```
Number of unique frequency-hopping patterns (configurable by NRRC < 18): 6.
```

Set the bandwidth configurations parameters as `CSRS = 20, BSRS = 2`, and `BHop = 1`. The condition `BHop < BSRS` does not guarantee frequency hopping. However, the opposite is true: `BHop >= BSRS` always disables frequency hopping. You can determine when an SRS bandwidth configuration (`CSRS`,`BSRS`,`BHop`) produces frequency hopping using the N_b parameter from the bandwidth configuration table and evaluating the condition NFHP = $N_{\mathrm{BHop}+1} \cdots N_{B_{\mathrm{SRS}}} > 1$.

```
isFreqHoppingConfiguration = hSRSNumberOfSubbandsOrHoppingPatterns(srs) > 1;
disp(['Frequency hopping is' repmat(' not',1,~isFreqHoppingConfiguration) ' enabled.'])
```

```
Frequency hopping is enabled.
```

In frequency-hopping cases, the read-only property `NRB` specifies the bandwidth over which the SRS hops and `NRBPerTransmission` specifies the instantaneous bandwidth allocated per SRS OFDM symbol.

```
t = table([srs.NRB; srs.NRBPerTransmission],'RowNames', {'Freq-Hopping bandwidth','Instantaneous
disp(t)
```

```
                            NRB
                            ___

    Freq-Hopping bandwidth  72
    Instantaneous bandwidth 12
```

Multi-User Configurations

This section explains how to configure multiple SRS transmissions suitable for multi-user setups. You can use time-domain, frequency-domain, and sequence-domain parameters to create sets of orthogonal (interference-free) SRS transmissions.

Time-Domain Orthogonal SRS

You can create time-domain orthogonal SRS transmissions in multiple ways, for example:

- Configure different slot periodicity and offset for different SRS by using the property `SRSPeriod`.
- Configure different symbol-wise time domain allocations for different SRS by using the property `SymbolStart`.

Configure a 10 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15 kHz;
```

Specify the slot periodicity and offset to create orthogonal SRS transmissions.

First SRS configuration:

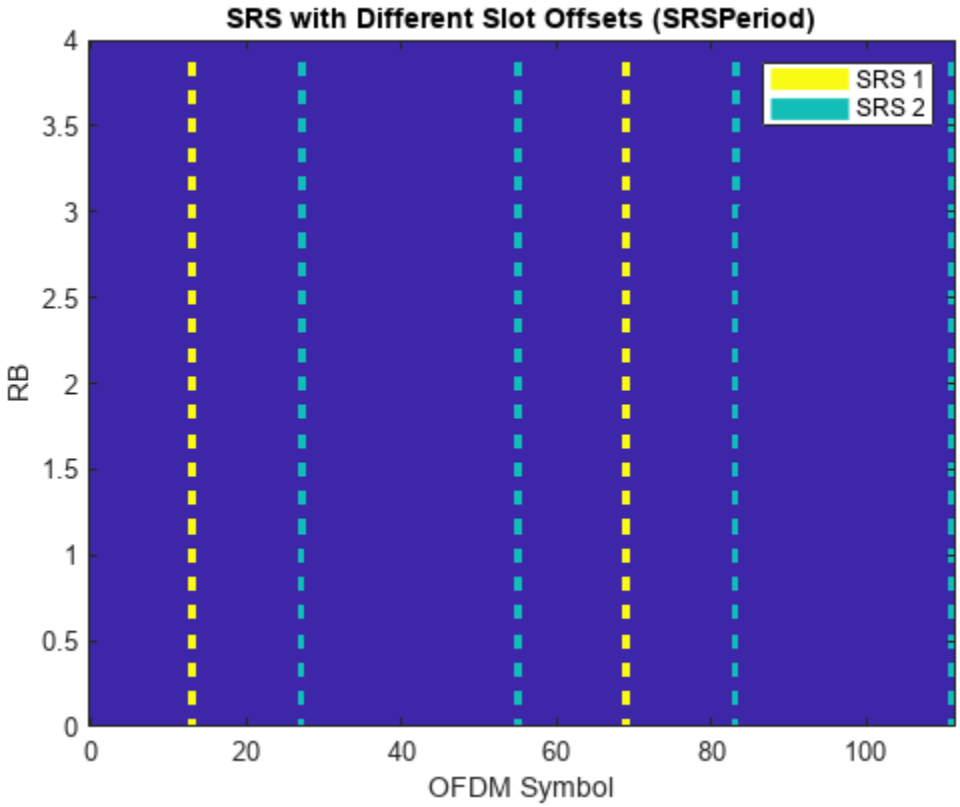
```
srs = nrSRSConfig;
srs(1).SRSPeriod = [1 0];
srs(1).SRSPeriod(1) = 4; % Slot periodicity and offset
srs(1).SRSPeriod(2) = 0 ; % Slot offset of first SRS
```

Second SRS configuration:

```
srs(2) = srs(1); % Create a copy of the configured SRS
srs(2).SRSPeriod(1) = 2; % Periodicity in slots
srs(2).SRSPeriod(2) = 1; % Offset in slots
```

This figure displays an OFDM resource grid containing the configured SRS transmissions.

```
hSRSGrid(carrier,srs,srs(1).SRSPeriod(1)*2, true); % Generate a multi-slot resource grid contain
title('SRS with Different Slot Offsets (SRSPeriod)')
ylim([0 srs(1).NRBPerTransmission]);
```



Specify the number of SRS symbols and location in the slot to create orthogonal SRS transmissions.

First SRS configuration:

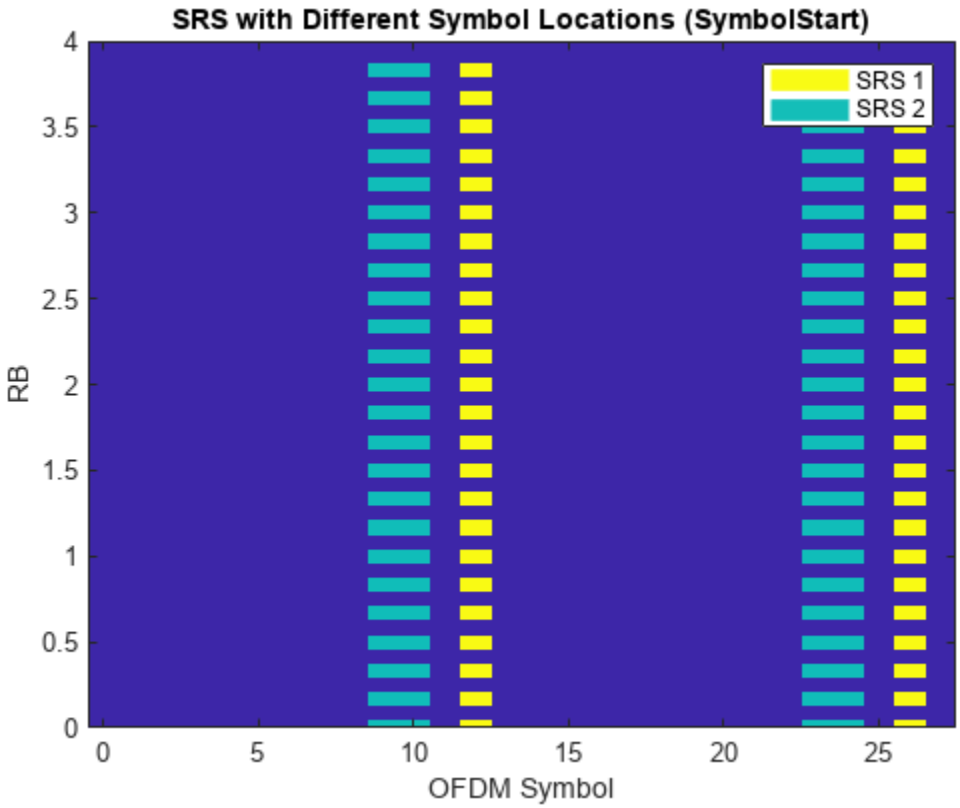
```
srs = nrSRSConfig;
srs(1).NumSRSSymbols = 1;  % Number of SRS symbols in a slot (1,2,4)
srs(1).SymbolStart = 12; % Time-domain position of the SRS in the slot. (
```

Second SRS configuration:

```
srs(2) = srs(1); % Create a copy of the configured SRS
srs(2).NumSRSSymbols = 2;
srs(2).SymbolStart = 9;
```

This figure displays the OFDM resource grid containing the SRS transmissions.

```
hSRSGrid(carrier,srs, 2, true); % Generate and display a 2-slot resource grid containing SRS
title('SRS with Different Symbol Locations (SymbolStart)');
ylim([0 srs(1).NRBPerTransmission]);
```



Frequency-Domain Orthogonal SRS

You can create frequency-domain orthogonal SRS transmissions in multiple ways:

- Configure different comb offsets for different SRS by using the property `KBarTC`.
- Configure different frequency-hopping patterns using `CSRS`, `BSRS`, `BHop` and `NRRC`.

Configure a 10 MHz bandwidth carrier with 15 kHz SCS.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15 kHz;
```

Specify the comb numbers and offsets to create orthogonal SRS transmissions.

First SRS configuration:

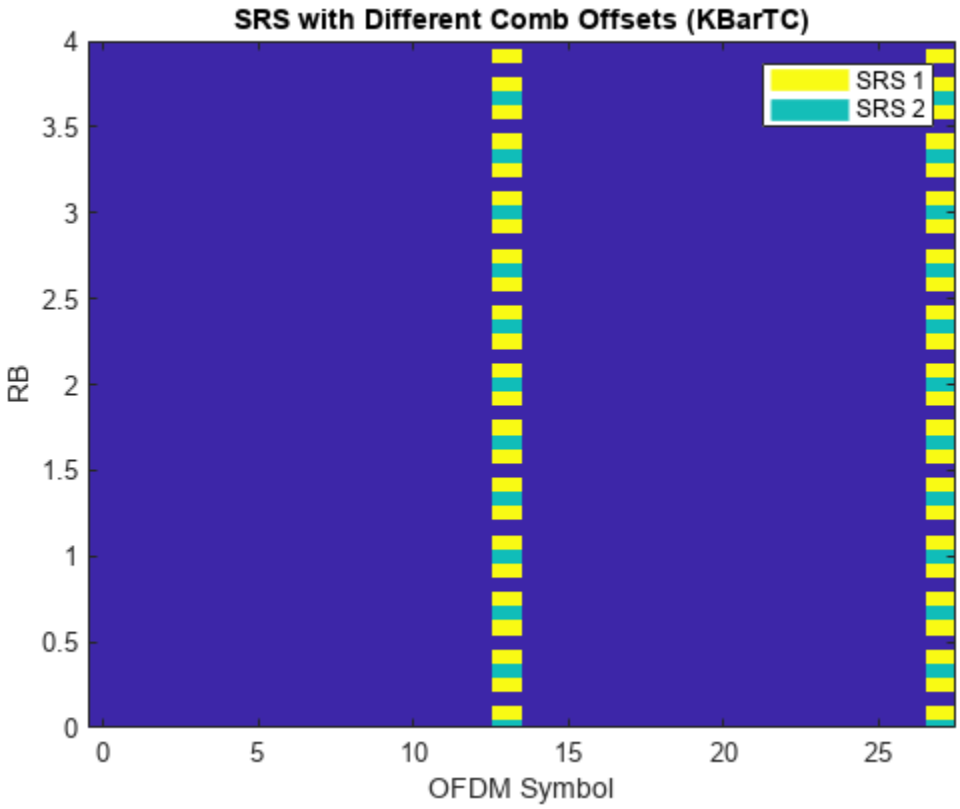
```
srs = nrSRSConfig;
srs(1).KTC    = 2; % Comb number (2,4). It indicates the allocation of the S
srs(1).KBarTC = 1; % Comb offset (0...KTC-1)
```

Second SRS configuration:

```
srs(2) = srs(1); % Create a copy of the configured SRS
srs(2).KTC    = 4;
srs(2).KBarTC = 0;
```

This figure displays the OFDM resource grid containing the SRS transmissions.

```
hSRSGrid(carrier,srs,2,true);
title('SRS with Different Comb Offsets (KBarTC)')
ylim([0 srs(1).NRBPerTransmission]);
```



Create frequency-domain orthogonal SRS configurations using different frequency-hopping patterns.

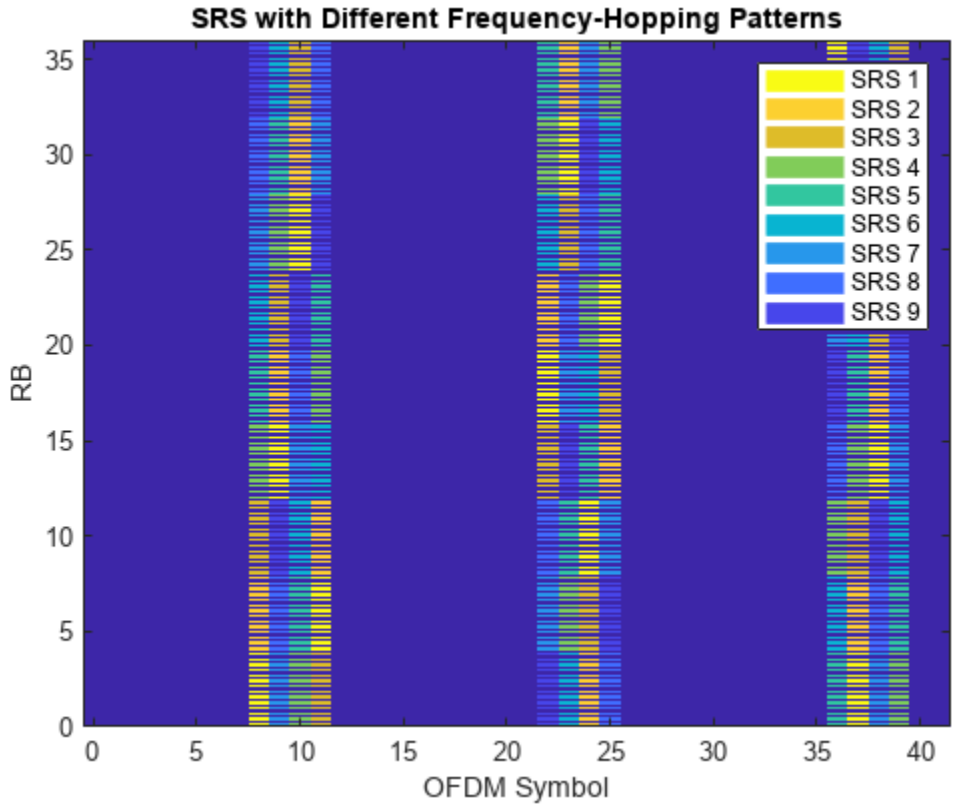
```
srs = nrSRSConfig;
srs.NumSRSSymbols = 4;  % Number of SRS symbols in a slot
srs.SymbolStart = 8; % Allocate SRS in OFDM symbols 10:13
```

Select the desired bandwidth configuration parameters, resource type and repetition factor.

```
srs.CSRS =  10 ;  % Bandwidth configuration C_SRS (0...63)
srs.BSRS =  2 ;  % Bandwidth configuration B_SRS (0...3)
srs.BHop =  0 ;  % Frequency hopping configuration (0...3). Set BHop >= BS
srs.ResourceType = periodic ; % ('periodic','semi-persistent','aperiodic')
srs.Repetition   = 1 ; % Number of equal consecutive SRS transmissions (1,2,4
```

This example calculates the number of orthogonal SRS sequences that can be configured by NRRC and creates frequency non-overlapping SRS configurations. The figure displays the OFDM resource grid containing the SRS transmissions.

```
NRRC = num2cell(hNRRCSet(srs));
srs(2:length(NRRC)) = srs(1); % Create N-1 copies of the configured SRS
[srs(:).NRRC] = deal(NRRC{:}); % Assign the appropriate NRRC to each SRS configuration

hSRSGrid(carrier,srs, 3, true); % Generate and display a 3-slot resource grid containing SRS
title('SRS with Different Frequency-Hopping Patterns');
ylim([0 srs(1).NRBPerTransmission*hSRSNumberOfSubbandsOrHoppingPatterns(srs(1))]);
```




```
N = hSRSNumberOfSubbandsOrHoppingPatterns(srs(1));
if srs(1).BHop < srs(1).BSRS % Frequency-hopping cases
    fprintf('Number of unique frequency-hopping patterns (configurable by NRRC < %d): %d.', N*srs
else
```

```
    fprintf('Number of unique subbands (configurable by NRRC < %d):  %d.', N*srs(1).NRBPerTransm
end

Number of unique frequency-hopping patterns (configurable by NRRC < 9): 9.
```

Note that the SRS transmissions are never overlapping regardless of the `Repetition` factor and `ResourceType` (`aperiodic` disables inter-slot frequency hopping). Disable frequency hopping by setting `BHop >= BSRS` and observe that the different SRS are still allocated to different subbands.

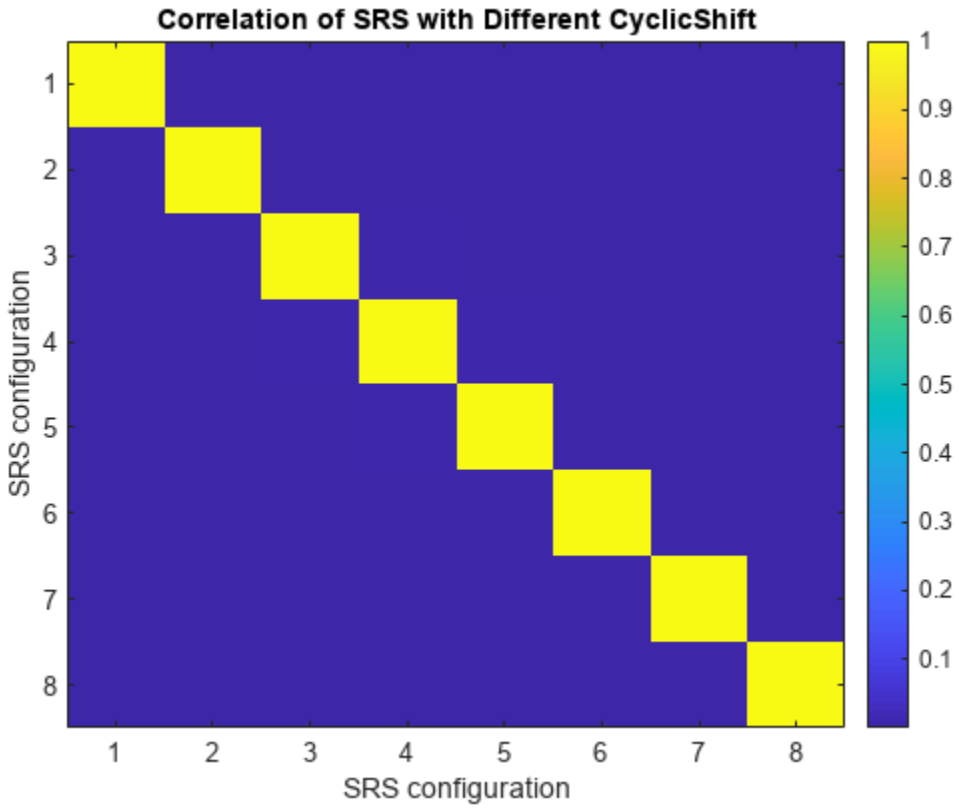
Cyclic-Shift Orthogonal SRS

This section generates multiple SRS allocated to the same time and frequency resources (OFDM symbols and subcarriers) but different time-domain cyclic shifts. Due to the properties of Zadoff-Chu sequences, this configuration produces orthogonal SRS. To demonstrate the orthogonality among the configured SRS, this section performs CP-OFDM modulation and calculates the cross-correlation of the time-domain waveforms.

```
% Configure a 10 MHz bandwidth carrier with 15 kHz SCS.
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;
carrier.SubcarrierSpacing = 15;

% Create a full-band SRS
srs = nrSRSConfig;
srs.CSRS = hSRSBandwidthConfiguration(srs,carrier.NSizeGrid);

% All SRS share the same physical resources, but they are configured with
% different cyclic shifts.
for i = 1:8
    srs(i) = srs(1);
    srs(i).CyclicShift = i-1;
end

% Create a resource grid containing SRS
numSlots = 1; % Number of slots to generate
for ich = length(srs):-1:1
    slotGrid{ich} = hSRSGrid(carrier,srs(ich),numSlots);
end

% Get OFDM modulation related information
ofdmInfo = nrOFDMInfo(carrier);

% OFDM modulation
nsrs = length(srs); % Number of SRS waveforms
numSamples = numSlots*ofdmInfo.SampleRate/1000/carrier.SlotsPerSubframe;
txWaveform = zeros(numSamples,srs(1).NumSRSPorts,nsrs);
for i = 1:nsrs
    txWaveform(:,:,i) = nrOFDMModulate(carrier,slotGrid{i});
end

% Cross correlation of SRS waveforms generated with different cyclic shifts
txWaveform = reshape(txWaveform,[],nsrs*srs(1).NumSRSPorts);
C = txWaveform'*txWaveform;
srsCorr = C./diag(C);
```

This figure shows the time-domain cross-correlation of the SRS waveforms with different cyclic shifts.

```
imagesc(abs(srsCorr))
title('Correlation of SRS with Different CyclicShift')
xlabel('SRS configuration'); ylabel('SRS configuration');
colorbar
```



The low correlations among SRS waveforms generated using different time-domain cyclic shifts show their orthogonality.

```
disp('Absolute value of correlation matrix: '); disp(' '); disp(abs(srsCorr));
```

```
Absolute value of correlation matrix:

    1.0000    0.0027    0.0021    0.0008    0.0003    0.0010    0.0007    0.0038
    0.0027    1.0000    0.0030    0.0006    0.0006    0.0001    0.0006    0.0005
    0.0021    0.0030    1.0000    0.0041    0.0007    0.0008    0.0003    0.0008
    0.0008    0.0006    0.0041    1.0000    0.0040    0.0004    0.0009    0.0000
    0.0003    0.0006    0.0007    0.0040    1.0000    0.0039    0.0002    0.0009
    0.0010    0.0001    0.0008    0.0004    0.0039    1.0000    0.0039    0.0003
    0.0007    0.0006    0.0003    0.0009    0.0002    0.0039    1.0000    0.0039
    0.0038    0.0005    0.0008    0.0000    0.0009    0.0003    0.0039    1.0000
```

Summary and Further Exploration

This example describes how to generate and map SRS sequences into an OFDM carrier resource grid, and how to generate the corresponding waveform for multiple carrier and SRS configurations. The example highlights the relationship between SRS configuration parameters and their effects on both the resource grid and SRS waveform properties. For example:

- The bandwidth allocated to the SRS generally increases with `CSRS` and decreases with `BSRS`. For frequency-hopping configurations, the instantaneous bandwidth and hopping bandwidth decrease with `BSRS` and `BHop`, respectively.
- The frequency position of the SRS depends on the `FrequencyStart` and `NRRC` parameters. `NRRC` allows to select different SRS subbands and frequency-hopping patterns.
- `BHop < BSRS` generally produces frequency hopping, but it is not guaranteed. However, `BHop >= BSRS` is sufficient to disable frequency hopping.
- Inter-slot frequency hopping is active only for `periodic` and `semi-persistent` resource types.
- Time-, frequency-, and sequence-related parameters can be used to create orthogonal SRS transmissions.

The example also demonstrates how to extract useful information from both the SRS bandwidth configuration table and read-only properties, such as, the number of unique SRS subbands and frequency-hopping patterns configurable by `NRRC`.

The SRS sequence generation for multiport configurations employs the orthogonalization mechanism presented in Cyclic-Shift Orthogonal SRS on page 2-71. To verify the orthogonality across multiple ports, configure a multi-port SRS, generate the symbols and indices, map the symbols into a resource grid, generate the SRS waveforms, and compute the cross-correlation between waveforms generated for different ports.

References

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

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

Local functions

This example uses these local functions:

```
function [csrs,bsrs] = hSRSBandwidthConfiguration(srs, NRB)
% [CSRS,BSRS] = hBandwidthConfiguration(SRS, NRB) returns the SRS
% bandwidth configuration parameters CSRS and BSRS required to transmit an
% SRS in a bandwidth specified by NRB. The function calculates CSRS and
% BSRS considering the SRS properties FrequencyStart and NRRC, so the
% available bandwidth NRB is reduced by the frequency origin of the SRS.
% For frequency hopping cases, the value of BSRS returned is equal to that
% of the BSRS property in the input SRS configuration object.

    f0 = hSRSFrequencyOrigin(srs);
    NRB = NRB - f0;
    if NRB < 4
        error('The available bandwidth is not sufficient to allocate an SRS transmission. Increas
    end

    % For frequency hopping configurations
    if srs.BHop >= srs.BSRS && nargout == 2 % No frequency hopping and BSRS is requested
        BSRS = 0:3;
    else
        BSRS = 0;
    end
```

```
    % Initialize gap between SRS frequency allocation and carrier bandwidth
    NRBGap = NRB;

    % Find the appropriate CSRS for each BSRS that minimizes the gap only
    % in non-hopping cases. For freq. hopping, find the value of CSRS only.
    for b = BSRS
        % NRB allocated to the SRS for BSRS = b and all CSRS
        srsNRBb = srs.BandwidthConfigurationTable{:,2*(b+1)};
        mSRSbMax = max( srsNRBb( srsNRBb <= NRB ));

        % Calculate gap between SRS allocation and carrier bandwidth
        gap = NRB - mSRSbMax;
        if gap < NRBGap
            csrs = srs.BandwidthConfigurationTable{ srsNRBb == mSRSbMax ,1};
            bsrs = b;
            NRBGap = gap;
        end
    end
    csrs = csrs(1);

    if srs.BHop < bsrs
        bsrs = srs.BSRS;
    end
end

function out = hSRSNumberOfSubbandsOrHoppingPatterns(srs)
% N = hSRSNumberOfSubbandsOrHoppingPatterns(SRS) returns the number of
% unique SRS subbands or frequency-hopping patterns N configurable by the
% SRS property NRRC. An SRS subband is the frequency band allocated for SRS
% in a given OFDM symbol (See SRS property NRBPerTransmission). N is
% calculated using TS 38.211 Table 6.4.1.4.3-1 for the bandwidth
% configuration parameters CSRS, BSRS, and BHop specified in the SRS
% configuration object SRS.

    bwct = nrSRSConfig.BandwidthConfigurationTable{:,:};
    if srs.BHop < srs.BSRS % Number of unique frequency-hopping patterns
        b0 = srs.BHop+1;
    else % Number of unique SRS subbands
        b0 = 0;
    end
    out = prod(bwct(srs.CSRS+1,2*(b0:srs.BSRS)+3));
end

function [Grid,dispGrid] = hSRSGrid(carrier,srs,Duration,displayGrid,chplevels)
% [GRID,DISPGRID] = hSRSGrid(CARRIER,SRS,DURATION,DISPLAYGRID,CHPLEVELS)
% returns a multi-slot OFDM resource grid GRID containing a set of sounding
% reference signals in a carrier, as specified by the configuration objects
% CARRIER and SRS. This function also returns a scaled version of the grid
% used for display purposes. The optional input DURATION (Default 1)
% specifies the number of slots of the generated grid. The resource grid
% can be displayed using the optional input DISPLAYGRID (Default false).
% CHPLEVELS specifies the channel power levels for display purposes only
% and it must be of the same size as SRS.

    numSRS = length(srs);
    if nargin < 5
        chplevels = 1:-1/numSRS:1/numSRS;
        if nargin < 4
```

```
            displayGrid = false;
            if nargin < 3
                Duration = 1;
            end
        end
    end

    SymbolsPerSlot = carrier.SymbolsPerSlot;
    emptySlotGrid = nrResourceGrid(carrier,max([srs(:).NumSRSPorts])); % Initialize slot grid

    % Create the SRS symbols and indices and populate the grid with the SRS symbols
    Grid = repmat(emptySlotGrid,1,Duration);
    dispGrid = repmat(emptySlotGrid,1,Duration); % Frame-size grid for display
    for ns = 0:Duration-1
        slotGrid = emptySlotGrid;
        dispSlotGrid = emptySlotGrid; % Slot-size grid for display
        for ich = 1:numSRS
            srsIndices = nrSRSIndices(carrier,srs(ich));
            srsSymbols = nrSRS(carrier,srs(ich));
            slotGrid(srsIndices) = srsSymbols;
            dispSlotGrid(srsIndices) = chplevels(ich)*srsSymbols; % Scale the SRS for display on
        end
        OFDMSymIdx = ns*SymbolsPerSlot + (1:SymbolsPerSlot);
        Grid(:,OFDMSymIdx,:) = slotGrid;
        dispGrid(:,OFDMSymIdx,:) = dispSlotGrid;
        carrier.NSlot = carrier.NSlot+1;
    end

    if displayGrid
        plotGrid(dispGrid(:,:,1),chplevels,"SRS " + (1:numSRS)');
    end
end

function varargout = plotGrid(Grid,chplevels,leg)
% plotGrid(GRID, CHPLEVEL,LEG) displays a resource grid GRID containing
% channels or signals at different power levels CHPLEVEL and create a
% legend for these using a cell array of character vector LEG

    if nargin < 3
        leg = {'SRS'};
        if nargin < 2
            chplevels = 1;
        end
    end

    cmap = colormap(gcf);
    chpscale = length(cmap); % Scaling factor

    h = figure;
    image(0:size(Grid,2)-1,(0:size(Grid,1)-1)/12,chpscale*abs(Grid(:,:,1))); % Multiplied with sc
    axis xy;

    title('Carrier Grid Containing SRS')
    xlabel('OFDM Symbol'); ylabel('RB');

    clevels = chpscale*chplevels(:);
    N = length(clevels);
    L = line(ones(N),ones(N),'LineWidth',8); % Generate lines
```

```
        % Index the color map and associate the selected colors with the lines
        set(L,{'color'},mat2cell(cmap( min(1+fix(clevels),length(cmap) ),:),ones(1,N),3)); % Set the

        % Create legend
        legend(leg(:));

        if nargout > 0
            varargout = {h};
        end
    end

    function hSRSAnnotations(carrier,srs)
    % hSRSAnnotations(carrier,srs) adds annotation to the current figure
    % indicating the frequency origin of the SRS and the number of RB used per
    % OFDM symbol for the configuration objects CARRIER and SRS.

        % Calculate the frequency origin of the first SRS symbol
        f0 = hSRSFrequencyOrigin(srs);

        hold on;
        hfig = gcf;
        set(hfig,'Units','Normalized');
        Sym0 = srs.SymbolStart-0.5;
        if isnumeric(srs.SRSPeriod)
            Sym0 = srs.SRSPeriod(2)*carrier.SymbolsPerSlot + srs.SymbolStart-0.5;
        end

        IP = get(gca,'Position');

        % Y-coordinate in the current axes of the SRS freq position f0
        Yf0 = f0/carrier.NSizeGrid*IP(4)+IP(2);
        Xc = Sym0/((carrier.NSlot+1)*carrier.SymbolsPerSlot)*IP(3)+IP(1);

        % Add annotation to the figure including f0 in RB
        if f0/carrier.NSizeGrid > 0.08 % Only plot f0 when there is enough space in the y-axis
            % Create doublearrow for F0
            YMin = IP(2);
            annotation(hfig,'doublearrow',Xc*[1 1], [YMin Yf0],...
                'Color',[1 0.4 0.15]);

            % Text for F0
            str = sprintf('$$F_0 = %d $$ RB', f0);
            Ystr = f0/carrier.NSizeGrid/2;
            text(gca,(Xc-IP(1))/IP(3)-0.25, Ystr,str,...
                'Color',[1 0.4 0.15],'FontSize',14, ...
                'Units','Normalized','Interpreter','latex');
        end

        % Create doublearrow from f0 and spanning the SRS bandwidth
        Yf1 = Yf0 + srs.NRBPerTransmission/carrier.NSizeGrid*IP(4);
        annotation(hfig,'doublearrow',Xc*[1 1], [Yf0 Yf1],...
            'Color',[1 0.4 0.15]);

        % Text for NRBt
        str = sprintf('$$\\textrm{NRB}_T = %d $$ RB', srs.NRBPerTransmission);
        Ystr = (f0 + 0.5*srs.NRBPerTransmission)/carrier.NSizeGrid;
        text(gca,(Xc-IP(1))/IP(3)-0.32, Ystr ,str,...
```

```
        'Color',[1 0.4 0.15],'FontSize',14, ...
        'Units','Normalized','Interpreter','latex');
end

function f0 = hSRSFrequencyOrigin(srs)
% Calculate the frequency origin of the first SRS symbol in a slot

    NSBTable = hSRSNumberOfSubbandsOrHoppingPatterns(srs);
    NRBt = srs.NRBPerTransmission;

    % Origin of the SRS in frequency in RB
    f0 = srs.FrequencyStart + NRBt*mod(floor(4*srs.NRRC/NRBt),NSBTable);
end

function [NRRC,NRB] = hNRRCSet(srs)
% Calculate the values of NRRC that generate a unique set of orthogonal SRS in frequency
    if srs.BHop < srs.BSRS % Frequency-hopping cases
        NRB = srs(1).NRB;  % Hopping bandwidth
    else
        NRB = nrSRSConfig.BandwidthConfigurationTable{srs(1).CSRS+1,2};
    end

    % Number of frequency-hopping patterns or SRS subbands depending on the values of BSRS and BH
    N = hSRSNumberOfSubbandsOrHoppingPatterns(srs);

    NRRC = NRB/4*(0:N-1)/N;
end
```

References

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

See Also

Functions
`nrSRS` | `nrSRSIndices`

Objects
`nrSRSConfig`

Related Examples

- “NR Uplink Channel State Information Estimation Using SRS” on page 2-78

NR Uplink Channel State Information Estimation Using SRS

This example shows how to use sounding reference signals (SRS) for synchronization, channel estimation, and uplink channel state information (CSI) estimation.

Introduction

Sounding reference signals are uplink physical signals used by user equipment (UE) for uplink channel sounding, including synchronization and CSI estimation. CSI comprises channel quality indicator (CQI), rank indicator (RI), and precoder matrix indicator (PMI). This example demonstrates how to use SRS to select the appropriate PMI under frequency-selective time-varying noisy channels. Uplink codebook-based transmissions use PMIs, as defined in TS 38.211 Section 6.3.1.5 [1].

This example performs a simulation including:

- SRS configuration and transmission
- Perfect and practical synchronization and channel estimation
- Signal-to-noise ratio (SNR) estimation
- PMI selection
- PMI selection performance assessment

Simulation Length and SNR

Set the length of the simulation in terms of the number of 10 ms frames. Set the SNR points to simulate. The SNR is defined per RE and applies to each receive antenna.

```
numFrames = 1; % 10 ms frames
snr = 20; % SNR in dB
```

UE and SRS Configuration

Set the key parameters of the simulation. These include:

- The bandwidth in resource blocks (12 subcarriers per resource block)
- Subcarrier spacing: 15, 30, 60, 120, 240 (kHz)
- Cyclic prefix length: normal or extended
- Number of transmit and receive antennas: 1, 2 or 4.
- Number of layers. It must be lower than or equal to the number of transmit and receive antennas.

The SRS parameters specified include:

- Number of SRS antenna ports: 1,2,4
- Number of OFDM symbols allocated for SRS per slot: 1,2,4
- Starting OFDM symbol of the SRS transmission within a slot. It must be (8...13) for normal CP and (6...11) for extended CP.
- Starting position of the SRS in frequency specified in RBs
- Bandwidth and frequency hopping configuration `CSRS`, `BSRS`, and `BHop`. Set `BHop >= BSRS` to disable frequency hopping.
- Transmission comb to specify the SRS frequency density in subcarriers: 2,4

- Number of repeated SRS symbols within a slot. It disables frequency hopping in blocks of `Repetition` symbols. Set `Repetition = 1` for no repetition.
- Periodicity and offset of the SRS in slots.
- Resource type can be 'periodic', 'semi-persistent', and 'aperiodic'. The frequency hopping pattern is reset for aperiodic SRS resource types in every slot.

```
% Create UE/carrier configuration
ue = nrCarrierConfig;
ue.NSizeGrid = 52;          % Bandwidth in number of resource blocks (52RBs at 15kHz SCS for 10MH
ue.SubcarrierSpacing = 15;  % 15, 30, 60, 120, 240 (kHz)
ue.CyclicPrefix = 'Normal'; % 'Normal' or 'Extended'

nTxAnts = 2;  % Number of transmit antennas (1,2,4)
nRxAnts = 2;  % Number of receive antennas
nLayers = min(nTxAnts,nRxAnts);

% Configure a periodic multi-port SRS and enable frequency hopping
srs = nrSRSConfig;
srs.NumSRSSymbols = 4;          % Number of OFDM symbols allocated per slot (1,2,4)
srs.SymbolStart = 8;            % Starting OFDM symbol within a slot
srs.NumSRSPorts = nTxAnts;      % Number of SRS antenna ports (1,2,4).
srs.FrequencyStart = 0;         % Frequency position of the SRS in BWP in RBs
srs.NRRC = 0;                   % Additional offset from FreqStart specified in blocks of 4 PRBs
srs.CSRS = 14;                  % Bandwidth configuration C_SRS (0...63). It controls the allocat
srs.BSRS = 0;                   % Bandwidth configuration B_SRS (0...3). It controls the allocate
srs.BHop = 0;                   % Frequency hopping configuration (0...3). Set BHop < BSRS to ena
srs.KTC = 2;                    % Comb number (2,4). Frequency density in subcarriers
srs.Repetition = 2;             % Repetition (1,2,4). It disables frequency hopping in blocks of
srs.SRSPeriod = [2 0];          % Periodicity and offset in slots. SRSPeriod(2) must be < SRSPeri
srs.ResourceType = 'periodic';  % Resource type ('periodic', 'semi-persistent','aperiodic'). Use
```

Synchronization, Channel Estimation and CSI Measurement Configuration

This example performs synchronization and channel estimation in SRS candidate slots. Timing and channel estimates are updated only in slots containing SRS transmissions. In frequency-hopping SRS setups, channel estimates are only updated in those resource blocks containing SRS symbols. When there is no SRS transmission, timing and channel estimates from previous slots are held and used for CSI acquisition. Similarly, noise power estimates are updated only in SRS candidate slots.

The logical variable `practicalSynchronization` controls channel synchronization behavior. When set to `true`, the example performs practical synchronization based on the values of the received SRS. When set to `false`, the example performs perfect synchronization. Synchronization is performed only in slots where the SRS is transmitted to keep perfect and practical channel estimates synchronized.

```
practicalSynchronization = true;
```

This example estimates CSI by splitting the carrier bandwidth into a number of subbands. Specify the size of the frequency subbands in RB

```
csiSubbandSize = 4; % Number of RBs per subband
```

Propagation Channel Model Configuration

Create a TDL channel model object and specify its propagation characteristics. Select the channel delay spread and maximum Doppler shift to create a time-varying and frequency-selective channel within the simulation duration and carrier bandwidth.

```
channel = nrTDLChannel;
channel.DelayProfile = 'TDL-C';
channel.DelaySpread = 40e-9;
channel.MaximumDopplerShift = 30;
channel.NumTransmitAntennas = nTxAnts;
channel.NumReceiveAntennas = nRxAnts;
channel.Seed = 5;

% Set channel sample rate
ofdmInfo = nrOFDMInfo(ue);
channel.SampleRate = ofdmInfo.SampleRate;

% Get the maximum delay of the channel
chInfo = info(channel);
maxChDelay = chInfo.MaximumChannelDelay;

% Reset random generator for reproducibility
rng('default');
```

Processing Loop

Measure the CSI per slot. The CSI is obtained using the following steps:

- *Generate resource grid.* Use SRS symbols and indices to create a resource element (RE) grid.
- *Generate waveform.* The generated grid is then OFDM modulated.
- *Model noisy channel.* The waveform is passed through a TDL fading channel. AWGN is added. The SNR for each layer is defined per RE and per receive antenna.
- *Perform synchronization and OFDM demodulation.* For perfect synchronization, the channel impulse response is reconstructed and used to synchronize the received waveform. For practical synchronization, the received waveform is correlated with the SRS. The synchronized signal is then OFDM demodulated.
- *Perform channel estimation.* For perfect channel estimation, the channel impulse response is reconstructed and OFDM demodulated to provide a channel estimate. For practical channel estimation, the transmitted SRS is used.
- *PMI selection.* The SRS-based channel estimate is used to select the best PMI in each CSI estimation subband. The PMI selection criterion maximizes the average signal-to-interference-plus-noise-ratio (SINR) after precoding.
- *PMI selection SINR loss.* The SINR loss is calculated by comparing the SINR after precoding with the estimated and ideal PMIs. Ideal PMIs are selected using perfect channel estimates.

```
% Number of slots to simulate
numSlots = numFrames*ue.SlotsPerFrame;

% Total number of subcarriers and symbols per slot
K = ue.NSizeGrid * 12;
L = ue.SymbolsPerSlot;

% Initialize arrays storing channel estimates
allTxGrid = zeros([K L*numSlots nTxAnts]);

slotGridSize = [K L nRxAnts nTxAnts];
hEst = zeros(slotGridSize);
hestInterp = zeros(slotGridSize);
hEstUpdate = zeros(slotGridSize);
```

```
totalGridSize = [K L*numSlots nRxAnts nTxAnts];
allHest = zeros(totalGridSize);
allHestPerfect = zeros(totalGridSize);
allHestInterp = zeros(totalGridSize);

% Initialize noise power estimate
nvar = 0;

% Calculate the number of CSI subbands for the carrier
numCSISubbands = ceil(ue.NSizeGrid/csiSubbandSize);

% Initialize SINR per subband, slot, and PMI
maxPMI = hMaxPUSCHPrecodingMatrixIndicator(nLayers,nTxAnts);
sinrSubband = zeros([numCSISubbands numSlots maxPMI+1]);

% Initialize PMI matrix and SINR loss
pmi = NaN(numCSISubbands,numSlots);
pmiPerfect = pmi;
loss = zeros(size(pmi));

% Initialize timing estimation offset
offset = chInfo.ChannelFilterDelay;

% Calculate SRS CDM lengths
cdmLengths = hSRSCDMLengths(srs);

% OFDM symbols used for CSI acquisition
csiSelectSymbols = srs.SymbolStart + (1:srs.NumSRSSymbols);

for nSlot = 0:numSlots-1

    % Update slot counter
    ue.NSlot = nSlot;

    % Generate SRS and map to slot grid
    [srsIndices,srsIndInfo] = nrSRSIndices(ue,srs);
    srsSymbols = nrSRS(ue,srs);

    % Create a slot-wise resource grid empty grid and map SRS symbols
    txGrid = nrResourceGrid(ue,nTxAnts);
    txGrid(srsIndices) = srsSymbols;

    % Determine if the slot contains SRS
    isSRSSlot= ~isempty(srsSymbols);

    % OFDM Modulation
    [txWaveform,waveformInfo] = nrOFDMModulate(ue,txGrid);

    txWaveform = [txWaveform; zeros(maxChDelay, size(txWaveform,2))]; %#ok<AGROW> % required late

    % Transmission through channel
    [rxWaveform,pathGains] = channel(txWaveform);

    % Add AWGN to the received time domain waveform
    % Normalize noise power to take account of sampling rate, which is
    % a function of the IFFT size used in OFDM modulation. The SNR
    % is defined per RE for each receive antenna (TS 38.101-4).
```

```
SNR = 10^(snr/10);
N0 = 1/sqrt(2.0*nRxAnts*double(waveformInfo.Nfft)*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));

rxWaveform = rxWaveform + noise;

% Perform timing offset estimation
pathFilters = getPathFilters(channel);

% Timing estimation is only performed in the slots where the SRS is
% transmitted to keep the perfect and practical channel estimation
% synchronized.
if isSRSSlot
    if practicalSynchronization
        % Practical synchronization. Correlate the received waveform
        % with the SRS to give timing offset estimate
        offset = nrTimingEstimate(ue,rxWaveform,srsIndices,srsSymbols);
    else
        offset = nrPerfectTimingEstimate(pathGains,pathFilters); %#ok<UNRCH>
    end
end

% Perform OFDM demodulation
rxGrid = nrOFDMDemodulate(ue,rxWaveform(1+offset:end,:));

% Perform practical channel estimation
% Update channel estimates only in the symbols and RBs containing SRS
% in this slot and hold the estimates from the previous slot in other
% locations. nvar is not updated when there is no SRS transmission. Use
% a time-averaging window that covers all the SRS symbols transmitted.
if isSRSSlot % this slot contains an SRS transmission
    [hEst,nvar] = nrChannelEstimate(ue,rxGrid,srsIndices,srsSymbols,'AveragingWindow',[0 7],

    % Use channel estimate from previous slot for OFDM symbols before the first SRS symbol
    hestInterp = repmat(hestInterp(:,end,:,:),1,ue.SymbolsPerSlot);

    % Update channel estimate in OFDM symbols and RB where the SRS is
    % present and hold all channel estimates until the end of the slot
    firstSymbol = srs.SymbolStart+1;
    lastSymbol = srs.SymbolStart + srs.NumSRSSymbols;
    hEstUpdate(:,firstSymbol:lastSymbol,:,:) = hEst(:,firstSymbol:lastSymbol,:,:);
    hEstUpdate(:,lastSymbol:L,:,:) = repmat(hEst(:,lastSymbol,:,:),1,ue.SymbolsPerSlot-lastSy

    idxHEstUpdate = hEstUpdate ~= 0; % Indices of updated channel estimates
    hestInterp(idxHEstUpdate) = hEstUpdate(idxHEstUpdate);
else % Hold previous channel estimates if this slot does not contain SRS
    hestInterp = repmat(hestInterp(:,end,:,:),1,ue.SymbolsPerSlot);
end

% PMI Selection
% Select the precoder matrix indicators for a number of layers using
% the interpolated channel estimates in the OFDM symbols specified by
% csiSelectSymbols. The PMIs are estimated per CSI subband
[pmi(:,nSlot+1),sinrSubband(:,nSlot+1,:),subbandIndices] = hPMISelect(nLayers, hestInterp(:,

% PMI selection SINR loss
% Calculate the performance loss as a ratio of the SINR after precoding
% with PMIs selected using a practical channel estimate and the SINR
```

```
    % after precoding with PMIs selected using a perfect channel estimate.

    % Calculate perfect channel estimate for perfect PMI selection
    hEstPerfect = nrPerfectChannelEstimate(ue,pathGains,pathFilters,offset);

    % Perfect noise estimate from noise realization
    noiseGrid = nrOFDMDemodulate(ue,noise(1+offset:end,:));
    nvarPerfect = var(noiseGrid(:));

    [loss(:,nSlot+1),pmiPerfect(:,nSlot+1)] = hPMISelectionSINRLoss(pmi(:,nSlot+1), nLayers,  hEs

    % Save a copy of all transmitted OFDM grids and channel estimates for
    % display purposes
    thisSlot = nSlot*L + (1:L); % Symbols of the current slot
    allTxGrid(:,thisSlot,:) = txGrid;
    allHest(:,thisSlot,:,:) = hEst;
    allHestInterp(:,thisSlot,:,:) = hestInterp;
    allHestPerfect(:,thisSlot,:,:) = hEstPerfect;

end
```

Results

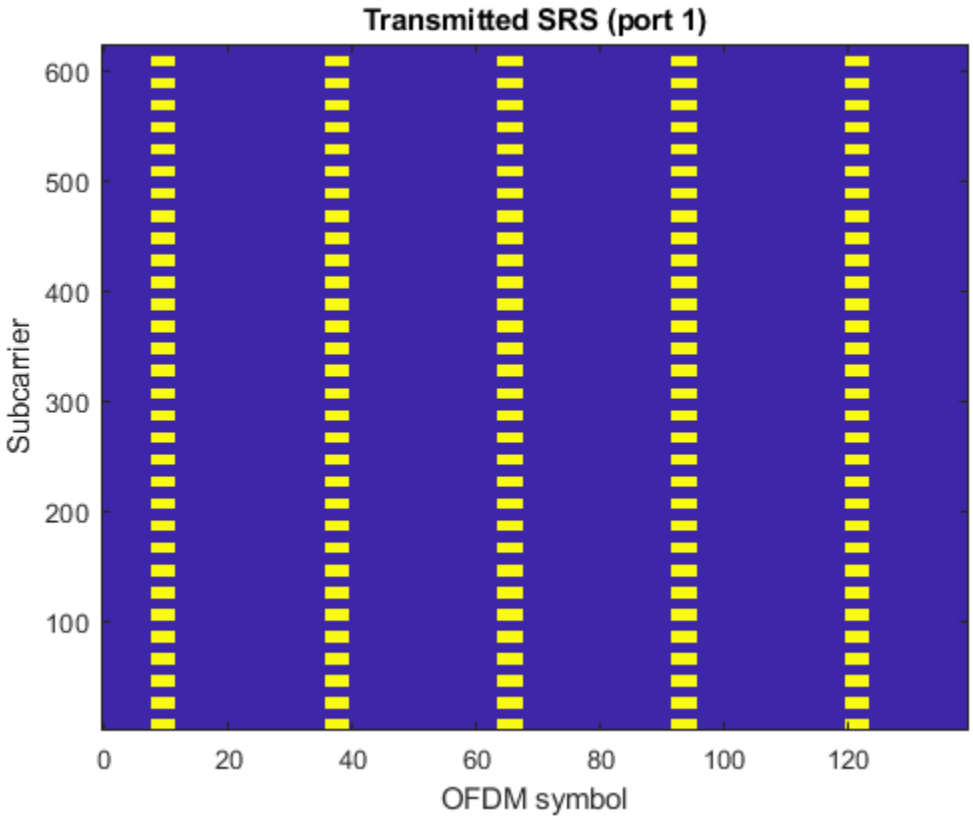
This section displays these results for all configured frames:

- Transmitted OFDM grid containing SRS
- Perfect and practical channel estimates, and channel estimation error. The error is calculated as the absolute value of the difference between the perfect and practical channel estimates.
- Selected PMI using perfect and practical channel estimates, and the PMI absolute error.
- Average SINR per subband after precoding with best estimated PMI
- SINR performance loss

```
% Create x-axis and y-axis vectors
symbols = 0:(ue.NSlot+1)*ue.SymbolsPerSlot-1;
slots = 0:ue.NSlot;
subcarriers = 1:ue.NSizeGrid*12;
resourceBlocks = 1:ue.NSizeGrid;
```

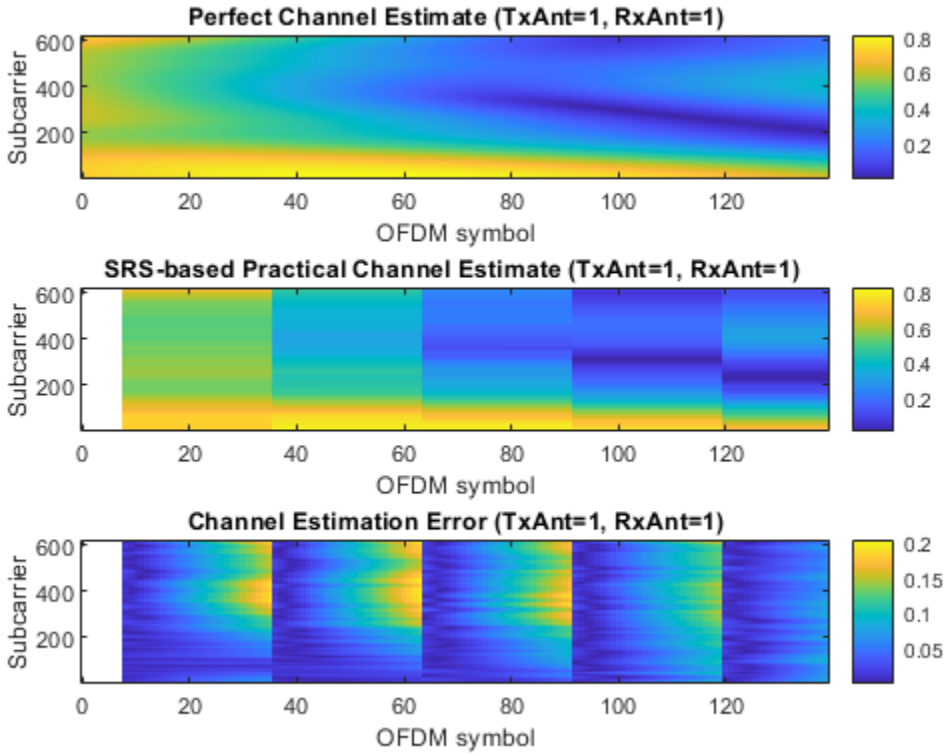
Display the transmitted OFDM grid containing SRS

```
figure
imagesc(symbols,subcarriers,abs(allTxGrid(:,:,1,1)));
xlabel('OFDM symbol'); ylabel('Subcarrier'); axis xy;
title('Transmitted SRS (port 1)');
```



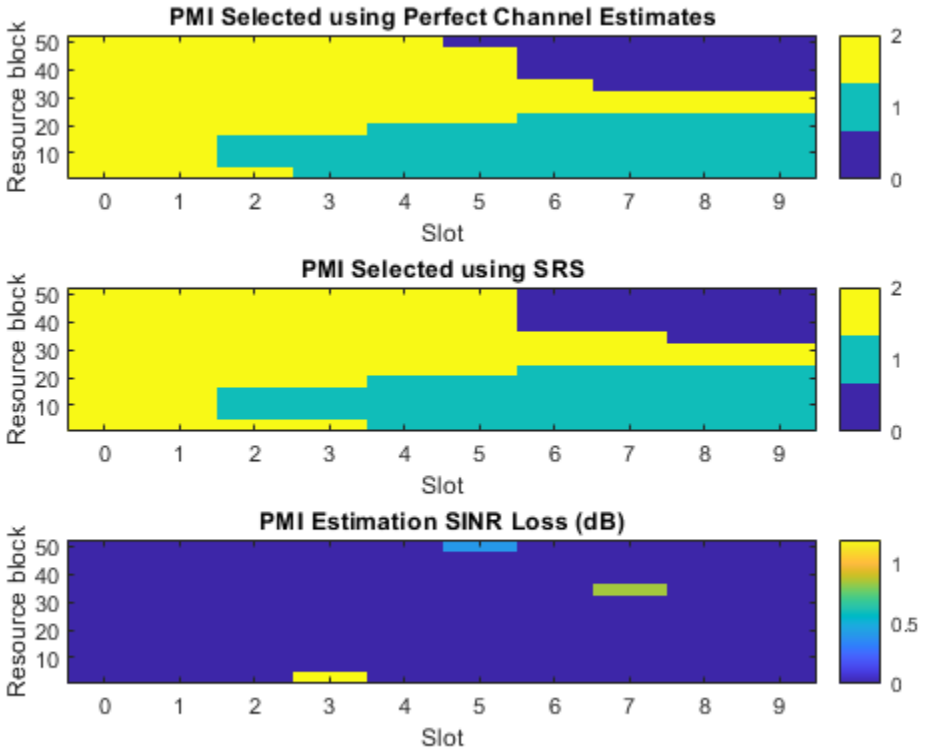
Display perfect and practical channel estimates, and channel estimation error per OFDM symbol and subcarrier. The channel estimation error is defined as the absolute value of the difference between the perfect and practical channel estimates.

```
% Remove first OFDM symbols not containing SRS to improve visualization of
% channel estimation error
hEstInterp = allHestInterp;
idx = 1:(srs.SRSPeriod(2)*ue.SymbolsPerSlot + srs.SymbolStart);
hEstInterp(:,idx,:,:) = NaN;
hEstInterp((srs.NRB*12+1):end,:,:,:) = NaN;

hChannelEstimationPlot(symbols,subcarriers,allHestPerfect,hEstInterp);
```



Display the selected PMI using perfect and practical channel estimates, and the PMI selection SINR loss per slot and RB. The SINR loss is defined as a ratio of the SINR after precoding with estimated and perfect PMIs. Estimated PMIs are obtained using a practical channel estimate and perfect PMIs are selected using a perfect channel estimate.

```
% First expand loss from subbands into RBs for display purposes
pmiRB = hExpandSubbandToRB(pmi, csiSubbandSize, ue.NSizeGrid);
pmiPerfectRB = hExpandSubbandToRB(pmiPerfect, csiSubbandSize, ue.NSizeGrid);
lossRB = hExpandSubbandToRB(loss, csiSubbandSize, ue.NSizeGrid);

hPMIPlot(slots,resourceBlocks,pmiRB,pmiPerfectRB,lossRB);
```



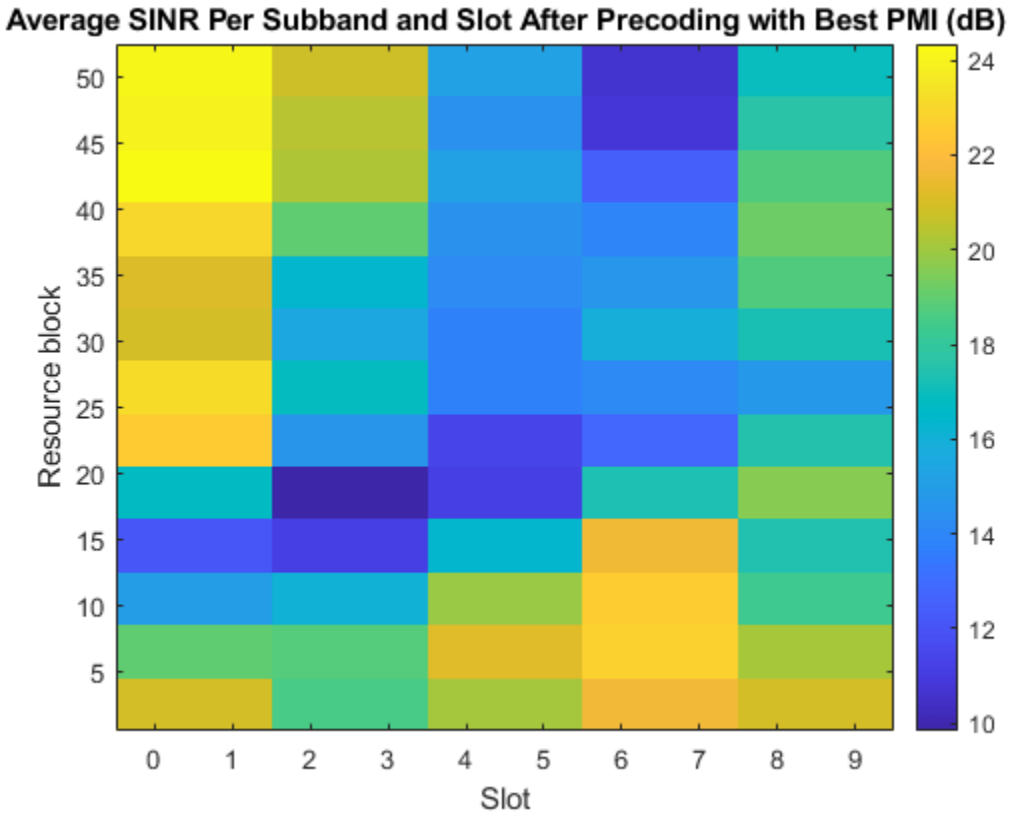
Next, compare the PMIs obtained using both perfect and practical channel estimates. This indicates the ratio of correct to total PMIs and the location in the resource grid where errors have occurred.

```
numLayers = min(size(allHestInterp,[3 4]));
if numLayers ~= 1
    pmiErr = sum( abs(pmi - pmiPerfect) > 0, [1 2])./ sum( ~isnan(pmi), [1 2]);

    TotPMIEst = sum(~isnan(pmi),[1 2]);
    fprintf('Number of estimated PMI: %d \n', TotPMIEst);
    fprintf('    Number of wrong PMI: %d \n', ceil(pmiErr*TotPMIEst));
    fprintf('         Relative error: %.1f (%%) \n', pmiErr*100);
else
    fprintf('For a single layer, PMI is always 0.\n');
end
```

```
Number of estimated PMI: 130
    Number of wrong PMI: 3
         Relative error: 2.3 (%)
```

Display the SINR per slot and RB obtained after precoding with the PMI that maximizes the SINR per subband.

```
hBestSINRPlot(slots,resourceBlocks,sinrSubband,pmi,csiSubbandSize);
```



Summary and Further Exploration

This example shows how to use SRS for synchronization, channel estimation and PMI selection commonly employed in codebook-based uplink transmission modes. The example also evaluates the channel estimation and PMI selection performance loss using the SINR after precoding.

You can investigate the performance of the channel estimation and PMI selection in more complex settings. Increase the SRS periodicity and observe how the channel estimates aging introduces a delay and worsens the SINR after precoding. In addition, you can investigate the performance under frequency hopping conditions by setting the SRS parameters BHop < BSRS. In this setup, channel estimates aging is not uniform in frequency.

Appendix

This example uses the following helper functions:

- hMaxPUSCHPrecodingMatrixIndicator.m
- hPMISelect.m
- hPMISelectionSINRLoss.m
- hPrecodedSINR.m
- hSINRPerSubband.m
- hSRSCDMLengths.m

References

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

2 3GPP TS 38.101-4. "NR; User Equipment (UE) radio transmission and reception. Part 4: Performance requirements" 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

Local functions

```
% Displays perfect and practical channel estimates and the channel
% estimation error for the first transmit and receive ports. The channel
% estimation error is defined as the absolute value of the difference
% between the perfect and practical channel estimates.
function hChannelEstimationPlot(symbols,subcarriers,allHestPerfect,allHestInterp)

    figure
    subplot(311)
    imagesc(symbols, subcarriers, abs(allHestPerfect(:,:,1,1)));
    axis xy; xlabel('OFDM symbol'); ylabel('Subcarrier');
    colorbar;
    title('Perfect Channel Estimate (TxAnt=1, RxAnt=1)');

    subplot(312)
    imagesc(symbols, subcarriers, abs(allHestInterp(:,:,1,1)), ...
            'AlphaData',~isnan(allHestInterp(:,:,1,1)))
    axis xy; xlabel('OFDM symbol'); ylabel('Subcarrier');
    colorbar;
    title('SRS-based Practical Channel Estimate (TxAnt=1, RxAnt=1) ');

    % Display channel estimation error, defined as the difference between the
    % SRS-based and perfect channel estimates
    subplot(313)
    hestErr = abs(allHestInterp - allHestPerfect);
    imagesc(symbols, subcarriers, hestErr(:,:,1,1),...
            'AlphaData',~isnan(hestErr(:,:,1,1)));
    axis xy; xlabel('OFDM symbol'); ylabel('Subcarrier');
    colorbar;
    title('Channel Estimation Error (TxAnt=1, RxAnt=1)');

end

% Displays the PMI evolution and PMI estimation SINR loss over time and
% frequency. The SINR loss is defined as a ratio of the SINR after
% precoding with estimated and perfect PMIs. Estimated PMIs are
% obtained using a practical channel estimate and perfect PMIs are
% selected using a perfect channel estimate.
function hPMIPlot(slots,resourceBlocks,pmiRB,pmiPerfectRB,lossRB)

    figure
    subplot(311)
    imagesc(slots,resourceBlocks,pmiPerfectRB,'AlphaData',~isnan(pmiPerfectRB)); axis xy;
    c = clim;
    cm = colormap;
    colormap( cm(1:floor(size(cm,1)/(c(2)-c(1)) -1):end,:) ); % Adjust colormap to PMI discrete v
    colorbar
    xlabel('Slot'); ylabel('Resource block'), title('PMI Selected using Perfect Channel Estimates
```

```
    subplot(312)
    imagesc(slots,resourceBlocks,pmiRB,'AlphaData',~isnan(pmiRB)); axis xy;
    colorbar,
    xlabel('Slot'); ylabel('Resource block'), title('PMI Selected using SRS')

    subplot(313)
    imagesc(slots,resourceBlocks,lossRB,'AlphaData',~isnan(lossRB));
    colormap(gca,cm)
    xlabel('Slot'); ylabel('Resource block'); axis xy; colorbar;
    title('PMI Estimation SINR Loss (dB)')

end

% Displays the SINR per resource block obtained after precoding with the
% PMI that maximizes the SINR per subband.
function hBestSINRPlot(slots,resourceBlocks,sinrSubband,pmi,csiBandSize)

    % Display SINR after precoding with best PMI
    bestSINRPerSubband = nan(size(sinrSubband,[1 2]));

    % Get SINR per subband and slot using best PMI
    [sb,nslot] = find(~isnan(pmi));
    for i = 1:length(sb)
        bestSINRPerSubband(sb(i),nslot(i)) = sinrSubband(sb(i),nslot(i),pmi(sb(i),nslot(i))+1);
    end

    % First expand SINR from subbands into RBs for display purposes
    bestSINRPerRB = hExpandSubbandToRB(bestSINRPerSubband, csiBandSize, length(resourceBlocks));

    figure
    sinrdb = 10*log10(abs(bestSINRPerRB));
    imagesc(slots,resourceBlocks,sinrdb,'AlphaData',~isnan(sinrdb));
    axis xy; colorbar;
    xlabel('Slot');
    ylabel('Resource block')
    title('Average SINR Per Subband and Slot After Precoding with Best PMI (dB)')

end

% Expands a 2D matrix of values per subband in the first dimension into a
% matrix of values per resource block.
function rbValues = hExpandSubbandToRB(subbandValues, bandSize, NRB)

    lastBandSize = mod(NRB,bandSize);
    lastBandSize = lastBandSize + bandSize*(lastBandSize==0);

    rbValues = [kron(subbandValues(1:end-1,:),ones(bandSize,1));...
                subbandValues(end,:).*ones(lastBandSize,1)];
end
```

See Also

Functions
nrSRS | nrSRSIndices

Objects
`nrSRSConfig`

Related Examples

- “NR SRS Configuration” on page 2-59

5G NR PRACH Configuration

This example shows how to configure the 5G New Radio (NR) physical random access channel (PRACH), as defined in TS 38.211 Sections 5.3.2 and 6.3.3. You can learn about PRACH time resources, their relation to PRACH preambles, and learn how to generate PRACH preambles without the need to look up configuration tables. This example also shows how to map PRACH symbols to the resource grid, and how to generate a time-domain waveform for a single PRACH preamble.

Configure Carrier and PRACH

Supported Combinations of Subcarrier Spacing

Table 6.3.3.2-1 in TS 38.211 lists the supported combinations of subcarrier spacing for the PRACH and the physical uplink shared channel (PUSCH) during initial access. You can access this table directly from the PRACH configuration object.

```
disp(nrPRACHConfig.Tables.SupportedSCSCombinations)
```

LRA	PRACHSubcarrierSpacing	PUSCHSubcarrierSpacing	NRBAllocation	kbar
839	1.25	15	6	7
839	1.25	30	3	1
839	1.25	60	2	133
839	5	15	24	12
839	5	30	12	10
839	5	60	6	7
139	15	15	12	2
139	15	30	6	2
139	15	60	3	2
139	30	15	24	2
139	30	30	12	2
139	30	60	6	2
139	60	60	12	2
139	60	120	6	2
139	120	60	24	2
139	120	120	12	2
139	120	480	3	1
139	120	960	2	23
139	480	120	48	2
139	480	480	12	2
139	480	960	6	2
139	960	120	96	2
139	960	480	24	2
139	960	960	12	2
571	30	15	96	2
571	30	30	48	2
571	30	60	24	2
571	120	120	48	2
571	120	480	12	1
571	120	960	7	47
571	480	120	192	2
571	480	480	48	2
571	480	960	24	2
1151	15	15	96	1
1151	15	30	48	1

```
    1151              15                  60                  24           1
    1151             120                 120                  97           6
    1151             120                 480                  25          23
    1151             120                 960                  13          45
```

The system information block 1 (SIB1) contains the radio resource control (RRC) information element *UplinkConfigCommonSIB* (TS 38.331 Section 6.3.2) that defines the subcarrier spacing for the PUSCH. The user equipment (UE) needs this information to transmit the PRACH preamble during the random-access procedure.

Carrier Configuration

Because the PUSCH is not defined at the PRACH preamble transmission, to configure the PUSCH subcarrier spacing and the frequency-domain dimensions of the resource grid, use the `nrCarrierConfig` object.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15; % Subcarrier spacing in kHz (15, 30, 60, 120, 480, 960)
```

Because the PRACH preamble is modulated with respect to the carrier, changing the carrier subcarrier spacing leads to a different PRACH waveform. To see how a different carrier affects the generated waveform, check the `Information associated with PRACH OFDM modulation` output for several carriers in the Generate Waveform for Single PRACH Preamble on page 2-101 section, .

PRACH Configuration

You can configure PRACH parameters by setting property values of the `nrPRACHConfig` object. According to TS 38.211, not all PRACH parameter combinations are valid. For more information on how the properties of `nrPRACHConfig` reflect these limitations, see `nrPRACHConfig`.

```
prach = nrPRACHConfig;
prach.FrequencyRange = 'FR1';            % Frequency range ('FR1', 'FR2')
prach.DuplexMode = 'FDD';                % Duplex mode ('FDD', 'TDD', 'SUL')
prach.ConfigurationIndex = 27;           % Configuration index (0...262). This value is automatic
prach.SubcarrierSpacing = 15;            % Subcarrier spacing in kHz (1.25, 5, 15, 30, 60, 120, 4
prach.SequenceIndex = 0;                 % Logical root sequence index (0...1149)
prach.PreambleIndex = 0;                 % Scalar preamble index within the cell (0...63)
prach.RestrictedSet = 'UnrestrictedSet'; % Type of restricted set ('UnrestrictedSet','RestrictedS
prach.ZeroCorrelationZone = 0;           % Cyclic shift configuration index (0...15)
prach.RBOffset = 0;                      % Starting resource block index of the initial uplink ba
prach.FrequencyStart = 0;                % Frequency offset of lowest PRACH transmission occasion
prach.FrequencyIndex = 0;                % Index of the PRACH transmission occasions in frequency
prach.TimeIndex = 0;                     % Index of the PRACH transmission occasions in time doma
                                         % For formats B2 and B3, this value is automatically upd

prach.ActivePRACHSlot = 0;               % Active PRACH slot number within a subframe or a 60 kHz
prach.NPRACHSlot = 0;                    % PRACH slot number
```

You can also modify `prach.LRA` to use a value of the Zadoff-Chu preamble sequence length introduced in Release 16 of the 3GPP specifications.

The `ConfigurationIndex` and `TimeIndex` properties depend on the PRACH format. The `SubcarrierSpacing`, `ActivePRACHSlot`, and `NPRACHSlot` properties determine whether the PRACH preamble is active. The next two sections discuss how to set these properties.

How to Set `ConfigurationIndex` Based on Preferred Format

Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211 define all possible PRACH configurations in the time domain. The combination of frequency range and duplex mode specifies which configuration table to use. Valid combinations are:

- FR1 and FDD (paired spectrum): Table 6.3.3.2-2
- FR1 and SUL (supplementary uplink): Table 6.3.3.2-2
- FR1 and TDD (unpaired spectrum): Table 6.3.3.2-3
- FR2 and TDD (unpaired spectrum): Table 6.3.3.2-4

For more information on how paired and unpaired spectrums relate to duplex mode, see the field *FDD-OrSUL* of the RRC information element *FrequencyInfoUL* in TS 38.331 Section 6.3.2.

You can access these configuration tables through the `Tables` property of the `nrPRACHConfig` object. For example:

```
nrPRACHConfig.Tables.ConfigurationsFR1PairedSUL % TS 38.211 Table 6.3.3.2-2
nrPRACHConfig.Tables.ConfigurationsFR1Unpaired  % TS 38.211 Table 6.3.3.2-3
nrPRACHConfig.Tables.ConfigurationsFR2          % TS 38.211 Table 6.3.3.2-4
```

TS 38.211 defines and categorizes 13 PRACH formats as long or short preambles. Long preambles have a sequence of length $L_{RA} = 839$, whereas short preambles have a sequence of length $L_{RA} = 139, 571, 1151$. The values 571 and 1151 are associated with shared spectrum channel access that is introduced in Release 16 of the 3GPP specifications. The formats associated with long preambles are: 0, 1, 2, 3. The formats associated with short preambles are: A1, A2, A3, B1, B2, B3, B4, C0, C2, including mixed formats A1/B1, A2/B2, and A3/B3.

The configuration indices in Tables 6.3.3.2-2 to 6.3.3.2-4 define the time resources in which each preamble format can be transmitted. Each preamble format is associated with several configuration indices. You can choose a PRACH format without the need to look up the configuration tables by setting the value of `ConfigurationIndex` based on the preferred format. This value corresponds to the largest range of time resources in which you can transmit the preferred preamble format.

```
format = B2; % PRACH preamble format ('0','1','2','3','A1','A2','A3','B1','B2
```

Select the configuration table based on `FrequencyRange` and `DuplexMode`.

```
if strcmpi(prach.FrequencyRange,'FR1')
    if strcmpi(prach.DuplexMode,'TDD') % TS 38.211 Table 6.3.3.2-3
        configTable = nrPRACHConfig.Tables.ConfigurationsFR1Unpaired;
    else % TS 38.211 Table 6.3.3.2-2
        configTable = nrPRACHConfig.Tables.ConfigurationsFR1PairedSUL;
    end
else % TS 38.211 Table 6.3.3.2-4
    configTable = nrPRACHConfig.Tables.ConfigurationsFR2;
end
```

Among all configurations corresponding to the same short preamble format in Table 6.3.3.2-2, the second to last configuration has the largest number of time resources for transmitting the PRACH preamble. In all the other cases, including mixed formats in Table 6.3.3.2-2, the last configuration has the largest number of time resources for transmitting the PRACH preamble. This example uses this information to set the value of the `ConfigurationIndex` property. If you select format B2 or B3, this example sets the maximum value of `TimeIndex`.

```
if strcmpi(prach.FrequencyRange,'FR1') && strcmpi(prach.DuplexMode,'FDD') && ...
        any(strcmpi(format,{'A1','A2','A3','B1','B4','C0','C2'}))
    prach.ConfigurationIndex = find(strcmpi(configTable.PreambleFormat,format),1,'last') - 2;
else
    if ~any(strcmpi(format,{'B2','B3'}))
        prach.ConfigurationIndex = find(strcmpi(configTable.PreambleFormat,format),1,'last') - 1
    else
        % Format B2 and B3 only appear in mixed formats, so select an
        % appropriate mixed format and set the maximum value of TimeIndex
        prach.ConfigurationIndex = find(endsWith(configTable.PreambleFormat,format),1,'last') - 1
        prach.TimeIndex = prach.NumTimeOccasions - 1;
    end
end
```

How to Select SubcarrierSpacing, ActivePRACHSlot, and NPRACHSlot to Generate Active PRACH Preamble

Tables 6.3.3.2-2 to 6.3.3.2-4 in TS 38.211 describe which PRACH slot corresponds to an active PRACH preamble. The third and fourth columns of these tables represent the system frame numbers that correspond to an active PRACH preamble. Depending on the selected frequency range, FR1 or FR2, the fifth column represents the slot numbers for 15 kHz or 60 kHz subcarrier spacing, respectively, corresponding to an active PRACH preamble. If a PRACH preamble is not active in the current time resources, no time transmission can take place.

For example, the selected PRACH configuration is active in any system frame and subframe if PRACH subcarrier spacing is set to 15 kHz, as shown in Table 6.3.3.2-2.

```
disp(configTable(prach.ConfigurationIndex+1,:))
```

ConfigurationIndex	PreambleFormat	x	y	SubframeNumber	StartingSymb
146	{'A2/B2'}	1	{[0]}	{[0 1 2 3 4 5 6 7 8 9]}	0

To verify that the PRACH preamble is active in the current slot, check the `prachSymbols` output of the `nrPRACH` function. This output is empty if the PRACH preamble is not active in the current slot. To generate an active PRACH preamble, loop through the values of the `NPRACHSlot` property until `prachSymbols` becomes nonempty.

The cases presented in this section show how to check whether the current PRACH short preamble is active. Both cases consider a PRACH short preamble format B2. If you change the format, the PRACH preamble may be active for values of the `NPRACHSlot` and `ActivePRACHSlot` properties different from those shown in this example.

Case 1: Typical PRACH subcarrier spacing configuration

Set up a PRACH preamble for the selected format with a typical subcarrier spacing configuration. This example considers a 15 kHz subcarrier spacing, which is the typical value for short preambles in FR1. If you change the value of the subcarrier spacing or the format, you may need to change the values of `ActivePRACHSlot` and `NPRACHSlot` to get an active PRACH slot.

```
% Store the user-defined configuration
subcarrierSpacing = prach.SubcarrierSpacing;
activePRACHSlot = prach.ActivePRACHSlot;
nPRACHSlot = prach.NPRACHSlot;

% Set values of SubcarrierSpacing, ActivePRACHSlot, and NPRACHSlot for this
```

```
% case
if any(strcmpi(format,{'0','1','2'}))
    prach.SubcarrierSpacing = 1.25;
elseif strcmpi(format,'3')
    prach.SubcarrierSpacing = 5;
else % Short preambles
    if strcmpi(prach.FrequencyRange,'FR1')
        prach.SubcarrierSpacing = 15; % Valid values: 15, 30
    else % FR2
        prach.SubcarrierSpacing = 60; % Valid values: 60, 120, 480, 960
    end
end
prach.ActivePRACHSlot = 0;
prach.NPRACHSlot = 0;
```

According to Table 6.3.3.2-2 in TS 38.211, the UE can transmit PRACH in any slot.

```
prachSymbols = nrPRACH(carrier,prach);
active = ~isempty(prachSymbols);
disp(['active: ' num2str(active)])
```

```
active: 1
```

Case 2: Alternative PRACH subcarrier spacing configuration

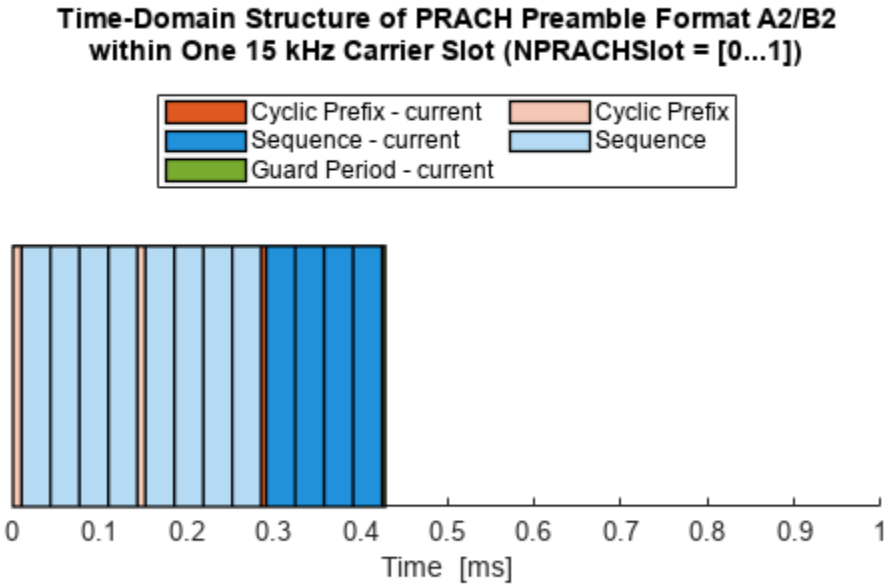
Set the PRACH subcarrier spacing to 30 kHz and use the default value of 15 kHz for the carrier subcarrier spacing. This means that each carrier slot contains two PRACH slots. This case does not consider PRACH long preambles and frequency range FR2 because they are not compatible with 30 kHz subcarrier spacing.

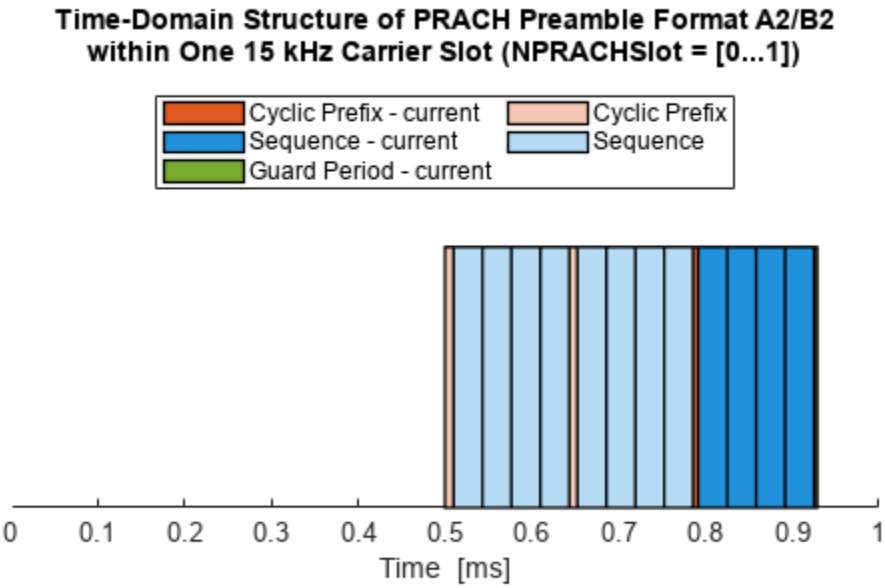
In the case of 30 kHz PRACH subcarrier spacing, only one of the two PRACH slots within a 15 kHz subcarrier spacing can be active. According to Table 6.3.3.2-2 in TS 38.211, either the first or the second PRACH slots can be active for PRACH preamble format B2. The value of `prach.ActivePRACHSlot` property defines which PRACH slot is active within the current carrier subframe. This property is the n_{slot}^{RA} parameter defined in TS 38.211 Section 5.3.2.

This case shows four combinations of the `NPRACHSlot` and `ActivePRACHSlot` property values and tests whether the PRACH preamble is active. This case displays the plot of the time-domain structure of the PRACH preamble for both combinations. The plot shows that the active PRACH preamble occupies the first half of the carrier slot when `ActivePRACHSlot` is `0` and occupies the second half of the carrier slot when `ActivePRACHSlot` is `1`. For more details on this plot, see the Plot Time-Domain Structure of Selected PRACH Preamble on page 2-98 section.

```
if ~any(strcmpi(format,{'0','1','2','3'})) && strcmpi(prach.FrequencyRange,'FR1') % Short preamb
    % Set subcarrier spacing to 30 kHz for this case
    prach.SubcarrierSpacing = 30;

    % Define all combinations of NPRACHSlot and ActivePRACHSlot to check
    nPRACHSlotCase2 = [0, 1, 2];
    activePRACHSlotCase2 = [0, 1];
    [NPRACHSlotCase2, ActivePRACHSlotCase2] = meshgrid(nPRACHSlotCase2,activePRACHSlotCase2);
    prachActivityTable = table(NPRACHSlotCase2(:),ActivePRACHSlotCase2(:),false*ones(numel(NPRACH
        'VariableNames',{'NPRACHSlot','ActivePRACHSlot','active'});

    % Loop over all combinations
    for i = 1:numel(NPRACHSlotCase2)
        prach.NPRACHSlot = NPRACHSlotCase2(i);
```

```
            prach.ActivePRACHSlot = ActivePRACHSlotCase2(i);
            prachSymbols = nrPRACH(carrier,prach);
            active = ~isempty(prachSymbols); % Check if the PRACH preamble is active in the current s
            prachActivityTable.active(i) = active;
            if active && prach.NPRACHSlot < 2
                % Plot the time-domain structure of the PRACH preamble for
                % active PRACH preambles in the first two slots
                hPRACHPreamblePlot(carrier,prach);
            end
        end
    else
        % Display a message for the filtered cases
        if any(strcmpi(format,{'0','1','2','3'}))
            disp(['PRACH long preamble format ' format ' is not compatible with 30 kHz subcarrier spa
        else % FR2
            disp('Frequency range FR2 is not compatible with 30 kHz subcarrier spacing.')
        end
    end
```





For short preamble formats with a 30 kHz subcarrier spacing, this table shows whether the PRACH preamble is active for each combination of the chosen values of the `NPRACHSlot` and `ActivePRACHSlot` properties.

```
if ~any(strcmpi(format,{'0','1','2','3'})) && strcmpi(prach.FrequencyRange,'FR1') % Short preamb
    disp(prachActivityTable)
end
```

NPRACHSlot	ActivePRACHSlot	active
0	0	1
0	1	0
1	0	0
1	1	1
2	0	1
2	1	0

Set the PRACH configuration object back to the user-defined configuration

```
prach.SubcarrierSpacing = subcarrierSpacing;
prach.ActivePRACHSlot = activePRACHSlot;
prach.NPRACHSlot = nPRACHSlot;
```

Inspect PRACH Configuration

The PRACH configuration object also has read-only properties that provide additional information about the current configuration:

- Preamble format: `Format`
- Maximum number of allowed PRACH time occasions: `NumTimeOccasions`
- Number of OFDM symbols in the PRACH slot grid corresponding to one transmission occasion: `PRACHDuration`
- Location of the first OFDM symbol of the current PRACH occasion: `SymbolLocation`
- Number of subframes spanned by a nominal PRACH slot: `SubframesPerPRACHSlot`
- Number of PRACH slots per overall period: `PRACHSlotsPerPeriod`

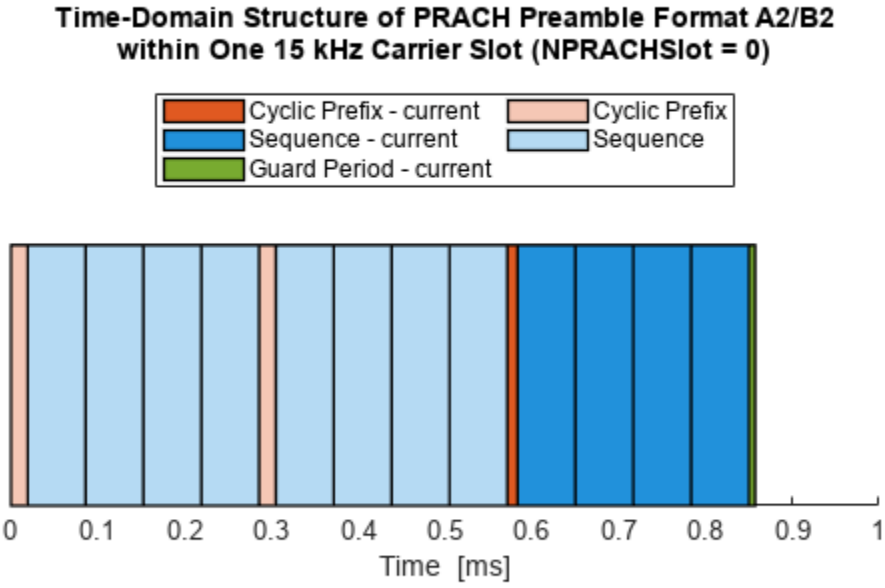
```
disp(prach)
```

```
  nrPRACHConfig with properties:

          FrequencyRange: 'FR1'
              DuplexMode: 'FDD'
       ConfigurationIndex: 146
        SubcarrierSpacing: 15
                      LRA: 139
            SequenceIndex: 0
            PreambleIndex: 0
            RestrictedSet: 'UnrestrictedSet'
      ZeroCorrelationZone: 0
                  RBOffset: 0
           FrequencyStart: 0
           FrequencyIndex: 0
                TimeIndex: 2
          ActivePRACHSlot: 0
               NPRACHSlot: 0

   Read-only properties:
                    Format: 'B2'
         NumTimeOccasions: 3
            PRACHDuration: 4
          SymbolLocation: 8
    SubframesPerPRACHSlot: 1
      PRACHSlotsPerPeriod: 10

   Constant properties:
                    Tables: [1x1 struct]
```

Plot Time-Domain Structure of Selected PRACH Preamble

This plot shows all the possible PRACH occasions (in the current carrier slot) in light colors and the current PRACH occasion (corresponding to the selected `TimeIndex`) in dark colors. This plot contains the cyclic prefix (CP), the PRACH active sequence periods, and a final guard period (GP) in red, blue, and green, respectively. If the PRACH preamble is not active in the current slot, the plot is empty. The plot shows time-related properties of the selected PRACH configuration and the PRACH position in the carrier slot. If the PRACH subcarrier spacing is smaller than the carrier subcarrier spacing, the plot shows the minimum number of carrier slots needed to transmit the PRACH preamble. The last PRACH occasion in time does not always correspond to the end of the carrier slot. The plot is empty for those time values in which no PRACH transmission is allowed for the current PRACH configuration.

```
hPRACHPreamblePlot(carrier,prach);
```



Generate and Map PRACH Symbols to Resource Grid

The PRACH resource grid shows the location of the PRACH preamble in both time and frequency domain. Using this resource grid, you can:

- Inspect the PRACH preamble visually in both time and frequency domain
- Generate the PRACH waveform, which is obtained by modulating the resource grid

The PRACH resource grid generation consists of these steps:

1. Generate an empty grid
2. Generate the symbols to be transmitted in the PRACH waveform
3. Generate the frequency indices and time indices in which the PRACH symbols are located
4. Map the PRACH symbols to the PRACH resource grid

Generate an empty PRACH resource grid.

```
prachGrid = nrPRACHGrid(carrier,prach);
size(prachGrid)
```

```
ans = 1×2

   624    14
```

Generate the PRACH symbols. The number of symbols depends on the PRACH configuration. The `prachSymbols` output is empty if the PRACH preamble is not active in the current slot.

```
prachSymbols = nrPRACH(carrier,prach);
```

Generate the PRACH indices. The value in each element of `prachIndices` is the linear index of the location of each element of `prachSymbols` in the PRACH resource grid.

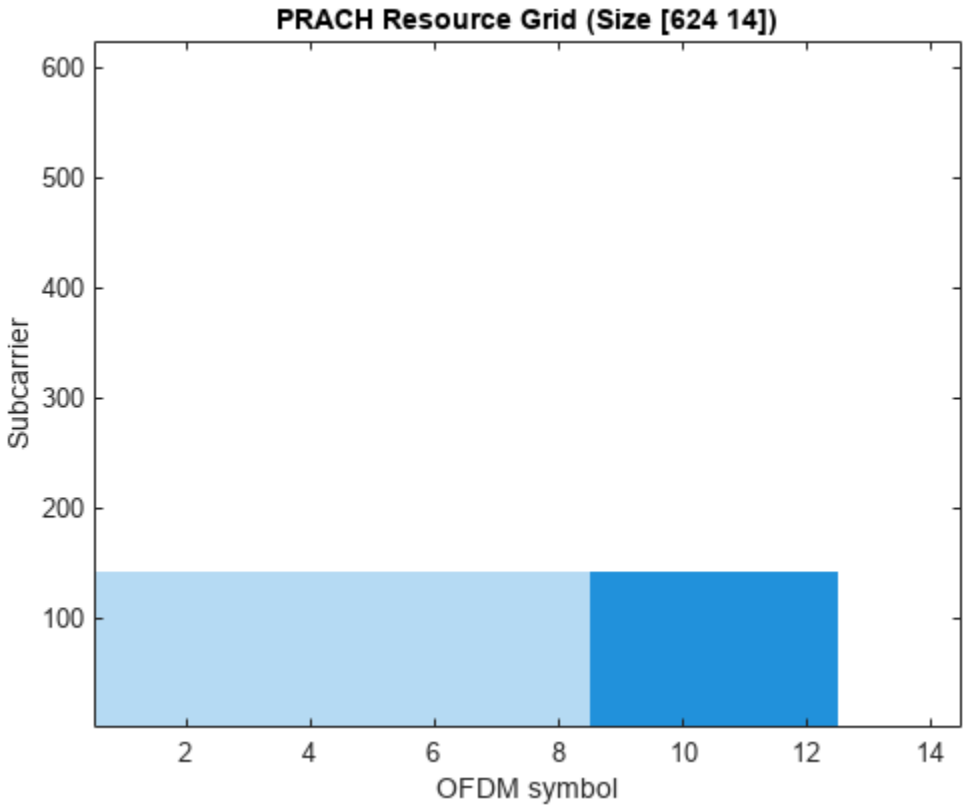
```
prachIndices = nrPRACHIndices(carrier,prach);
```

Map the PRACH symbols to the PRACH resource grid using the indices. To represent β_{PRACH} in TS 38.211 Section 6.3.3.2, the mapping applies a scaling factor of 1 to the PRACH symbols.

```
prachGrid(prachIndices) = 1 * prachSymbols;
```

The `hPRACHResourceGridPlot` helper function plots the PRACH resource grid to show the location of the active PRACH. The plot shows all the time occasions in which the PRACH can be transmitted. The plot shows all the possible PRACH occasions in the current carrier slot in light blue and the current PRACH occasion (corresponding to the selected `TimeIndex`) in dark blue. The plot is empty for OFDM symbols not used by any PRACH occasion for the current configuration. If the PRACH preamble is not active in the current slot, the plot is empty.

```
hPRACHResourceGridPlot(carrier,prach);
```



The PRACH resource grid contains 14 OFDM symbols except for these cases:

- For long preamble format 0, each preamble has one active sequence period that spans one subframe. Therefore, the slot grid related to format 0 has one OFDM symbol.

- For long preamble format 1, each preamble has two active sequence periods that span two subframes. Therefore, the slot grid related to format 1 has two OFDM symbols.
- For long preamble format 2, each preamble has four active sequence periods that span four subframes. Therefore, the slot grid related to format 2 has four OFDM symbols.
- For long preamble format 3, each preamble has four active sequence periods that span one subframe. Therefore, the slot grid related to format 3 has four OFDM symbols.
- For short preamble format C0, each preamble has one active sequence period. However, because of the guard and the cyclic prefix, the preamble spans two OFDM symbols. Therefore, the slot grid related to format C0 has seven OFDM symbols.

You can retrieve the number of active sequence periods from the value of the `PRACHDuration` property of the PRACH configuration object.

Generate Waveform for Single PRACH Preamble

Generate a time-domain waveform for a single PRACH preamble by modulating the PRACH resource grid. To set the number of time-domain samples over which to apply windowing and overlapping of OFDM symbols, use `windowing`. This example uses the default value for windowing. For more details about windowing, see `nrPRACHOFDMModulate`.

```
windowing = [];
[prachWaveform,prachInfo] = nrPRACHOFDMModulate(carrier,prach,prachGrid,'Windowing',windowing);
```

The output `prachWaveform` is a column vector corresponding to the time-domain waveform. The output `prachInfo` is a structure that provides dimensional information related to the PRACH. This example displays this information by using the `hPRACHInfoDisplay` helper function. The function displays the information related to the number of samples corresponding to CP, PRACH active sequence period T_{SEQ}, and GP for each OFDM symbol in a tabular format. The table lists all the OFDM symbols that fit in the resource grid. For short preamble formats, the values marked with * correspond to all possible PRACH occasions except the current one (marked light blue in the resource grid plot). For short preamble formats, the values within angle brackets represent OFDM symbols not used by any PRACH occasion for the current configuration (corresponding to an empty space in time in the resource grid plot).

Check the information related to the OFDM symbols against the `PRACHDuration`, `SymbolLocation`, and `NumTimeOccasions` properties. These properties show that:

- Each PRACH occasion lasts 4 OFDM symbols
- The current PRACH occasion starts at OFDM symbol 8
- 3 PRACH occasions are possible in time

```
hPRACHInfoDisplay(carrier,prach,windowing);
```

```
Information associated with PRACH:
   SubcarrierSpacing:         15 kHz
   Number of subcarriers:     624

Information associated with PRACH OFDM modulation:
   Nfft:                      1024
   Windowing:                 72
   Offset:                    0 samples

   Symbol    TCP     TSEQ       GP
   ------  ------   ------    -----
```

```
   0*     296*    1024*       0*
   1*       0*    1024*       0*
   2*       0*    1024*       0*
   3*       0*    1024*       0*
   4*     296*    1024*       0*
   5*       0*    1024*       0*
   6*       0*    1024*       0*
   7*       0*    1024*       0*
   8      180     1024        0
   9        0     1024        0
  10        0     1024        0
  11        0     1024      108
 <12>   <   0>   <1024>   <   0>
 <13>   <   0>   <1024>   <   0>

    *  : OFDM symbols for unused PRACH time occasions
   <#> : OFDM symbols not used by any PRACH time occasion
         for the current configuration

Total samples:             15360
Sample rate:               15.360 MHz
Duration:                  1.000 ms
Total number of subframes: 1
```

Summary and Further Exploration

This example shows how to configure the PRACH, set the configuration index based on the selected format, and determine whether a PRACH preamble is active in the current time resources. This example guides you through PRACH resource grid and time-domain waveform generation. Plotting the time-domain structure of the PRACH preamble displays all the available PRACH occasions for the selected configuration within one subframe. Plotting the resource grid displays all the available PRACH occasions for the selected configuration in both the time and frequency domain.

This example shows how to generate a waveform for a single PRACH preamble. For an example that generates a waveform for multiple PRACH preambles, see “5G NR PRACH Waveform Generation” on page 2-104.

Selected Bibliography

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

2 3GPP TS 38.331. "NR; Radio Resource Control (RRC); Protocol specification." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrPRACH` | `nrPRACHIndices` | `nrPRACHGrid`

Objects
`nrPRACHConfig`

Related Examples

- “5G NR PRACH Waveform Generation” on page 2-104
- “5G NR PRACH Detection and False Alarm Test” on page 2-108

5G NR PRACH Waveform Generation

This example implements a 5G NR PRACH waveform generator using 5G Toolbox™. The example shows how to parameterize and generate a 5G New Radio (NR) physical random access channel (PRACH) waveform, as defined in TS 38.211 [1]. The example demonstrates the parameterization and generation of one PRACH configuration in a single carrier, and displays the positions of the PRACH preambles in the resource grid. You can define the length of the waveform, in terms of subframes, and set the pattern of the active PRACH preambles in the generated waveform.

Waveform and Carrier Configuration

Configure one carrier and set the length of the generated waveform in terms of 1 ms subframes. Visualize the generated resource grid by setting the `DisplayGrids` field to 1.

Use the `waveconfig` structure to store configuration parameters needed for the PRACH waveform generation. The `waveconfig` structure contains these fields:

- `NumSubframes`: Number of 1 ms subframes in generated waveform.
- `DisplayGrids`: If set to 1, the example displays the resource grid.
- `Windowing`: Number of time-domain samples over which to apply windowing and overlapping of OFDM symbols. For more information, see `nrPRACHOFDMModulate`.
- `Carriers`: Carrier-specific configuration object, as described in `nrCarrierConfig`.
- `PRACH`: Structure containing the PRACH-related configuration, as described in detail in the PRACH Configuration section.

```
waveconfig = [];
waveconfig.NumSubframes = 10; % Number of 1 ms subframes in generated waveform
waveconfig.DisplayGrids = 1;  % Display the resource grid
waveconfig.Windowing = [];    % Use the default windowing

% Define a carrier configuration object
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.NSizeGrid = 52;

% Store the carrier into the waveconfig structure
waveconfig.Carriers = carrier;
```

PRACH Configuration

Set the parameters for the PRACH, taking into account that the numerology of the PRACH can be different from that of the carrier. This example sets the PRACH configuration corresponding to a PRACH short preamble format B2 with 15 kHz subcarrier spacing.

You can also set additional PRACH parameters. For more information, see `nrPRACHConfig`.

Add the field `PRACH` to the `waveconfig` structure to store the PRACH configuration and related parameters. The field `PRACH` is a structure containing these fields:

- `Config`: PRACH configuration object
- `AllocatedPreambles`: Index (0-based) of the allocated PRACH preambles to transmit. This field considers only the active PRACH preambles. Set this value to 'all' to include all the active PRACH preambles in the waveform.

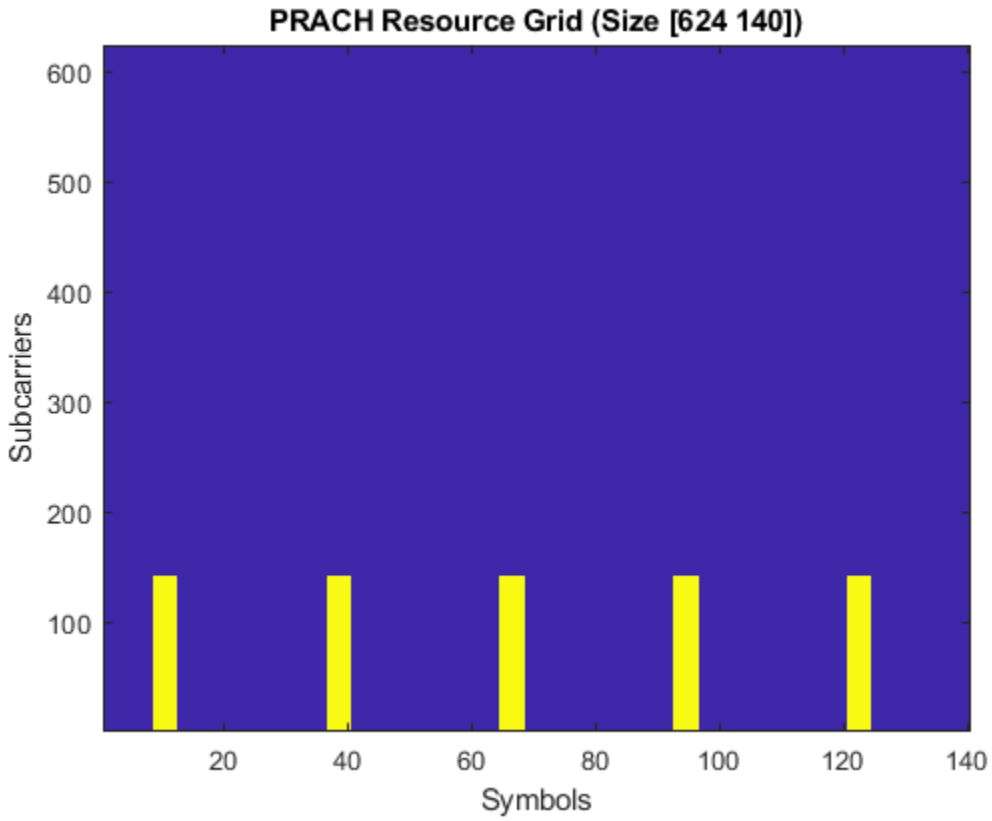
- `Power`: PRACH power scaling in dB. This parameter represents β_{PRACH} (in dB) in TS 38.211 Section 6.3.3.2.

```
% PRACH configuration
prach = nrPRACHConfig;
prach.FrequencyRange = 'FR1';   % Frequency range ('FR1', 'FR2')
prach.DuplexMode = 'FDD';       % Duplex mode ('FDD', 'TDD', 'SUL')
prach.ConfigurationIndex = 145; % Configuration index (0...255)
prach.SubcarrierSpacing = 15;   % Subcarrier spacing (1.25, 5, 15, 30, 60, 120)
prach.FrequencyIndex = 0;       % Index of the PRACH transmission occasions in frequency domain
prach.TimeIndex = 2;            % Index of the PRACH transmission occasions in time domain (0...
prach.ActivePRACHSlot = 0;      % Active PRACH slot number within a subframe or a 60 kHz slot (0

% Store the PRACH configuration and additional parameters in the
% waveconfig structure
waveconfig.PRACH.Config = prach;
waveconfig.PRACH.AllocatedPreambles = 'all'; % Index of the allocated PRACH preambles
waveconfig.PRACH.Power = 0;                  % PRACH power scaling in dB
```

Waveform Generation

Generate the PRACH complex baseband waveform by using the parameters stored in the `waveconfig` structure.

```
[waveform,gridset,winfo] = hNRPRACHWaveformGenerator(waveconfig);
```



When `waveconfig.DisplayGrids` is set to `1`, the waveform generator also plots the PRACH resource grid, in PRACH numerology. For more information on the number of OFDM symbols in the resource grid, see 5G NR PRACH Configuration.

The waveform generator function returns the time domain waveform, and two structures: `gridset` and `winfo`.

The structure `winfo` contains these fields:

- `NPRACHSlot`: PRACH slot numbers of each allocated PRACH preamble
- `PRACHSymbols`: PRACH symbols corresponding to each allocated PRACH slot
- `PRACHSymbolsInfo`: Additional information associated with PRACH symbols
- `PRACHIndices`: PRACH indices corresponding to each allocated PRACH slot
- `PRACHIndicesInfo`: Additional information associated with PRACH indices

The structure `gridset` contains these fields:

- `ResourceGrid`: Resource grid corresponding to this carrier
- `Info`: Structure with information corresponding to the PRACH OFDM modulation. If the PRACH is configured for FR2 or the PRACH slot for the current configuration spans more than one subframe, some of the OFDM-related information may be different between PRACH slots. In this case, the info structure is an array of the same length as the number of PRACH slots in the waveform.

```
if ~isempty(gridset.Info)
    disp('Information associated with PRACH OFDM modulation for the first PRACH slot:')
    disp(gridset.Info(1))
end
```

```
Information associated with PRACH OFDM modulation for the first PRACH slot:
                  Nfft: 1024
            SampleRate: 15360000
    CyclicPrefixLengths: [188 0 0 0 188 0 0 0 180 0 0 0 0 0]
           GuardLengths: [0 0 0 108 0 0 0 108 0 0 0 108 0 144]
          SymbolLengths: [1212 1024 1024 1132 1212 1024 1024 1132 1204 ... ]
           OffsetLength: 0
              Windowing: 72
```

Summary and Further Exploration

This example shows how to generate a time-domain waveform for a single PRACH configuration on a single carrier. You can set the length of the generated waveform in terms of number of subframes. You can also set the pattern of PRACH preambles in the generated waveform. The example also shows the OFDM-related information for the PRACH.

To generate a waveform containing multiple PRACH configurations in the same carrier, run this example for several PRACH configurations and add the generated waveforms together.

For more information about the PRACH configuration and the PRACH resource grid, see “5G NR PRACH Configuration” on page 2-91.

Selected Bibliography

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

See Also

Functions

`nrPRACH` | `nrPRACHIndices` | `nrPRACHGrid`

Objects

`nrPRACHConfig`

Related Examples

- “5G NR PRACH Configuration” on page 2-91
- “5G NR PRACH Detection and False Alarm Test” on page 2-108

5G NR PRACH Detection and False Alarm Test

This example implements the physical random access channel (PRACH) missed detection and false alarm conformance tests, as defined in TS 38.141-1. You can measure the probability of correct detection of the PRACH preamble in the presence of a preamble signal or switch the PRACH transmission off to measure the false alarm probability.

Introduction

The PRACH is an uplink transmission used by User Equipment (UE) to initiate synchronization with the gNodeB. TS 38.141-1 Section 8.4.1.5 defines the probability of PRACH detection to be greater than or equal to 99% at specific SNR values for a set of PRACH configurations and propagation conditions. There are several detection error cases:

- Detecting an incorrect preamble
- Not detecting a preamble
- Detecting the correct preamble but with the wrong timing estimation

TS 38.141-1 states that a correct detection is achieved when the estimation error of the timing offset of the strongest path is less than the time error tolerance given in Table 8.4.1.1-1. For channel propagation conditions TDLC300-100 and PRACH preamble format 0, the time error tolerance is 2.55 microseconds.

In this example, a PRACH waveform is configured and passed through an appropriate channel. At the receiver side, the example performs PRACH detection and calculates the PRACH detection probability. The example considers the parameters defined in TS 38.141-1 Table 8.4.1.5-1 and Table A.6-1. These are: normal mode (i.e., unrestricted set), 2 receive antennas, TDLC300-100 channel, normal cyclic prefix, burst format 0, SNR -6.0 dB. If you change the PRACH configuration to use one of the other PRACH preamble formats listed in Table A.6-1, you need to update the values of the time error tolerance and the SNR, according to TS 38.141-1 Table 8.4.1.1-1 and Tables 8.4.1.5-1 to 8.4.1.5-3, respectively.

Simulation Configuration

The example considers 10 PRACH slots at a number of SNRs. You should use a large number of `numPRACHSlots` to produce meaningful results. You can set `SNRdB` as an array of values or a scalar. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86. Table 8.4.1.5-1 in TS 38.141-1 specifies the frequency offset `foffset` that is modeled between the transmitter and receiver. The `timeErrorTolerance` variable specifies the time error tolerance, as defined in Table 8.4.1.1-1 in TS 38.141-1. You can set the detection threshold to a value in the range [0,1] or to empty to use the default value in the `nrPRACHDetect` function. To simulate a false alarm test, disable the PRACH transmission by setting `prachEnabled` to `false` instead.

```
numPRACHSlots = 10;              % Number of PRACH slots to simulate at each SNR
SNRdB = [-21, -16, -11, -6, -1]; % SNR range in dB
foffset = 400.0;                 % Frequency offset in Hz
timeErrorTolerance = 2.55;       % Time error tolerance in microseconds
threshold = [];                  % Detection threshold
prachEnabled = true;             % Enable PRACH transmission. To simulate false alarm test, disab
```

Carrier Configuration

Use the `nrCarrierConfig` configuration object `carrier` to specify the carrier settings. The example considers a carrier characterized by a subcarrier spacing of 15 kHz and a bandwidth of 5 MHz. That is, the carrier spans 25 resource blocks, according to Table 5.3.2-1 in TS 38.104.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.NSizeGrid = 25;

% Compute the OFDM-related information
ofdmInfo = nrOFDMInfo(carrier);
```

PRACH Configuration

Table A.6-1 in TS 38.141-1 specifies the PRACH configurations to use for the PRACH detection conformance test.

Set the PRACH configuration by using the `nrPRACHConfig` configuration object `prach`, according to Table A.6-1 and Section 8.4.1.4.2 in TS 38.141-1.

```
% Set PRACH configuration
prach = nrPRACHConfig;
prach.FrequencyRange = 'FR1';                   % Frequency range
prach.DuplexMode = 'FDD';                       % Frequency Division Duplexing (FDD)
prach.ConfigurationIndex = 27;                  % Configuration index for format 0
prach.SubcarrierSpacing = 1.25;                 % Subcarrier spacing
prach.SequenceIndex = 22;                       % Logical sequence index
prach.PreambleIndex = 32;                       % Preamble index
prach.RestrictedSet = 'UnrestrictedSet';        % Normal mode
prach.FrequencyStart = 0;                       % Frequency location

% Define the value of ZeroCorrelationZone using the NCS table stored in
% the nrPRACHConfig object
switch prach.Format
    case {'0','1','2'}
        ncsTable = nrPRACHConfig.Tables.NCSFormat012;
        ncsTableCol = (string(ncsTable.Properties.VariableNames) == prach.RestrictedSet);
    case '3'
        ncsTable = nrPRACHConfig.Tables.NCSFormat3;
        ncsTableCol = (string(ncsTable.Properties.VariableNames) == prach.RestrictedSet);
    otherwise
        ncsTable = nrPRACHConfig.Tables.NCSFormatABC;
        ncsTableCol = contains(string(ncsTable.Properties.VariableNames), num2str(prach.LRA));
end
NCS = 13;
zeroCorrelationZone = ncsTable.ZeroCorrelationZone(ncsTable{:,ncsTableCol}==NCS);
prach.ZeroCorrelationZone = zeroCorrelationZone; % Cyclic shift index
```

Propagation Channel Configuration

Use the `nrTDLChannel` object to configure the tapped delay line (TDL) propagation channel model `channel` as described in TS 38.141-1 Table 8.4.1.1-1.

```
channel = nrTDLChannel;
channel.DelayProfile = "TDLC300";           % Delay profile
channel.MaximumDopplerShift = 100.0;        % Maximum Doppler shift in Hz
channel.SampleRate = ofdmInfo.SampleRate;   % Input signal sample rate in Hz
```

```
channel.MIMOCorrelation = "Low";             % MIMO correlation
channel.TransmissionDirection = "Uplink";    % Uplink transmission
channel.NumReceiveAntennas = 2;              % Number of receive antennas
channel.NormalizePathGains = true;           % Normalize delay profile power
channel.Seed = 42;                           % Channel seed. Change this for different channel rea
channel.NormalizeChannelOutputs = true;      % Normalize for receive antennas

% Get the channel characteristic information
channelInfo = info(channel);
```

Loop for SNR Values

Use a loop to run the simulation for the set of SNR points given by the vector `SNRdB`. The SNR vector configured here is a range of SNR points including a point at -6.0 dB, the SNR at which the test requirement for PRACH detection rate (99%) is to be achieved for preamble format 0, as discussed in Table 8.4.1.5-1 in TS 38.141-1.

`hNRPRACHWaveformGenerator` generates an output signal normalized to the same transmit power as for an uplink data transmission within the 5G Toolbox™. Therefore, the same normalization must take place on the noise added to the PRACH. The noise added before OFDM demodulation will be amplified by the IFFT by a factor equal to the square root of the size of the IFFT (N_{FFT}). To ensure that the power of the noise added is normalized after demodulation, and thus to achieve the desired SNR, the desired noise power is divided by N_{FFT}. In addition, as real and imaginary parts of the noise are created separately before being combined into complex additive white Gaussian noise, the noise amplitude is scaled by $1/\sqrt{2}$ so the generated noise power is 1.

At each SNR test point, calculate the detection probability on a slot by slot basis using these steps:

- *PRACH Transmission:* Use `hNRPRACHWaveformGenerator` to generate a PRACH waveform. Send the PRACH preambles with the timing offsets defined in TS 38.141-1 Figure 8.4.1.4.2-2. Set a timing offset base value to 50% of the number of cyclic shifts for PRACH generation. This offset is increased for each preamble, adding a step value of 0.1 microseconds, until the end of the tested range, which is 0.9 microseconds for PRACH preamble format 0. This pattern then repeats.
- *Noisy Channel Modeling:* Pass the waveform through a TDL channel and add additive white Gaussian noise. Add additional samples to the end of the waveform to cover the range of delays expected from the channel modeling (a combination of implementation delay and channel delay spread). This implementation delay is then removed to ensure the implementation delay is not interpreted as an actual timing offset in the preamble detector.
- *Application of Frequency Offset:* Apply the frequency offset to the received waveform as defined by the specification.
- *PRACH Detection:* Perform PRACH detection using the `nrPRACHDetect` function for all cell preamble indices (0:63). Use the detected PRACH index and offset returned by `nrPRACHDetect` to determine where a detection was successful according to the constraints discussed in the Introduction section.

```
% Initialize variables storing detection probability at each SNR
pDetection = zeros(size(SNRdB));

% Store the configuration parameters needed to generate the PRACH waveform
waveconfig.NumSubframes = prach.SubframesPerPRACHSlot;
waveconfig.Windowing = [];
waveconfig.Carriers = carrier;
waveconfig.PRACH.Config = prach;
waveconfig.PRACH.Enable = prachEnabled;
```

```
% The temporary variables 'prach_init', 'waveconfig_init', 'ofdmInfo_init',
% and 'channelInfo_init' are used to create the temporary variables
% 'prach', 'waveconfig', 'ofdmInfo', and 'channelInfo' within the SNR loop
% to create independent instances in case of parallel simulation
prach_init = prach;
waveconfig_init = waveconfig;
ofdmInfo_init = ofdmInfo;
channelInfo_init = channelInfo;

for snrIdx = 1:numel(SNRdB) % comment out for parallel computing
% parfor snrIdx = 1:numel(SNRdB) % uncomment for parallel computing
% To reduce the total simulation time, you can execute this loop in
% parallel by using the Parallel Computing Toolbox. Comment out the 'for'
% statement and uncomment the 'parfor' statement. If the Parallel Computing
% Toolbox(TM) is not installed, 'parfor' defaults to normal 'for' statement

    % Display progress in the command window
    timeNow = char(datetime('now','Format','HH:mm:ss'));
    fprintf([timeNow ': Simulating SNR = %+5.1f dB...'], SNRdB(snrIdx));

    % Set the random number generator settings to default values
    rng('default');

    % Initialize variables for this SNR point, required for initialization
    % of variables when using the Parallel Computing Toolbox
    prach = prach_init;
    waveconfig = waveconfig_init;
    ofdmInfo = ofdmInfo_init;
    channelInfo = channelInfo_init;

    % Reset the channel so that each SNR point will experience the same
    % channel realization
    reset(channel);

    % Normalize noise power to account for the sampling rate, which is a
    % function of the IFFT size used in OFDM modulation. The SNR is defined
    % per carrier resource element for each receive antenna.
    SNR = 10^(SNRdB(snrIdx)/10);
    N0 = 1/sqrt(2.0*channel.NumReceiveAntennas*double(ofdmInfo.Nfft)*SNR);

    % Detected preamble count
    detectedCount = 0;

    % Loop for each PRACH slot
    numActivePRACHSlots = 0;
    for nSlot = 0:numPRACHSlots-1

        % Generate PRACH waveform for the current slot
        prach.NPRACHSlot = nSlot;
        waveconfig.PRACH.Config.NPRACHSlot = nSlot;
        [waveform,~,winfo] = hNRPRACHWaveformGenerator(waveconfig);

        % Set PRACH timing offset in microseconds as per TS 38.141-1 Figure
        % 8.4.1.4.2-2 and Figure 8.4.1.4.2-3
        if prach.LRA==839 % Long preamble, values as in Figure 8.4.1.4.2-2
            baseOffset = ((winfo.WaveformResources.PRACH.Resources.PRACHSymbolsInfo.NumCyclicShi
            timingOffset = baseOffset + mod(nSlot,10)/10; % (microseconds)
```

```
else % Short preamble, values as in Figure 8.4.1.4.2-3
    baseOffset = 0; % (microseconds)
    timingOffset = baseOffset + mod(nSlot,9)/10; % (microseconds)
end
sampleDelay = fix(timingOffset / 1e6 * ofdmInfo.SampleRate);

% Generate transmit waveform
txwave = [zeros(sampleDelay,1); waveform];

% Pass data through channel model. Append zeros at the end of the
% transmitted waveform to flush channel content. These zeros take
% into account any delay introduced in the channel. This is a mix
% of multipath delay and implementation delay. This value may
% change depending on the sampling rate, delay profile and delay
% spread
rxwave = channel([txwave; zeros(channelInfo.MaximumChannelDelay, size(txwave,2))]);

% Add noise
noise = N0*complex(randn(size(rxwave)), randn(size(rxwave)));
rxwave = rxwave + noise;

% Skip this slot if the PRACH is inactive.
% Skip the detection of this slot after advancing the channel to
% make sure that the channel is always synchronized with the
% current slot.
% If the PRACH is inactive in this slot, the receiver should not
% expect any PRACH transmission and thus should not even try to
% detect a PRACH. Skipping the detection of an inactive slot is
% particularly important when performing a conformance test. If the
% PRACH is inactive, the reference waveform computed internally in
% the |nrPRACHDetect| function is empty. This leads to an empty
% correlation and thus to an empty detected preamble. This empty
% preamble leads to an incorrect value of the detection
% probability.
if isempty(winfo.WaveformResources.PRACH.Resources.PRACHSymbols)
    continue;
end
numActivePRACHSlots = numActivePRACHSlots + 1;

% Remove the implementation delay of the channel filter
rxwave = rxwave((channelInfo.ChannelFilterDelay + 1):end, :);

% Apply frequency offset
t = ((0:size(rxwave, 1)-1)/channel.SampleRate).';
rxwave = rxwave .* repmat(exp(1i*2*pi*foffset*t), 1, size(rxwave, 2));

% PRACH detection for all cell preamble indices
[detected, offsets] = nrPRACHDetect(carrier, prach, rxwave, 'DetectionThreshold', thresho

% Test for preamble detection
if (length(detected)==1)
    if ~prachEnabled
        % For the false alarm test, any preamble detected is wrong
        detectedCount = detectedCount + 1;
    else
        % Test for correct preamble detection
        if (detected==prach.PreambleIndex)
```

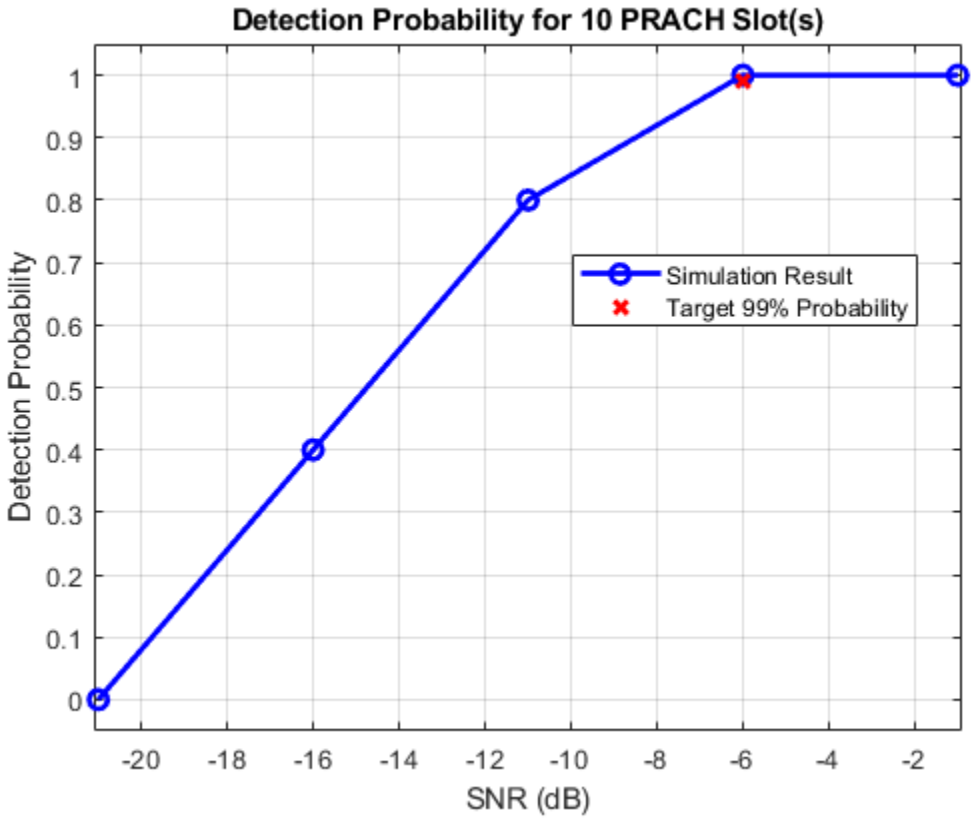
```
                    % Calculate timing estimation error
                    trueOffset = timingOffset/1e6; % (s)
                    measuredOffset = offsets(1)/channel.SampleRate;
                    timingerror = abs(measuredOffset-trueOffset);

                    % Test for acceptable timing error
                    if (timingerror<=timeErrorTolerance/1e6)
                        detectedCount = detectedCount + 1; % Detected preamble
                    end
                end
            end
        end

    end % of nSlot loop

    % Compute final detection probability for this SNR
    pDetection(snrIdx) = detectedCount/numActivePRACHSlots;

    % Display the detection probability for this SNR
    fprintf('Detection probability: %d%%\n', pDetection(snrIdx)*100);

end % of SNR loop
```

```
23:16:32: Simulating SNR = -21.0 dB...Detection probability: 0%
23:16:34: Simulating SNR = -16.0 dB...Detection probability: 40%
23:16:34: Simulating SNR = -11.0 dB...Detection probability: 80%
23:16:35: Simulating SNR =  -6.0 dB...Detection probability: 100%
23:16:36: Simulating SNR =  -1.0 dB...Detection probability: 100%
```

Results

At the end of the SNR loop, the example plots the calculated detection probabilities for each SNR value against the target probability.

```
% Plot detection probability
figure('Name','Detection Probability');
plot(SNRdB,pDetection,'b-o','LineWidth',2,'MarkerSize',7);
title(['Detection Probability for ', num2str(numPRACHSlots) ' PRACH Slot(s)'] );
xlabel('SNR (dB)'); ylabel('Detection Probability');
grid on; hold on;
% Plot target probability
if prachEnabled
    % For a missed detection test, detection probability should be >= 99%
    pTarget = 99;
else
    % For a false alarm test, detection probability should be < 0.1%
    pTarget = 0.1; %#ok<UNRCH>
end
plot(-6.0,pTarget/100,'rx','LineWidth',2,'MarkerSize',7);
legend('Simulation Result', ['Target ' num2str(pTarget) '% Probability'],'Location','best');
minP = 0;
if(~isnan(min(pDetection)))
    minP = min([pDetection(:); pTarget]);
end
axis([SNRdB(1)-0.1 SNRdB(end)+0.1 minP-0.05 1.05])
```



References

1 3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing. Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

2 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions

`nrPRACH` | `nrPRACHIndices` | `nrPRACHGrid` | `nrPRACHDetect`

Objects

`nrPRACHConfig`

Related Examples

- “5G NR PRACH Configuration” on page 2-91
- “5G NR PRACH Waveform Generation” on page 2-104
- “SNR Definition Used in Link Simulations” on page 5-86

NR UCI Multiplexing on PUSCH

This example shows the different processing steps involved in the data and control multiplexing to form a codeword associated with a physical uplink shared channel (PUSCH) using 5G Toolbox™ features.

Introduction

Uplink control information (UCI) messages consist of a hybrid automatic repeat request acknowledgment (HARQ-ACK), channel state information (CSI), and a scheduling request (SR). These UCI messages are encoded and transmitted through the physical uplink control channel (PUCCH) or are multiplexed on the PUSCH. The CSI reporting configuration can be aperiodic (using a PUSCH), periodic (using a PUCCH), or semi-persistent (using a PUCCH or DCI-activated PUSCH). A CSI report comprises of two parts. CSI part 1 has a fixed payload size and is used to identify the number of information bits in CSI part 2. CSI part 1 must be transmitted completely before the transmission of CSI part 2. The HARQ-ACK (if any) and CSI (if any) is encoded and multiplexed with or without encoded UL-SCH data, and then transmitted on a PUSCH.

Data and Control Multiplexing

The encoded data, encoded HARQ-ACK, encoded CSI part 1, and encoded CSI part 2 are multiplexed to form a codeword with the steps outlined in TS 38.212 Section 6.2.7 [1] on page 2-127.

The UCI information is transmitted in only the OFDM symbols that are unused for demodulation reference signal (DM-RS) transmission. In any OFDM symbol used for UCI transmission for a UCI type, the mapping of that UCI type depends on the number of resource elements (REs) available for UCI transmission and the remaining REs required for that UCI type. If the number of remaining REs required for that UCI type in an OFDM symbol is greater than half of the available REs for the UCI transmission, the mapping of the UCI type is contiguous. Otherwise, the mapping is uniformly distributed across available REs in an OFDM symbol to achieve the diversity gain. The number of coded bits that are occupied in an RE for UCI or data transmission, is equal to the product of the modulation order and the number of layers.

The coded HARQ-ACK bits are placed from the OFDM symbol, after the first consecutive DM-RS OFDM symbols. The coded CSI part 1 or part 2 bits are placed at the starting OFDM symbol that is unused for DM-RS in the shared channel symbol allocation. The multiplexing operation depends on the number of HARQ-ACK bits. When the number of HARQ-ACK bits is less than or equal to 2, the coded HARQ-ACK bits are punctured. Otherwise, the coded HARQ-ACK bits are rate-matched.

Multiplexing involves these processing steps.

- Step 1: When the number of HARQ-ACK bits is less than or equal to 2, find the reserved HARQ-ACK locations.
- Step 2: When the number of HARQ-ACK bits is greater than 2, map the coded HARQ-ACK bits (if any).
- Step 3: Map the coded CSI part 1 and CSI part 2 bits (if any).
- Step 4: Map the coded UL-SCH bits (if any).
- Step 5: When the number of HARQ-ACK bits is less than or equal to 2, map the coded HARQ-ACK bits (if any).
- Step 6: Form the codeword.

This example shows steps 1 to 6 for two separate cases for the PUSCH, with one resource block occupying all of the OFDM symbols in a slot, a single layer, a pi/2-BPSK modulation scheme, and DM-RS symbols occupying OFDM symbols 2, 7, and 11 (0-based). The REs other than DM-RS REs in the DM-RS OFDM symbols are used for data transmission. For the first case, the number of HARQ-ACK bits is less than or equal to 2. For the second case, the number of HARQ-ACK bits is greater than 2. This figure shows the grid with only the DM-RS symbol locations. This grid is populated with the coded types UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2, according to the multiplexing operation for each case. The transport block size is set to 24, and the target code rate is set to 314/1024. The number of payload bits of each CSI part is set to 10, and all of the associated beta factors are set to 1.

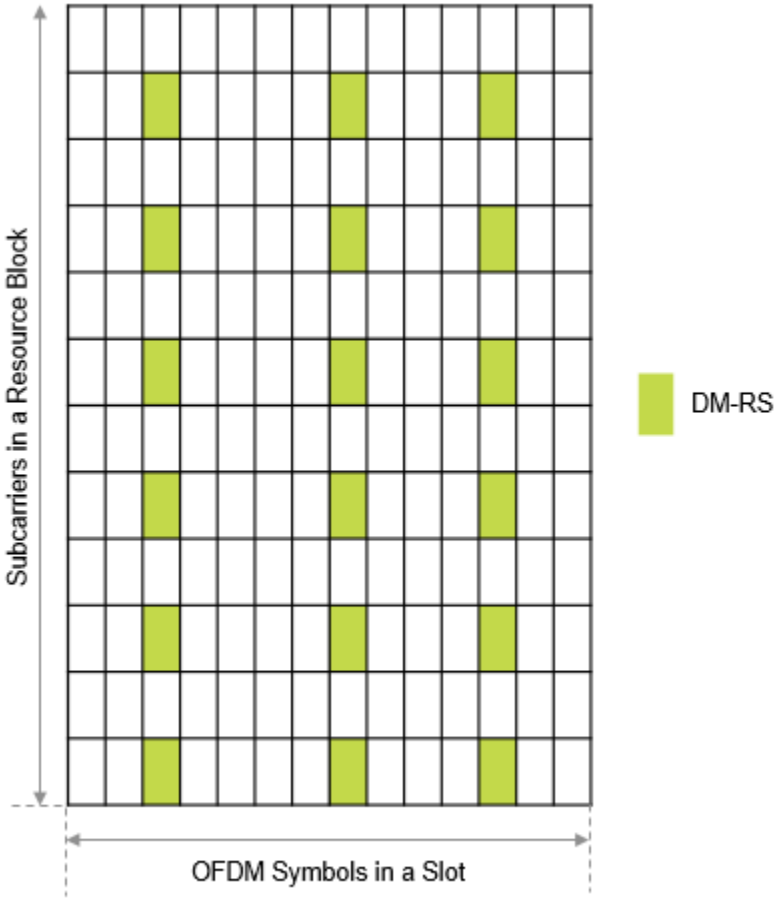


Figure 1: DM-RS Locations

Configure Carrier Resource Grid and PUSCH

Configure an `nrCarrierConfig` object to create a 15 kHz carrier resource grid with one resource block. Configure an `nrPUSCHConfig` object to get the respective DM-RS locations as shown in the previous figure. Also, configure the UCI parameters (`BetaOffsetACK`, `BetaOffsetCSI1`, `BetaOffsetCSI2`, and `UCIScaling`). The property `BetaOffsetACK` determines the number of resources for multiplexing HARQ-ACK in a PUSCH. The properties `BetaOffsetCSI1` and `BetaOffsetCSI2` determine the number of resources for multiplexing CSI reports in a PUSCH. The property `UCIScaling` limits the number of REs assigned to the UCI on PUSCH.

```
% Set the carrier configuration
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;   % Subcarrier spacing in kHz (15, 30, 60, 120, 240)
carrier.CyclicPrefix = 'normal';  % Cyclic prefix ('normal' or 'extended')
carrier.NSizeGrid = 1;            % Number of resource blocks in carrier resource grid (1...275)

% Set the PUSCH configuration
pusch = nrPUSCHConfig;
pusch.Modulation = 'pi/2-BPSK';     % Modulation scheme ('pi/2-BPSK', 'QPSK', '16QAM', '64QAM', c
pusch.NumLayers = 1;                % Number of layers (1, 2, 3, or 4)
pusch.SymbolAllocation = [0 14];    % OFDM symbol allocation [S L]
pusch.PRBSet = 0;                   % Vector of PRB allocated with values in the range [0, 274]
pusch.MappingType = 'A';            % Mapping type ('A' or 'B')
pusch.FrequencyHopping = 'neither'; % Frequency hopping configuration ('neither', 'interSlot', or
pusch.SecondHopStartPRB = 0;        % Second hop start PRB in the range [0, 274], ensure the comb
pusch.TransformPrecoding = 0;       % Transform precoding (0 or 1)

% Set the UCI parameters
pusch.BetaOffsetACK = 1;  % Beta offset for HARQ-ACK
pusch.BetaOffsetCSI1 = 1; % Beta offset for CSI part 1
pusch.BetaOffsetCSI2 = 1; % Beta offset for CSI part 2
pusch.UCIScaling = 1;     % UCI scaling factor

% Set the DM-RS parameters
pusch.DMRS.DMRSConfigurationType = 1;   % DM-RS configuration type (1 or 2)
pusch.DMRS.DMRSTypeAPosition = 2;       % DM-RS type A position (2 or 3)
pusch.DMRS.DMRSLength = 1;              % DM-RS length (1 or 2)
pusch.DMRS.DMRSAdditionalPosition = 2;  % Number of DM-RS additional positions (0, 1, 2, or 3)
pusch.DMRS.NumCDMGroupsWithoutData = 1; % Number of CDM groups without data (1, 2, or 3)

% Set the PT-RS parameters
pusch.EnablePTRS = 0;            % Disable or Enable PT-RS (0 or 1)
pusch.PTRS.FrequencyDensity = 2; % PT-RS frequency density (2 or 4)
pusch.PTRS.TimeDensity = 1;      % PT-RS time density (1, 2, or 4)
pusch.PTRS.REOffset = '00';      % PT-RS resource element offset ('00', '01', '10', or '11')
pusch.PTRS.NumPTRSSamples = 2;   % Number of PT-RS samples (2 or 4)
pusch.PTRS.NumPTRSGroups = 2;    % Number of PT-RS groups (2, 4, or 8)

% Set the target code rate, transport block size, payload lengths of CSI part 1 and CSI part 2
tcr = 314/1024; % Target code rate in the range (0, 1)
tbs = 24;       % Transport block size
ocsi1 = 10;     % Number of CSI part 1 bits
ocsi2 = 10;     % Number of CSI part 2 bits
```

Case 1: Number of HARQ-ACK Bits Less Than or Equal to 2

For illustration purposes, the number of HARQ-ACK bits is set to 1 in this example. To obtain the information about the number of coded bits of each type, use the `nrULSCHInfo` function. For the specified PUSCH configuration, target code rate, and payload values, the number of coded HARQ-ACK bits is 2, the number of coded CSI part 1 and CSI part 2 bits is 19 each, and the number of coded UL-SCH bits is 94.

Step 1: When the number of HARQ-ACK bits is less than or equal to 2, find the **reserved HARQ-ACK** locations and mark them on the grid.

The number of reserved HARQ-ACK locations is achieved by calculating the rate-matching length of the HARQ-ACK with number of HARQ-ACK bits set to 2. With the specified PUSCH configuration,

target code rate, and transport block size, the number of reserved HARQ-ACK locations is 4. The HARQ-ACK is mapped to the REs in the OFDM symbol that is available after the first consecutive DM-RS OFDM symbols. This figure shows the locations of reserved HARQ-ACK.

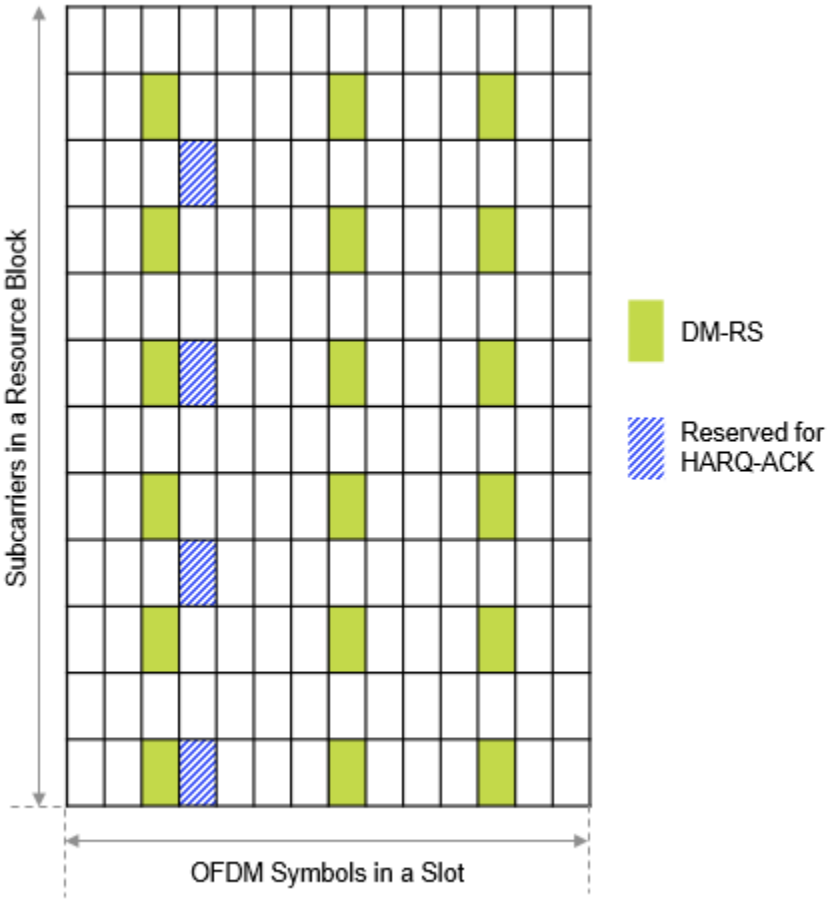


Figure 2: Reserved HARQ-ACK Locations

Step 2: When the number of HARQ-ACK bits is greater than 2, map the coded **HARQ-ACK** bits.

Because the number of HARQ-ACK bits is less than or equal to 2 in this case, skip this step.

Step 3: Map the coded **CSI part 1** and **CSI part 2** bits

The CSI mapping (CSI part 1 followed by CSI part 2) starts from the first non-DMRS OFDM symbol available in the PUSCH allocation. For the configured setup, the mapping starts from OFDM symbol 0 (0-based). The mapping locations are determined based on the number of REs available and the number of REs required for CSI part 1 transmission. CSI part 1 transmission required 19 REs. Because only 12 REs are available for transmission, every RE is occupied in this case for CSI part 1 in OFDM symbol 0. For the transmission of the remaining CSI part 1 symbols, the mapping goes to the next OFDM symbol not used for DM-RS (that is, OFDM symbol 1). In OFDM symbol 1, 12 REs are available, but CSI part 1 requires only 7 REs. Because the remaining REs required for CSI part 1 are more than half of the REs available for UCI transmission, CSI part 1 is mapped to contiguous REs. This figure shows this scenario.

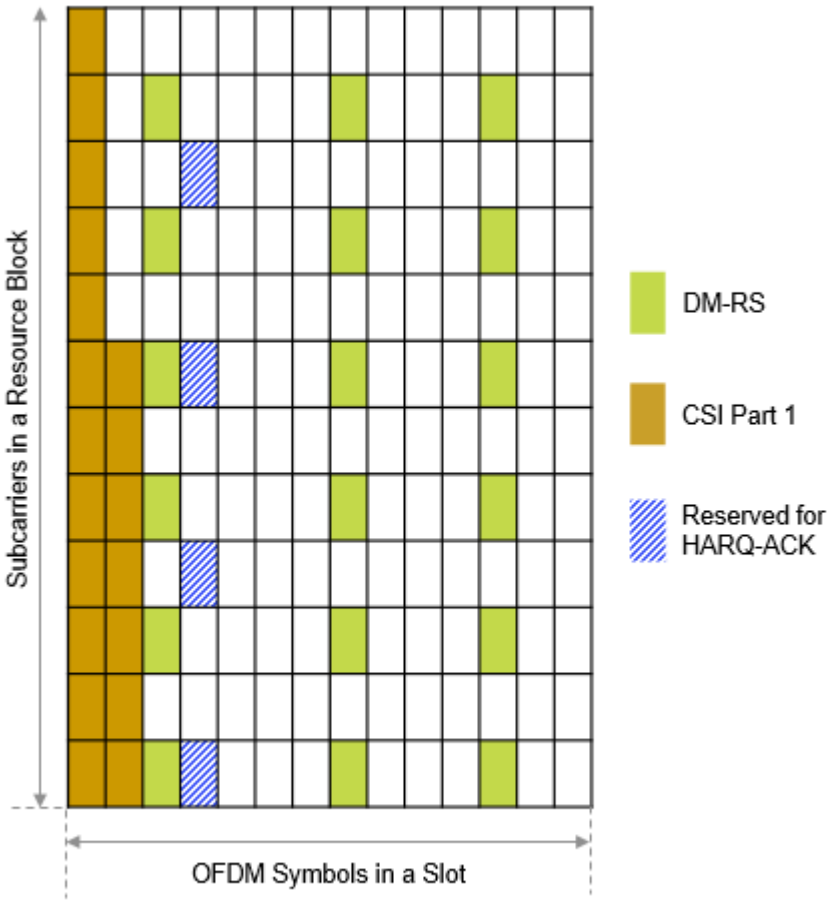


Figure 3: CSI Part 1 Locations

Once the coded CSI part 1 is completely mapped, the mapping of coded CSI part 2 starts. The mapping starts from the first non-DMRS OFDM symbol that is used for PUSCH transmission (that is, OFDM symbol 0). In OFDM symbol 0, no REs are available for UCI transmission, this OFDM symbol is skipped. As a result, mapping moves to the next OFDM symbol (that is, OFDM symbol 1). In OFDM symbol 1, 5 REs are available for UCI transmission, and 19 REs are required for CSI part 2 transmission. This scenario leads to mapping 5 REs with CSI part 2 and then continuing with the same mapping rule as in CSI part 1 for the next OFDM symbols other than DM-RS. Except the coded CSI part 1, the coded CSI part 2 and UL-SCH data can map to the reserved HARQ-ACK locations. This figure shows the mapping of CSI part 2. The reserved HARQ-ACK locations are covered by CSI part 2 in OFDM symbol 3. Also, in OFDM symbol 4, CSI part 2 is distributed because the number of REs required for the remaining CSI part 2 transmission is less than half of the REs available for the UCI transmission.

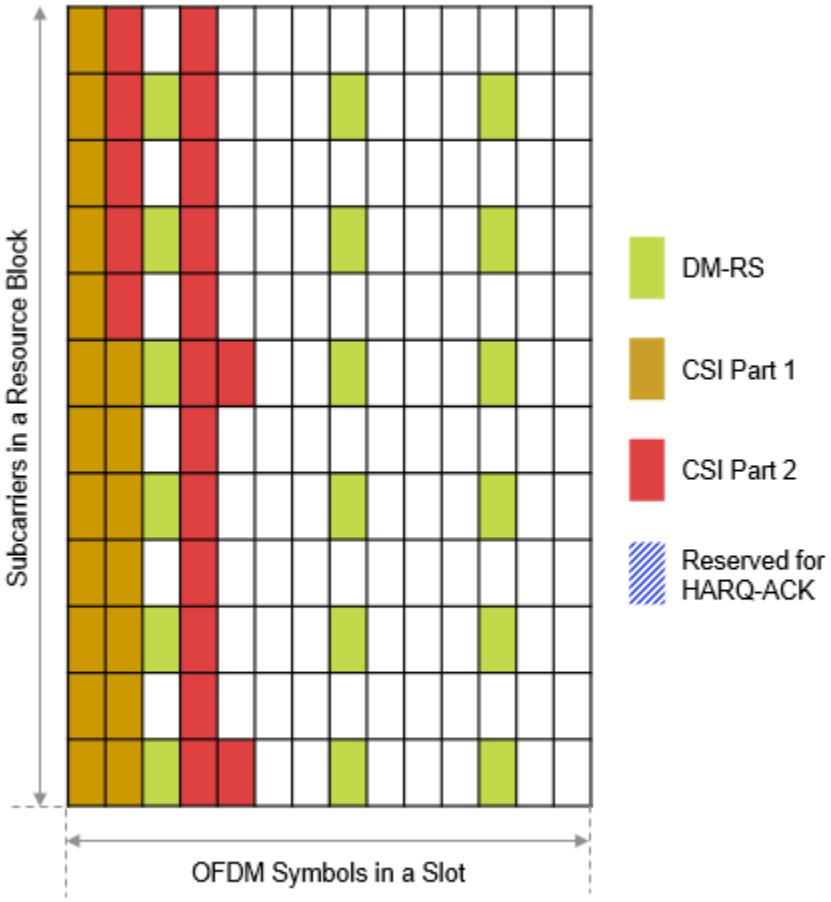


Figure 4: CSI Part 2 Locations

Step 4: Map the coded **UL-SCH** data bits.

This figure shows how the UL-SCH data is mapped to the remaining locations in the grid.

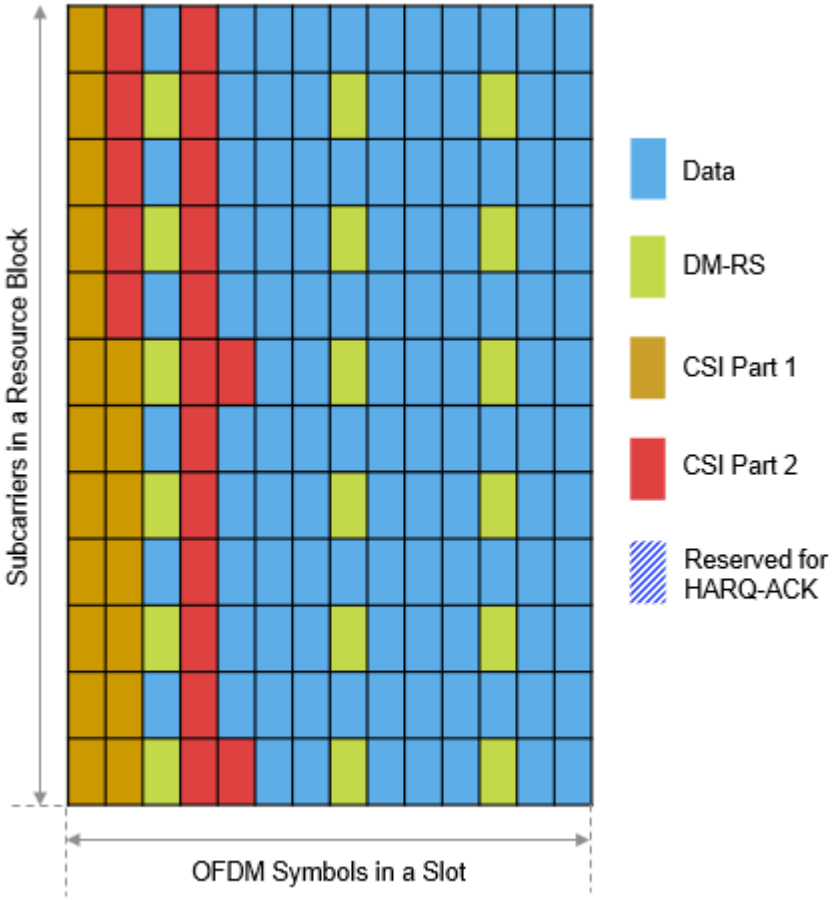


Figure 5: UL-SCH Data Locations

Step 5: When the number of HARQ-ACK bits is less than or equal to 2, map the coded **HARQ-ACK** bits.

The HARQ-ACK bits are mapped within the reserved HARQ-ACK locations in a distributed pattern. This figure indicates the HARQ-ACK mapping to the grid. The HARQ-ACK punctures the CSI part 2 that occupied the reserved HARQ-ACK locations.

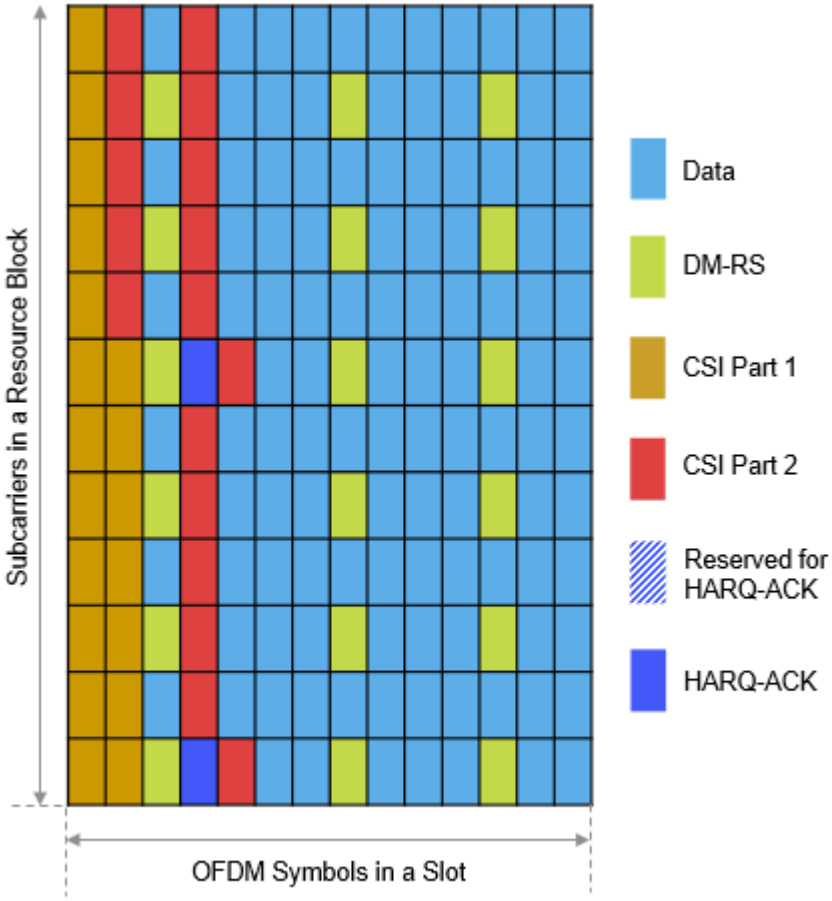


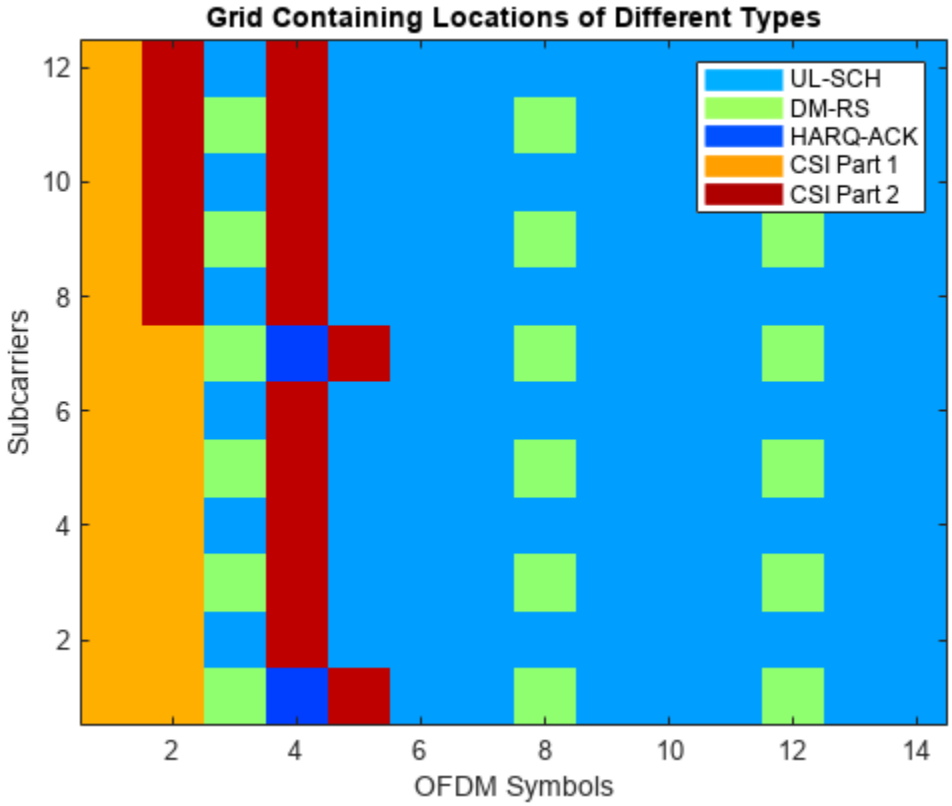
Figure 6: HARQ-ACK Locations

Step 6: Form the **codeword** by reading the bits frequency first and time next approach at each RE other than DM-RS REs.

Use the `nrULSCHMultiplex` function to get the codeword, performing the steps 1 to 6.

```
% Set the payload size of the HARQ-ACK to a value less than or equal to 2
oack = 1; % Number of HARQ-ACK bits

% Get the UL-SCH coding information
info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);

% Set random coded UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2 bits
culsch = randi([0 1],info.GULSCH,1);
cack = randi([0 1],info.GACK,1);
ccsi1 = randi([0 1],info.GCSI1,1);
ccsi2 = randi([0 1],info.GCSI2,1);

% Get the codeword and locations of each type (data and UCI)
[cw,indInfo] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);

% Create and plot the output grid for the first layer with predefined symbol
```

```
% values for the different types
createAndPlotGrid(carrier,pusch,cw,indInfo)
```



Case 2: Number of HARQ-ACK Bits Greater Than 2

For the same configuration setup mentioned in case 1, change the number of HARQ-ACK bits from 1 to 3. For this setup, the number of coded HARQ-ACK bits is 6, the number of coded CSI part 1 and part 2 bits is 19 each, and the number of coded UL-SCH bits is 106.

Step 1: When the number of HARQ-ACK bits is less than or equal to 2, find the **reserved HARQ-ACK** locations and mark them on the grid.

Because the number of HARQ-ACK bits is greater than 2 in this case, skip this step.

Step 2: Map the coded **HARQ-ACK** bits.

The HARQ-ACK is mapped to the REs in the OFDM symbol that is available after the first consecutive DM-RS OFDM symbols. The number of REs required for HARQ-ACK is 6. Because this value is not greater than half of the number of REs available for UCI transmission, the mapping of the HARQ-ACK is distributed as shown in this figure.

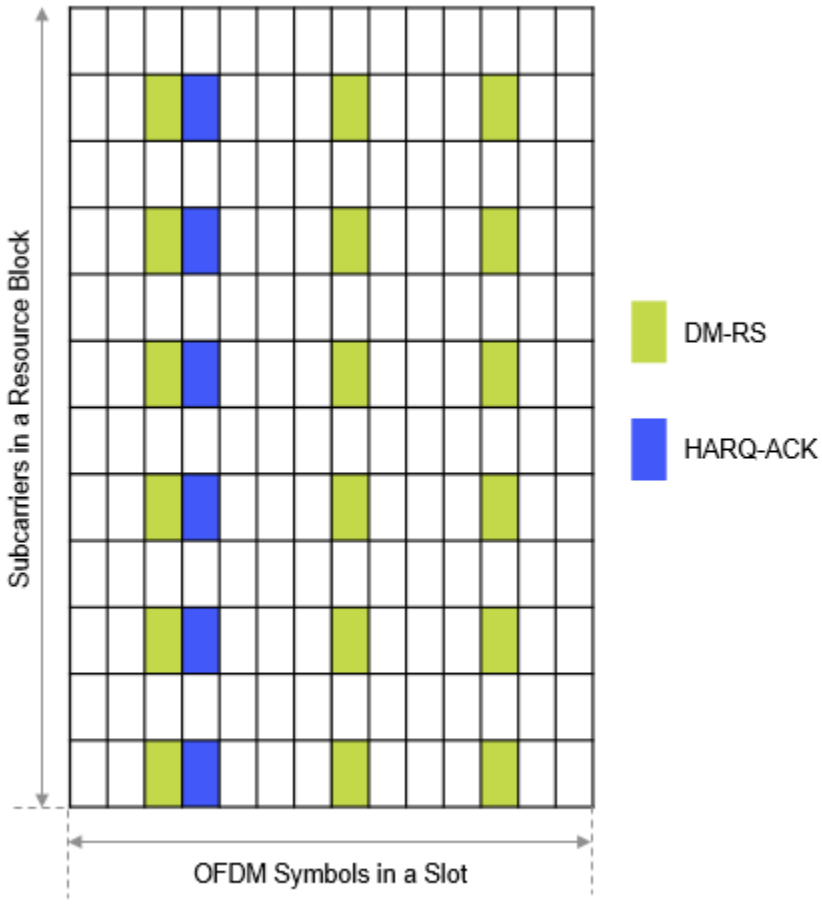


Figure 7: HARQ-ACK Locations

Step 3: Map the coded **CSI part 1** and coded **CSI part 2** bits similar to case 1.

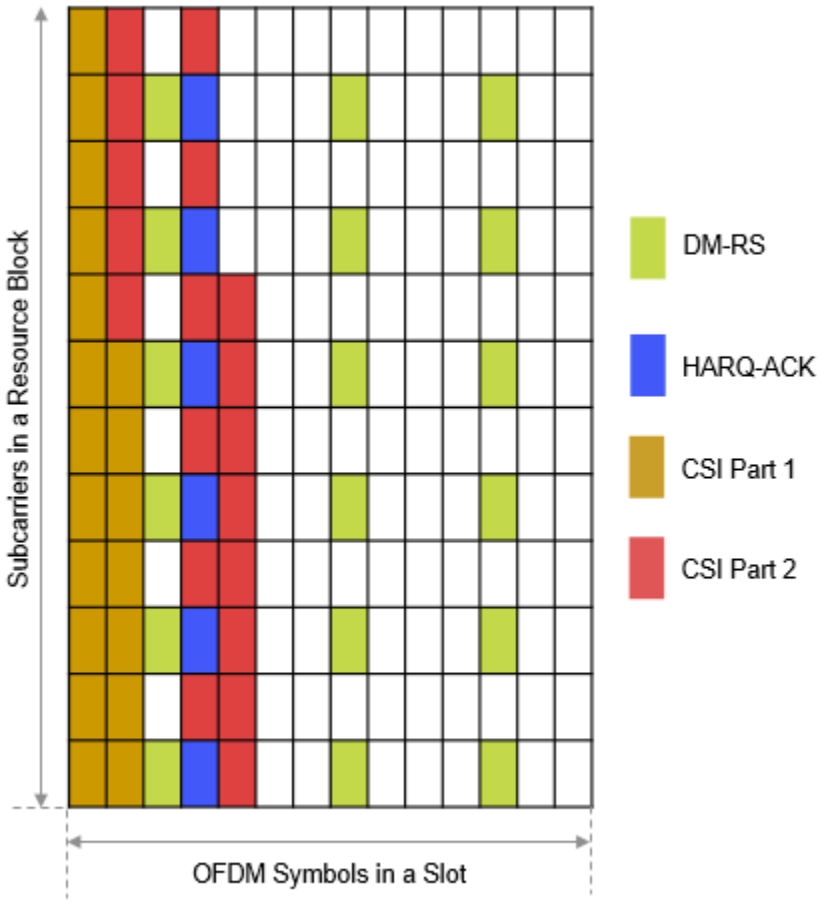


Figure 8: CSI Part 1 and CSI Part 2 Locations

Step 4: Map the coded **UL-SCH** data bits similar to case 1.

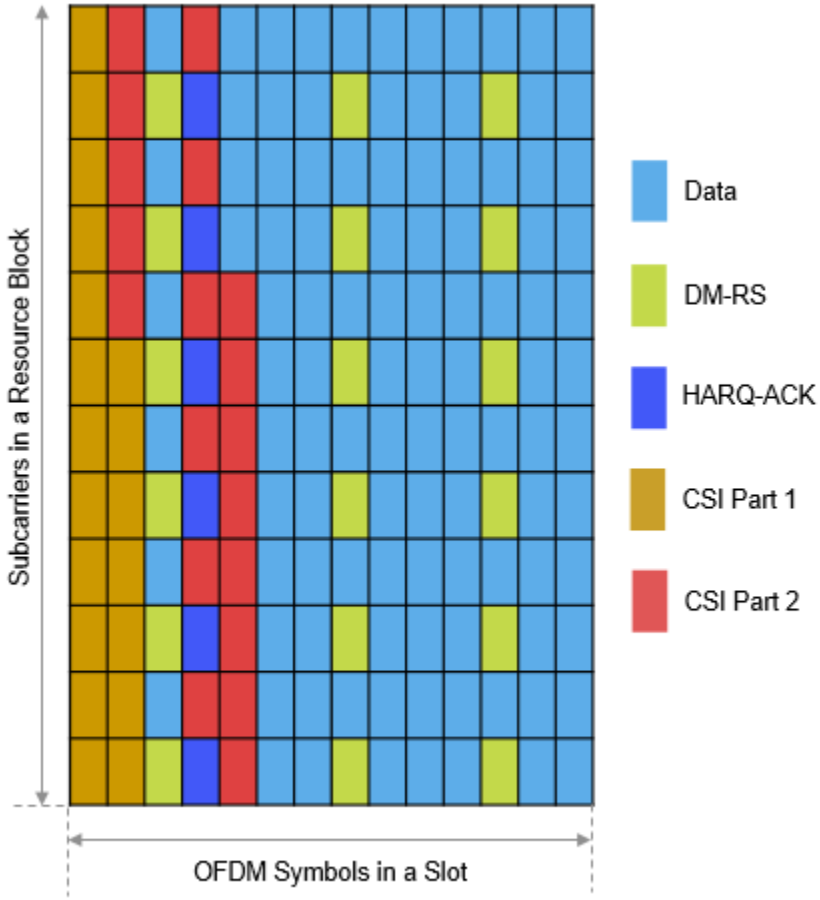


Figure 9: UL-SCH Locations

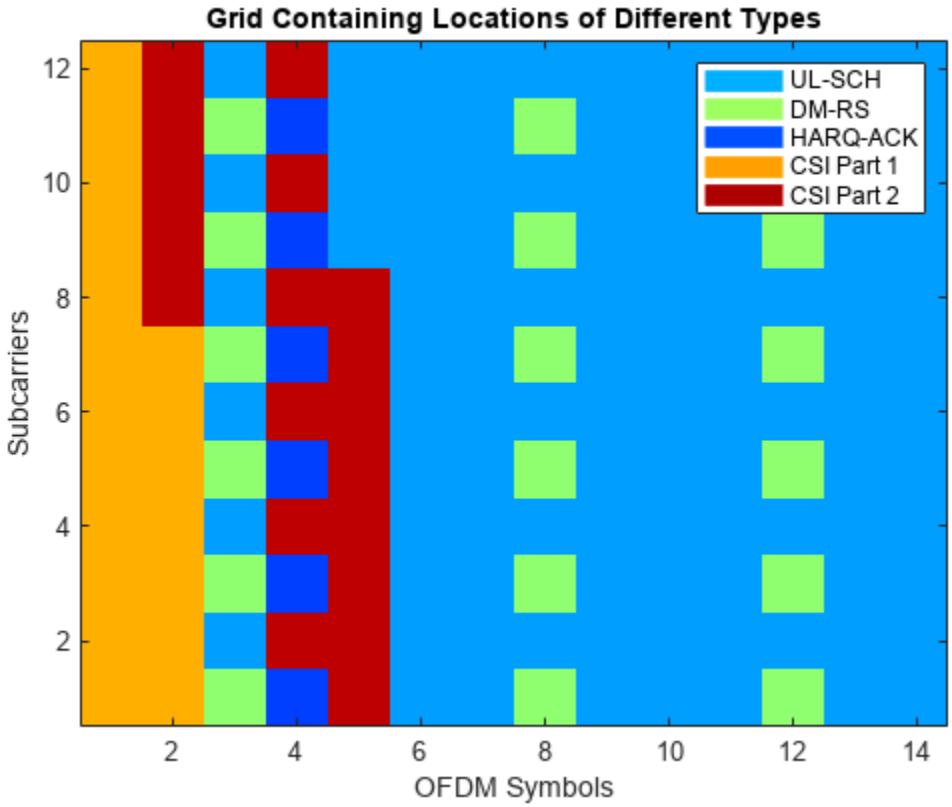
Step 5: When the number of HARQ-ACK bits is less than or equal to 2, map the coded **HARQ-ACK** bits.

Because the number of HARQ-ACK bits is greater than 2 in this case, skip this step.

Step 6: Form the **codeword.**

```
% Set the payload size of the HARQ-ACK bits to a value greater than 2
oack = 3; % Number of HARQ-ACK bits

% Get the UL-SCH coding information
info = nrULSCHInfo(pusch,tcr,tbs,oack,ocsi1,ocsi2);

% Set random coded UL-SCH, HARQ-ACK, CSI part 1, and CSI part 2 bits
culsch = randi([0 1],info.GULSCH,1);
cack = randi([0 1],info.GACK,1);
ccsi1 = randi([0 1],info.GCSI1,1);
ccsi2 = randi([0 1],info.GCSI2,1);

% Get the codeword and locations of each type (data and UCI)
[cw,indInfo] = nrULSCHMultiplex(pusch,tcr,tbs,culsch,cack,ccsi1,ccsi2);
```

```
% Create and plot the output grid for the first layer with predefined symbol
% values for the different types
createAndPlotGrid(carrier,pusch,cw,indInfo)
```



Further Exploration

Change the different parameters affecting the time or frequency allocation of the PUSCH and the number of payload bits of each UCI type to vary the RE positions of the UCI types.

Enable PT-RS to vary the bit capacities of the UCI types in the codeword.

Enable intra-slot frequency hopping to observe the mapping of different UCI types in each hop.

This example shows how to generate the bit capacities of respective UCI types and perform multiplexing to generate a codeword associated with a PUSCH. The example highlights the different steps involved in multiplexing when the number of HARQ-ACK bits is less than or equal to 2 and greater than 2.

References

1 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local Function

```
function createAndPlotGrid(carrier,pusch,cw,indInfo)

    % Initialize the grid
    qm = strcmpi(pusch.Modulation,{'pi/2-BPSK','QPSK','16QAM','64QAM','256QAM'})*[1 2 4 6 8]';
    gridOFDM = complex(zeros([carrier.NSizeGrid*12 carrier.SymbolsPerSlot double(pusch.NumLayers)

    % Specify predefined values for each entity to get the desired visualization
    chpLevel = struct;
    chpLevel.PUSCH = 0.6;
    chpLevel.DMRS = 1.1;
    chpLevel.ACK = 0.4;
    chpLevel.CSI1 = 1.5;
    chpLevel.CSI2 = 2;

    % Assign the codeword with predefined values
    cw1 = zeros(size(cw));
    cw1(indInfo.ULSCHIndices) = chpLevel.PUSCH;
    cw1(indInfo.ACKIndices) = chpLevel.ACK ;
    cw1(indInfo.CSI1Indices) = chpLevel.CSI1 ;
    cw1(indInfo.CSI2Indices) = chpLevel.CSI2 ;

    % Legend labels for the plot
    labelStr = {'UL-SCH','DM-RS','HARQ-ACK','CSI Part 1','CSI Part 2'};

    % Get the PUSCH and DM-RS indices
    puschIndices = nrPUSCHIndices(carrier,pusch);
    dmrsIndices = nrPUSCHDMRSIndices(carrier,pusch);

    % Get the PT-RS indices
    if pusch.EnablePTRS
        labelStr{end+1} = 'PT-RS';
        chpLevel.PTRS = 1.6;
        ptrsIndices = nrPUSCHPTRSIndices(carrier,pusch);
    end

    % Map the DM-RS, PT-RS, and PUSCH indices to the grid
    [~,puschExtInd] = nrExtractResources(puschIndices,gridOFDM);
    gridOFDM(dmrsIndices) = chpLevel.DMRS;
    if pusch.TransformPrecoding
        % DFT-s-OFDM
        titleStr = "Projections of Different Types Before Transform Precoding";
        cwLen = zeros(size(puschIndices,1),pusch.NumLayers*qm);
        if pusch.EnablePTRS
            cwLen(ptrsIndices) = chpLevel.PTRS;
        end
        cwLen(cwLen==0) = cw1;
        gridOFDM(reshape(puschExtInd',[],1)) = cwLen;
    else
        % CP-OFDM
        titleStr = "Grid Containing Locations of Different Types";
        if pusch.EnablePTRS
            gridOFDM(ptrsIndices) =  chpLevel.PTRS;
        end
        gridOFDM(reshape(puschExtInd',[],1)) = cw1;
    end
```

```
    % Plot the grid
    figure
    map = jet(64);
    cscaling = 30;
    im = image(1:size(gridOFDM,2),1:size(gridOFDM,1),cscaling*abs(gridOFDM(:,:,1)));
    colormap(im.Parent,map)

    % Add a legend to the image
    chpval = struct2cell(chpLevel);
    clevels = cscaling*[chpval{:}];
    N = length(clevels);
    L = line(ones(N),ones(N), 'LineWidth',8); % Generate lines
    % Index the color map and associate the selected colors with the lines
    set(L,{'color'},mat2cell(map(min(1+clevels,length(map)),:),ones(1,N),3)) % Set the colors ac
    % Create the legend
    legend(labelStr{:})
    axis xy
    ylabel('Subcarriers')
    xlabel('OFDM Symbols')
    title(titleStr)

end
```

See Also

Functions
`nrULSCHInfo` | `nrULSCHMultiplex`

Objects
`nrCarrierConfig` | `nrPUSCHConfig`

Related Examples

- “NR PUSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 2-25

Physical Layer Subcomponents

5G LDPC Block Error Rate Simulation Using the Cloud or a Cluster

This example shows how to use the cloud or a cluster for block error rate (BLER) simulation of low-density parity-check (LDPC) coding for the 5G NR downlink shared transport channel (DL-SCH).

For ultra-reliable low-latency communication in 5G systems, some use cases require a BLER as low as 1e-6 [1]. In this low BLER region, obtaining accurate results requires simulating many millions of blocks. On a single desktop computer, this simulation can take days. You can use the cloud or a cluster to reduce simulation time. For example, a 256-worker cluster on AWS is about 42 times as fast as a 6-core desktop in one test scenario. For more details, see Sample Results.

In this example, you first calculate one point on a BLER curve by using a single desktop computer. You then use MATLAB Parallel Server in the cloud or on a cluster on your local network to calculate the BLER curve across a range of signal-to-noise ratios.

DL-SCH with LDPC Coding

First, simulate one transport block for the 5G NR DL-SCH with LDPC coding. This code is the basis of the `helperLDPCBLERSim` function which uses `parfor` to simulate many transport blocks in parallel.

```
% Set up DL-SCH coding parameters
TBS = 3816;             % Transport block size, a positive integer
codeRate = 308/1024;    % Target code rate, a real number between 0 and 1
rv = 0;                 % Redundancy version, 0-3
modulation = 'QPSK';    % Modulation scheme, QPSK, 16QAM, 64QAM, 256QAM
nlayers = 1;            % Number of layers, 1-4 for a transport block
cbsInfo = nrDLSCHInfo(TBS,codeRate);
disp('DL-SCH coding parameters')
disp(cbsInfo)

switch modulation
    case 'QPSK'
        bitsPerSymbol = 2;
    case '16QAM'
        bitsPerSymbol = 4;
    case '64QAM'
        bitsPerSymbol = 6;
    case '256QAM'
        bitsPerSymbol = 8;
end

% Set up AWGN channel
EbNo = 1.25; % in dB
outlen = ceil(TBS/codeRate);
snrdB = convertSNR(EbNo,"ebno",...
    BitsPerSymbol=bitsPerSymbol,CodingRate=TBS/outlen);

% Random transport block data generation
in = randi([0 1],TBS,1,'int8');
% Transport block CRC attachment
tbIn = nrCRCEncode(in,cbsInfo.CRC);
% Code block segmentation and CRC attachment
cbsIn = nrCodeBlockSegmentLDPC(tbIn,cbsInfo.BGN);
% LDPC encoding
```

```
enc = nrLDPCEncode(cbsIn,cbsInfo.BGN);
% Rate matching and code block concatenation
chIn = nrRateMatchLDPC(enc,outlen,rv,modulation,nlayers);
% Symbol mapping
symOut = nrSymbolModulate(chIn,modulation);
% AWGN channel
[rxSig, noiseVar] = awgn(symOut,snrdB);
% Symbol demapping
rxllr = nrSymbolDemodulate(rxSig,modulation,noiseVar);
% Rate recovery
raterec = nrRateRecoverLDPC(rxllr,TBS,codeRate,rv,modulation,nlayers);
% LDPC decoding, with early termination and at most 12 iterations
decBits = nrLDPCDecode(raterec,cbsInfo.BGN,12);
% Code block desegmentation and CRC decoding
[blk,~] = nrCodeBlockDesegmentLDPC(decBits,cbsInfo.BGN,TBS+cbsInfo.L);
% Transport block CRC decoding
[out,~] = nrCRCDecode(blk,cbsInfo.CRC);
% Compare
blockError = any(out~=in)
```

```
DL-SCH coding parameters
    CRC: '16'
      L: 16
    BGN: 2
      C: 1
    Lcb: 0
      F: 8
     Zc: 384
      K: 3840
      N: 19200


blockError =

  logical

   0
```

Parallelization Strategy

The `helperLDPCBLERSim` function uses the current parallel pool to calculate the LDPC BLER in parallel. For each `EbNo` value, the function simulates blocks by successive `parfor` calls until a target number of block errors is achieved or each worker on a cluster has simulated a specified maximum number of blocks. In each `parfor` call, all workers on the cluster work in parallel for the same `EbNo` value and simulate the same number of blocks, so all workers are expected to finish their computation at roughly the same time.

Preventing some workers from finishing earlier than other workers is crucial to the efficient use of a cluster, and minimizing overall execution time.

Use Desktop Computer to Get One Point on BLER Curve

```
delete(gcp('nocreate'));  % If a parpool exists, shut it down first
pool = parpool('local');  % Create a local parpool for helperLDPCBLERSim
targetNumBlockErrors = 100;
numBlocksInsideParfor = 1000; % Number of blocks to simulate per worker in one parfor loop
maxNumBlocksPerWorker = 1e9;
```

```
[BLER, snrdB, finalNumBlockErrors, finalNumBlocks, executionTime] = ...
    helperLDPCBLERSim(TBS, codeRate, EbNo, targetNumBlockErrors, ...
        maxNumBlocksPerWorker, numBlocksInsideParfor)
delete(pool);

Starting parallel pool (parpool) using the 'Processes' profile ...
Connected to parallel pool with 20 workers.
Begin simulation for EbNo = 1.25 using 20 workers
EbNo = 1.25  Elapsed time = 00:01:42  Number of blocks simulated = 20000  Number of block errors

BLER =

    0.0340


snrdB =

   -0.9572


finalNumBlockErrors =

   681


finalNumBlocks =

       20000


executionTime =

  102.6536

Parallel pool using the 'Processes' profile is shutting down.
```

Set up Your Cluster

Before you can run the next sections, you must get access to a cluster. On the MATLAB **Home** tab, go to **Parallel** > **Discover Clusters** to find out if you already have access to a cluster with MATLAB Parallel Server™. For more information, see “Discover Clusters” (Parallel Computing Toolbox).

Use Cluster on Your Local Network to Generate One BLER Curve

Check whether this MATLAB session has access to a MATLAB Parallel Server cluster on your local network. If so, create a parpool to use all workers on the cluster and then generate a BLER curve.

```
[pool, cluster] = helperCreateParpool("Cluster");
EbNo = -0.5:0.25:2.25;
if ~isempty(cluster)
    disp('Found MATLAB Parallel Server cluster on your local network')
    [BLER, snrdB, finalNumBlockErrors, finalNumBlocks, executionTime] = ...
        helperLDPCBLERSim(TBS, codeRate, EbNo, targetNumBlockErrors*ones(size(EbNo)), ...
            maxNumBlocksPerWorker*ones(size(EbNo)), numBlocksInsideParfor*ones(size(EbNo)));
    figure
    semilogy(snrdB,BLER,'x-')
    xlabel('SNR (dB)');
    ylabel('BLER')
```

```
    grid on
    delete(pool);
end
```

Use the Cloud to Generate One BLER Curve

Check whether this MATLAB session has access to a MATLAB Parallel Server cluster running in AWS managed by MathWorks Cloud Center. If so, create a parpool to use all workers on the cluster and then generate a BLER curve.

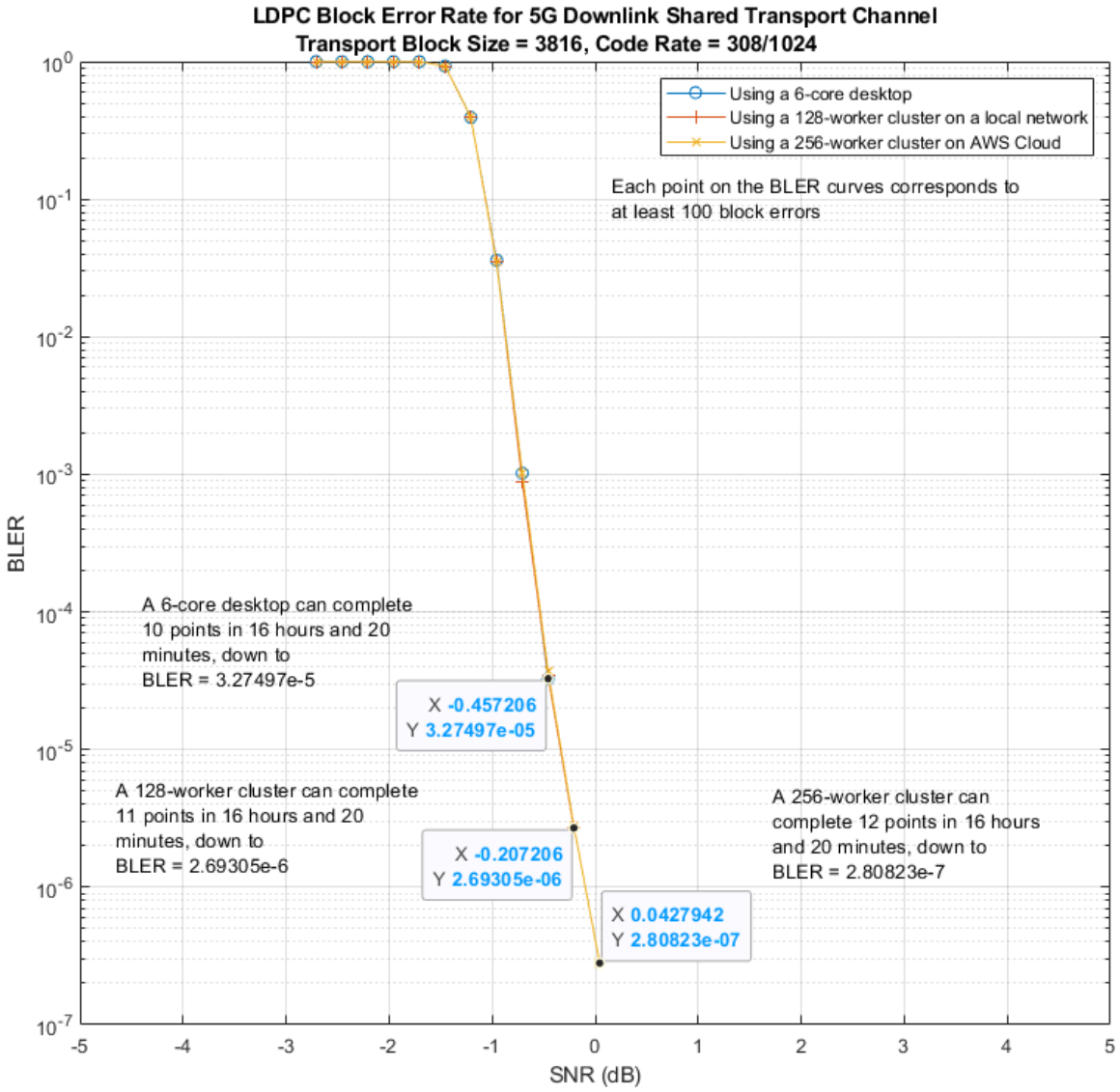
To learn how to start and test your first cluster on the cloud, see “Start and Test MATLAB Parallel Server Cluster in Cloud Center”.

```
[pool, cluster] = helperCreateParpool("Cloud");
if ~isempty(cluster)
    disp('Found MATLAB Parallel Server cluster on the cloud')
    if strcmpi(cluster.State,'online')
        [BLER, snrdB, finalNumBlockErrors, finalNumBlocks, executionTime] = ...
            helperLDPCBLERSim(TBS, codeRate, EbNo, targetNumBlockErrors*ones(size(EbNo)), ...
                maxNumBlocksPerWorker*ones(size(EbNo)), numBlocksInsideParfor*ones(size(EbNo)));
        figure
        semilogy(snrdB,BLER,'x-')
        xlabel('SNR (dB)');
        ylabel('BLER')
        grid on
        delete(pool);
        disp("If you do not need to use the cluster, go to https://cloudcenter.mathworks.com and
    else
        disp("The cluster " + cluster.Name + " is not online. Go to https://cloudcenter.mathworks
    end
end
```

Sample Results

This example compares running on a 6-core desktop, a 128-worker cluster on a local network, and a 256-worker cluster on AWS Cloud to generate three BLER curves. The desktop has a 6-core Intel Xeon W-2133 processor. The 128-worker cluster has eight 16-core Intel Xeon E5-2683 v4 processors. The 256-worker cluster has sixteen m5.8xlarge AWS EC2 machines (16 cores per machine). In 16 hours and 20 minutes, the 256-worker cluster completed 12 points, down to BLER = 2.80823e-7. In the same period of time, the 128-worker cluster completed 11 points (down to BLER = 2.69305e-6) and the 6-core desktop completed 10 points (down to BLER = 3.27497e-5).

```
blerFig = openfig('LDPCBLERcurves.fig');
```



To compare the relative speeds of the three platforms, the execution times for simulating 768,000 blocks at two SNR values were logged.

```
-------------------------------------------------------------------------
   SNR      |  6-core desktop  |  128-worker cluster  |  256-worker cluster
-------------------------------------------------------------------------
 -0.457 dB  |   5455 seconds   |   407 seconds        |   129 seconds
-------------------------------------------------------------------------
 -0.207 dB  |   5006 seconds   |   381 seconds        |   120 seconds
-------------------------------------------------------------------------
```

At SNR = -0.457 dB, the 256-worker cluster is about 5455/129 = 42.287 times as fast as the 6-core desktop, and about 407/129 = 3.155 times as fast as the 128-worker cluster.

At SNR = -0.207 dB, the 256-worker cluster is about 5006/120 = 41.717 times as fast as the 6-core desktop, and about 381/120 = 3.175 times as fast as the 128-worker cluster.

Hence, the 256-worker cluster is generally about 42 times as fast as the 6-core desktop and about 3.16 times as fast as the 128-worker cluster.

Further Exploration

You can use the framework of this example to generate BLER curves for other transport block sizes and code rates. The speedup achievable by a sufficiently large cluster makes it practical to find LDPC error floors (probably below BLER = 1e-7) at high SNR values.

You can also explore the helper functions for creating a parallel pool and running an LDPC BLER simulation:

- helperCreateParpool.m
- helperLDPCBLERSim.m

References

1 Sybis, Michal, Krzysztof Wesolowski, Keeth Jayasinghe, Venkatkumar Venkatasubramanian, and Vladimir Vukadinovic. "Channel Coding for Ultra-Reliable Low-Latency Communication in 5G Systems." In 2016 IEEE 84th Vehicular Technology Conference (VTC-Fall), 1-5. Montreal, QC, Canada: IEEE, 2016. https://doi.org/10.1109/VTCFall.2016.7880930.

See Also

Related Examples

- “Start and Test MATLAB Parallel Server Cluster in Cloud Center”

Signal Reception

Extract PBCH Symbols and Channel Estimates for Decoding

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

PBCH Coding and Beamforming

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

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

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

```
carrier = nrCarrierConfig('NSizeGrid',20);
P = 2;
txGrid = nrResourceGrid(carrier,P);
F = [1 0.3i];
[~,bfInd] = nrExtractResources(pbchInd,txGrid);
txGrid(bfInd) = pbchTxSym*F;
```

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

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

PBCH Transmission and Decoding

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

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

Create channel estimates including beamforming.

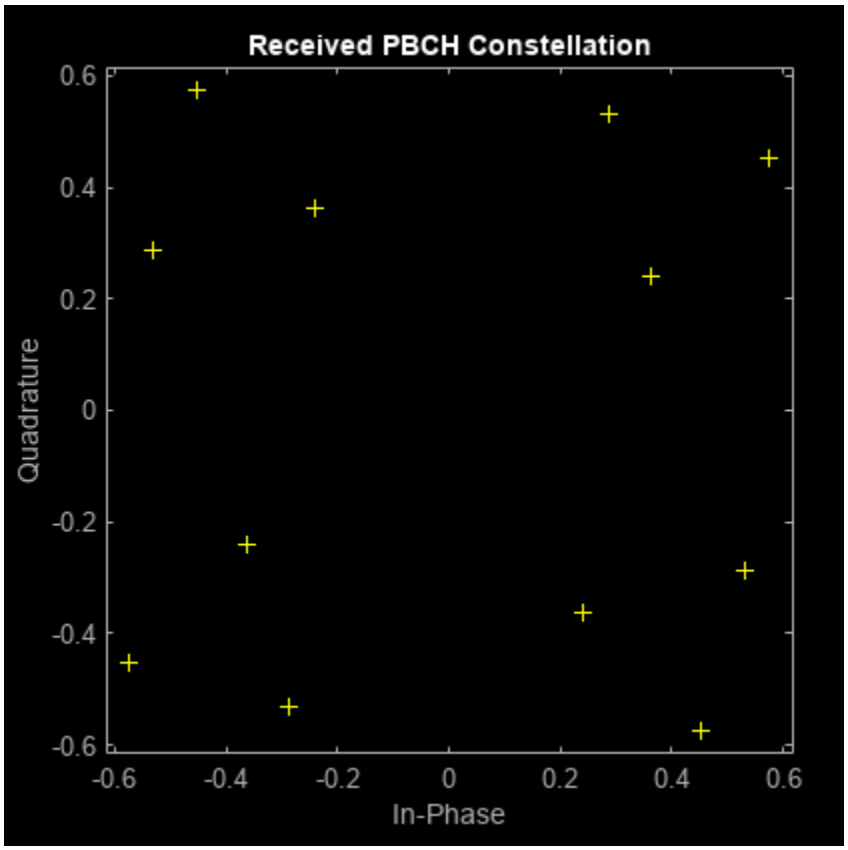
```
 hEstGrid = repmat(permute(H.'*F.',[3 4 1 2]),[240 4]);
 nEst = 0;
```

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

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

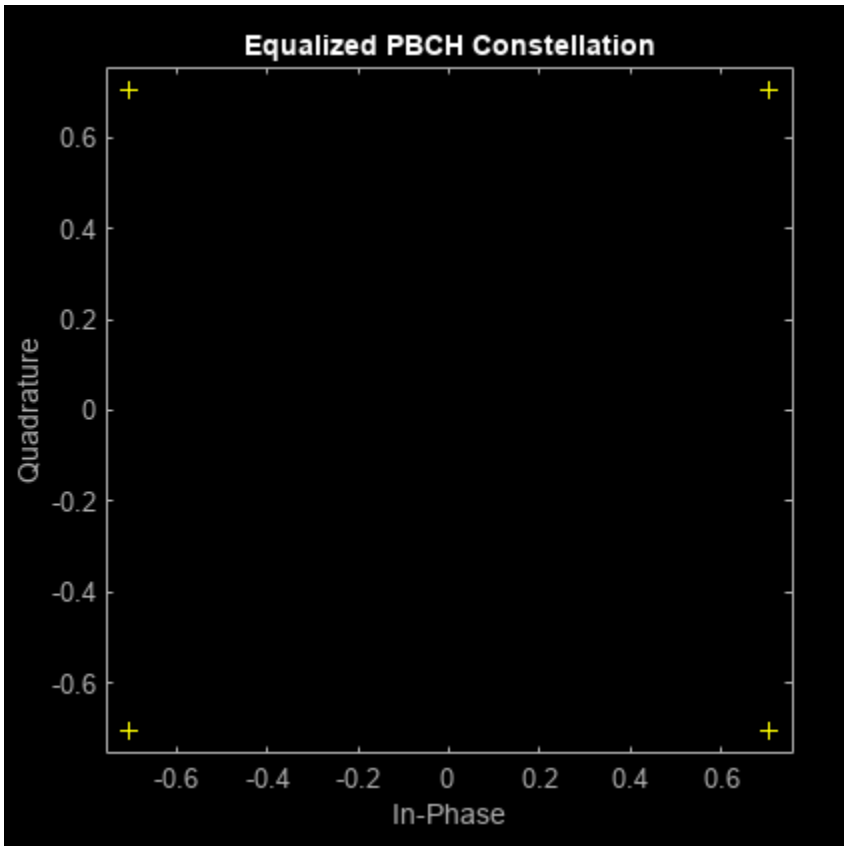
In preparation for PBCH decoding, extract symbols from the received grid and the channel estimate grid.

```
[pbchRxSym,pbchHestSym] = nrExtractResources(pbchInd,rxGrid,hEstGrid);
scatterplot(pbchRxSym(:),[],[],'y+');
title('Received PBCH Constellation');
```



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

```
pbchEqSym = nrEqualizeMMSE(pbchRxSym,pbchHestSym,nEst);
scatterplot(pbchEqSym(:),[],[],'y+');
title('Equalized PBCH Constellation');
```



Retrieve soft bits by performing PBCH decoding on the equalized symbols.

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

```
pbchBits = 864×1
10^10 ×

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

See Also

Functions

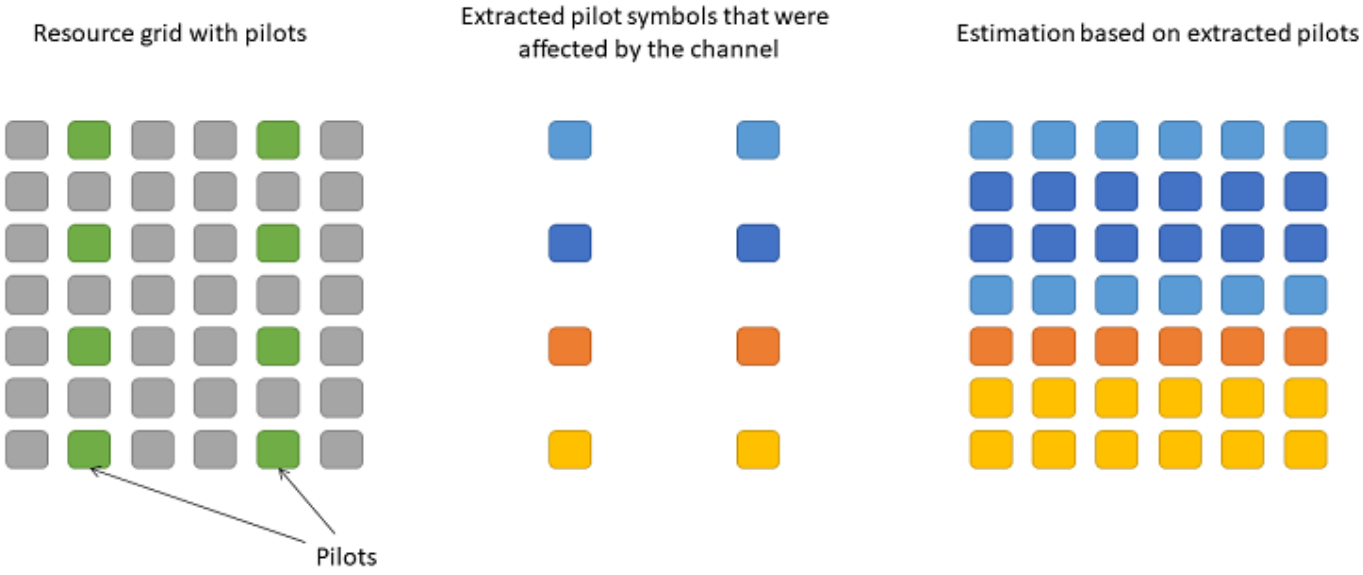
`nrEqualizeMMSE` | `nrExtractResources`

Deep Learning Data Synthesis for 5G Channel Estimation

This example shows how to train a convolutional neural network (CNN) for channel estimation using Deep Learning Toolbox™ and data generated with 5G Toolbox™. Using the trained CNN, you perform channel estimation in single-input single-output (SISO) mode, utilizing the physical downlink shared channel (PDSCH) demodulation reference signal (DM-RS).

Introduction

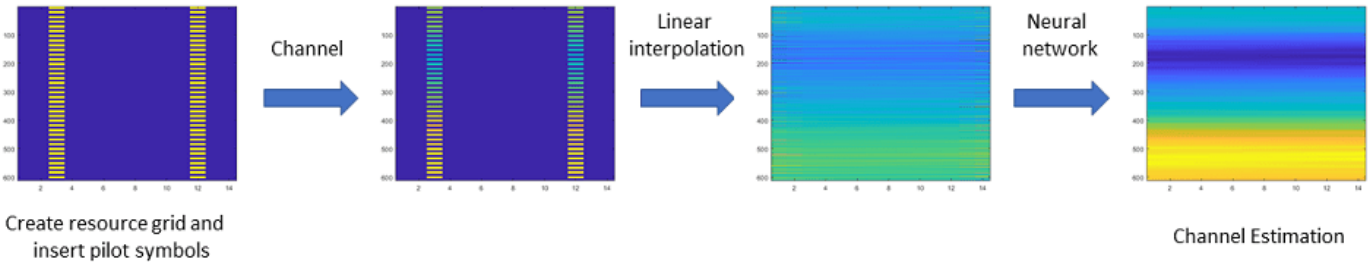
The general approach to channel estimation is to insert known reference pilot symbols into the transmission and then interpolate the rest of the channel response by using these pilot symbols.



For an example showing how to use this channel estimation approach, see “NR PDSCH Throughput” on page 1-58.

You can also use deep learning techniques to perform channel estimation. For example, by viewing the resource grid as a 2-D image, you can turn the problem of channel estimation into an image processing problem, similar to denoising or super-resolution, where CNNs are effective.

Using 5G Toolbox, you can customize and generate standard-compliant waveforms and channel models to use as training data. Using Deep Learning Toolbox, you can use this training data to train a channel estimation CNN. This example shows how to generate such training data and how to train a channel estimation CNN. The example also shows how to use the channel estimation CNN to process images that contain linearly interpolated received pilot symbols. The example concludes by visualizing the results of the neural network channel estimator in comparison to practical and perfect estimators.



Neural Network Training

Neural network training consists of these steps:

- Data generation
- Splitting the generated data into training and validation sets
- Defining the CNN architecture
- Specifying the training options, optimizer, and learning rate
- Training the network

Due to the large number of signals and possible scenarios, training can take several minutes. By default, training is disabled, a pretrained model is used. You can enable training by setting `trainModel` to true.

```
trainModel = false;
```

If you have Parallel Computing Toolbox™ installed and a supported CUDA-enabled NVIDIA® GPU set up, the network training uses GPU acceleration by default. The `trainNetwork` (Deep Learning Toolbox) function allows you to override this default behavior. For a list of supported GPUs, see “GPU Computing Requirements” (Parallel Computing Toolbox).

Data generation is set to produce 256 training examples or training data sets. This amount of data is sufficient to train a functional channel estimation network on a CPU in a reasonable time. For comparison, the pretrained model is based on 16,384 training examples.

Training data of the CNN model has a fixed size dimensionality, the network can only accept 612-by-14-by-1 grids, i.e. 612 subcarriers, 14 OFDM symbols and 1 antenna. Therefore, the model can only operate on a fixed bandwidth allocation, cyclic prefix length, and a single receive antenna.

The CNN treats the resource grids as 2-D images, hence each element of the grid must be a real number. In a channel estimation scenario, the resource grids have complex data. Therefore, the real and imaginary parts of these grids are input separately to the CNN. In this example, the training data is converted from a complex 612-by-14 matrix into a real-valued 612-by-14-by-2 matrix, where the third dimension denotes the real and imaginary components. Because you have to input the real and imaginary grids into the neural network separately when making predictions, the example converts the training data into 4-D arrays of the form 612-by-14-by-1-by-2N, where N is the number of training examples.

To ensure that the CNN does not overfit the training data, the training data is split into validation and training sets. The validation data is used for monitoring the performance of the trained neural network at regular intervals, as defined by `valFrequency`, approximately 5 per epoch. Stop training when the validation loss stops improving. In this instance, the validation data size is the same as the size of a single mini-batch due to the small size of the data set.

The returned channel estimation CNN is trained on various channel configurations based on different delay spreads, doppler shifts, and SNR ranges between 0 and 10 dB.

```
% Set the random seed for reproducibility (this has no effect if a GPU is
% used)
rng(42)

if trainModel
    % Generate the training data
    [trainData,trainLabels] = hGenerateTrainingData(256);

    % Set the number of examples per mini-batch
    batchSize = 32;

    % Split real and imaginary grids into 2 image sets, then concatenate
    trainData = cat(4,trainData(:,:,1,:),trainData(:,:,2,:));
    trainLabels = cat(4,trainLabels(:,:,1,:),trainLabels(:,:,2,:));

    % Split into training and validation sets
    valData = trainData(:,:,:,1:batchSize);
    valLabels = trainLabels(:,:,:,1:batchSize);

    trainData = trainData(:,:,:,batchSize+1:end);
    trainLabels = trainLabels(:,:,:,batchSize+1:end);

    % Validate roughly 5 times every epoch
    valFrequency = round(size(trainData,4)/batchSize/5);

    % Define the CNN structure
    layers = [ ...
        imageInputLayer([612 14 1],'Normalization','none')
        convolution2dLayer(9,64,'Padding',4)
        reluLayer
        convolution2dLayer(5,64,'Padding',2,'NumChannels',64)
        reluLayer
        convolution2dLayer(5,64,'Padding',2,'NumChannels',64)
        reluLayer
        convolution2dLayer(5,32,'Padding',2,'NumChannels',64)
        reluLayer
        convolution2dLayer(5,1,'Padding',2,'NumChannels',32)
        regressionLayer
    ];

    % Set up a training policy
    options = trainingOptions('adam', ...
        'InitialLearnRate',3e-4, ...
        'MaxEpochs',5, ...
        'Shuffle','every-epoch', ...
        'Verbose',false, ...
        'Plots','training-progress', ...
        'MiniBatchSize',batchSize, ...
        'ValidationData',{valData, valLabels}, ...
        'ValidationFrequency',valFrequency, ...
        'ValidationPatience',5);

    % Train the network. The saved structure trainingInfo contains the
    % training progress for later inspection. This structure is useful for
    % comparing optimal convergence speeds of different optimization
```

```
    % methods.
    [channelEstimationCNN,trainingInfo] = trainNetwork(trainData, ...
        trainLabels,layers,options);

else
    % Load pretrained network if trainModel is set to false
    load('trainedChannelEstimationNetwork.mat')
end
```

Inspect the composition and individual layers of the model. The model has 5 convolutional layers. The input layer expects matrices of size 612-by-14, where 612 is the number of subcarriers and 14 is the number of OFDM symbols. Each element is a real number, since the real and imaginary parts of the complex grids are input separately.

```
channelEstimationCNN.Layers
```

```
ans =

  11x1 Layer array with layers:

     1   'imageinput'         Image Input         612x14x1 images
     2   'conv_1'             2-D Convolution     64 9x9x1 convolutions with stride [1  1] and pa
     3   'relu_1'             ReLU                ReLU
     4   'conv_2'             2-D Convolution     64 5x5x64 convolutions with stride [1  1] and p
     5   'relu_2'             ReLU                ReLU
     6   'conv_3'             2-D Convolution     64 5x5x64 convolutions with stride [1  1] and p
     7   'relu_3'             ReLU                ReLU
     8   'conv_4'             2-D Convolution     32 5x5x64 convolutions with stride [1  1] and p
     9   'relu_4'             ReLU                ReLU
    10   'conv_5'             2-D Convolution     1 5x5x32 convolutions with stride [1  1] and pa
    11   'regressionoutput'   Regression Output   mean-squared-error with response 'Response'
```

Create Channel Model for Simulation

Set the simulation noise level in dB. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
SNRdB = 10;
```

Load the predefined simulation parameters, including the PDSCH parameters and DM-RS configuration.

```
simParameters = hDeepLearningChanEstSimParameters();
carrier = simParameters.Carrier;
pdsch = simParameters.PDSCH;
```

Create a TDL channel model and set channel parameters. To compare different channel responses of the estimators, you can change these parameters later.

```
channel = nrTDLChannel;
channel.Seed = 0;
channel.DelayProfile = 'TDL-A';
channel.DelaySpread = 3e-7;
channel.MaximumDopplerShift = 50;

% This example supports only SISO configuration
channel.NumTransmitAntennas = 1;
```

```
channel.NumReceiveAntennas = 1;

waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;
```

Get the maximum number of delayed samples by a channel multipath component. This number is calculated from the channel path with the largest delay and the implementation delay of the channel filter. This number is needed to flush the channel filter when obtaining the received signal.

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

Simulate PDSCH DM-RS Transmission

Simulate a PDSCH DM-RS transmission by performing these steps:

- Generate the resource grid
- Insert DM-RS symbols
- Perform OFDM modulation
- Send modulated waveform through the channel model
- Add white Gaussian noise
- Perform perfect timing synchronization
- Perform OFDM demodulation

The DM-RS symbols in the grid are used for channel estimation. This example does not transmit any data, therefore, the resource grid does not include any PDSCH symbols.

```
% Generate DM-RS indices and symbols
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);

% Create resource grid
pdschGrid = nrResourceGrid(carrier);

% Map PDSCH DM-RS symbols to the grid
pdschGrid(dmrsIndices) = dmrsSymbols;

% OFDM-modulate associated resource elements
txWaveform = nrOFDMModulate(carrier,pdschGrid);
```

To flush the channel content, append zeros at the end of the transmitted waveform. These zeros take into account any delay introduced in the channel, such as multipath and implementation delay. The number of zeros depends on the sampling rate, delay profile, and delay spread.

```
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))];
```

Send data through the TDL channel model.

```
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);
```

Add additive white Gaussian noise (AWGN) to the received time-domain waveform. To take into account sampling rate, normalize the noise power. The SNR is defined per resource element (RE) for each receive antenna (3GPP TS 38.101-4). For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
SNR = 10^(SNRdB/10); % Calculate linear SNR
N0 = 1/sqrt(2.0*simParameters.NRxAnts*double(waveformInfo.Nfft)*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;
```

Perform perfect synchronization. To find the strongest multipath component, use the information provided by the channel.

```
% Get path filters for perfect channel estimation
pathFilters = getPathFilters(channel);
[offset,~] = nrPerfectTimingEstimate(pathGains,pathFilters);

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

OFDM-demodulate the received data to recreate the resource grid.

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

% Pad the grid with zeros in case an incomplete slot has been demodulated
[K,L,R] = size(rxGrid);
if (L < carrier.SymbolsPerSlot)
    rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
end
```

Compare and Visualize Various Channel Estimations

You can perform and compare the results of perfect, practical, and neural network estimations of the same channel model.

To perform perfect channel estimation, use the `nrPerfectChannelEstimate` function using the value of the path gains provided by the channel.

```
estChannelGridPerfect = nrPerfectChannelEstimate(carrier,pathGains, ...
    pathFilters,offset,sampleTimes);
```

To perform practical channel estimation, use the `nrChannelEstimate` function.

```
[estChannelGrid,~] = nrChannelEstimate(carrier,rxGrid,dmrsIndices, ...
    dmrsSymbols,'CDMLengths',pdsch.DMRS.CDMLengths);
```

To perform channel estimation using the neural network, you must interpolate the received grid. Then split the interpolated image into its real and imaginary parts and input these images together into the neural network as a single batch. Use the `predict` (Deep Learning Toolbox) function to make predictions on the real and imaginary images. Finally, concatenate and transform the results back into complex data.

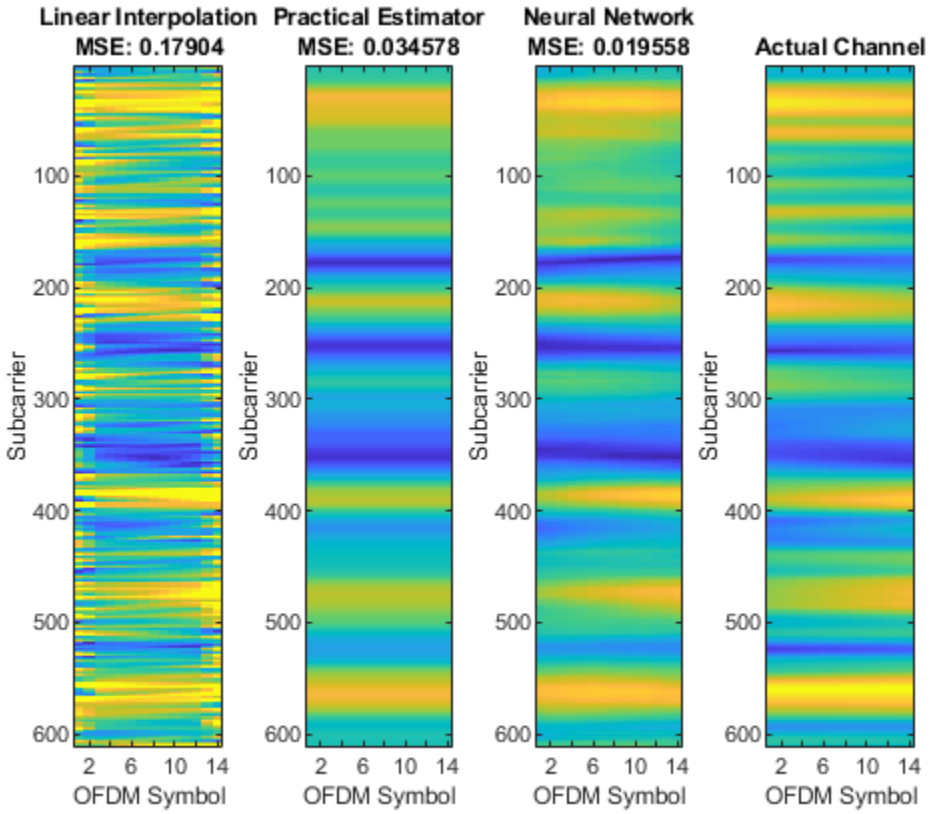
```
% Interpolate the received resource grid using pilot symbol locations
interpChannelGrid = hPreprocessInput(rxGrid,dmrsIndices,dmrsSymbols);

% Concatenate the real and imaginary grids along the batch dimension
nnInput = cat(4,real(interpChannelGrid),imag(interpChannelGrid));

% Use the neural network to estimate the channel
estChannelGridNN = predict(channelEstimationCNN,nnInput);

% Convert results to complex
estChannelGridNN = complex(estChannelGridNN(:,:,:,1),estChannelGridNN(:,:,:,2));
```

Calculate the mean squared error (MSE) of each estimation method.

```
neural_mse = mean(abs(estChannelGridPerfect(:) - estChannelGridNN(:)).^2);
interp_mse = mean(abs(estChannelGridPerfect(:) - interpChannelGrid(:)).^2);
practical_mse = mean(abs(estChannelGridPerfect(:) - estChannelGrid(:)).^2);
```

Plot the individual channel estimations and the actual channel realization obtained from the channel filter taps. Both the practical estimator and the neural network estimator outperform linear interpolation.

```
plotChEstimates(interpChannelGrid,estChannelGrid,estChannelGridNN,estChannelGridPerfect,...
    interp_mse,practical_mse,neural_mse);
```



References

1. van de Beek, Jan–Jaap, Ove Edfors, Magnus Sandell, Sarah Kate Wilson, and Per Ola Borjesson. “On Channel Estimation in OFDM Systems.” In 1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty–First Century, 2:815–19, July 1995.
2. Ye, Hao, Geoffrey Ye Li, and Biing-Hwang Juang. “Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems.” IEEE Wireless Communications Letters 7, no. 1 (February 2018): 114–17.
3. Soltani, Mehran, Vahid Pourahmadi, Ali Mirzaei, and Hamid Sheikhzadeh. “Deep Learning–Based Channel Estimation.” Preprint, submitted October 13, 2018.

Local Functions

```
function hest = hPreprocessInput(rxGrid,dmrsIndices,dmrsSymbols)
% Perform linear interpolation of the grid and input the result to the
% neural network This helper function extracts the DM-RS symbols from
% dmrsIndices locations in the received grid rxGrid and performs linear
% interpolation on the extracted pilots.

    % Obtain pilot symbol estimates
    dmrsRx = rxGrid(dmrsIndices);
    dmrsEsts = dmrsRx .* conj(dmrsSymbols);

    % Create empty grids to fill after linear interpolation
    [rxDMRSGrid, hest] = deal(zeros(size(rxGrid)));
    rxDMRSGrid(dmrsIndices) = dmrsSymbols;

    % Find the row and column coordinates for a given DMRS configuration
    [rows,cols] = find(rxDMRSGrid ~= 0);
    dmrsSubs = [rows,cols,ones(size(cols))];
    [l_hest,k_hest] = meshgrid(1:size(hest,2),1:size(hest,1));

    % Perform linear interpolation
    f = scatteredInterpolant(dmrsSubs(:,2),dmrsSubs(:,1),dmrsEsts);
    hest = f(l_hest,k_hest);

end

function [trainData,trainLabels] = hGenerateTrainingData(dataSize)
% Generate training data examples for channel estimation. Run dataSize
% number of iterations to create random channel configurations and pass an
% OFDM-modulated fixed resource grid with only the DM-RS symbols inserted.
% Perform perfect timing synchronization and OFDM demodulation, extracting
% the pilot symbols and performing linear interpolation at each iteration.
% Use perfect channel information to create the label data. The function
% returns 2 arrays - the training data and labels.

    fprintf('Starting data generation...\n')

    % List of possible channel profiles
    delayProfiles = {'TDL-A', 'TDL-B', 'TDL-C', 'TDL-D', 'TDL-E'};

    simParameters = hDeepLearningChanEstSimParameters();
    carrier = simParameters.Carrier;
    pdsch = simParameters.PDSCH;

    % Create the channel model object
    nTxAnts = simParameters.NTxAnts;
    nRxAnts = simParameters.NRxAnts;

    channel = nrTDLChannel; % TDL channel object
    channel.NumTransmitAntennas = nTxAnts;
    channel.NumReceiveAntennas = nRxAnts;

    % Use the value returned from <matlab:edit('nrOFDMInfo') nrOFDMInfo> to
    % set the channel model sample rate
    waveformInfo = nrOFDMInfo(carrier);
    channel.SampleRate = waveformInfo.SampleRate;
```

```
% Get the maximum number of delayed samples by a channel multipath
% component. This number is calculated from the channel path with the largest
% delay and the implementation delay of the channel filter, and is required
% to flush the channel filter to obtain the received signal.
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;

% Return DM-RS indices and symbols
dmrsSymbols = nrPDSCHDMRS(carrier,pdsch);
dmrsIndices = nrPDSCHDMRSIndices(carrier,pdsch);

% Create resource grid
grid = nrResourceGrid(carrier,nTxAnts);

% PDSCH DM-RS precoding and mapping
[~,dmrsAntIndices] = nrExtractResources(dmrsIndices,grid);
grid(dmrsAntIndices) = dmrsSymbols;

% OFDM modulation of associated resource elements
txWaveform_original = nrOFDMModulate(carrier,grid);

% Acquire linear interpolator coordinates for neural net preprocessing
[rows,cols] = find(grid ~= 0);
dmrsSubs = [rows, cols, ones(size(cols))];
hest = zeros(size(grid));
[l_hest,k_hest] = meshgrid(1:size(hest,2),1:size(hest,1));

% Preallocate memory for the training data and labels
numExamples = dataSize;
[trainData, trainLabels] = deal(zeros([612 14 2 numExamples]));

% Main loop for data generation, iterating over the number of examples
% specified in the function call. Each iteration of the loop produces a
% new channel realization with a random delay spread, doppler shift,
% and delay profile. Every perturbed version of the transmitted
% waveform with the DM-RS symbols is stored in trainData, and the
% perfect channel realization in trainLabels.
for i = 1:numExamples
    % Release the channel to change nontunable properties
    channel.release

    % Pick a random seed to create different channel realizations
    channel.Seed = randi([1001 2000]);

    % Pick a random delay profile, delay spread, and maximum doppler shift
    channel.DelayProfile = string(delayProfiles(randi([1 numel(delayProfiles)])));
    channel.DelaySpread = randi([1 300])*1e-9;
    channel.MaximumDopplerShift = randi([5 400]);

    % Send data through the channel model. Append zeros at the end of
    % the transmitted waveform to flush channel content. These zeros
    % take into account any delay introduced in the channel, such as
    % multipath delay and implementation delay. This value depends on
    % the sampling rate, delay profile, and delay spread
    txWaveform = [txWaveform_original; zeros(maxChDelay, size(txWaveform_original,2))];
    [rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

    % Add additive white Gaussian noise (AWGN) to the received time-domain
```

```
            % waveform. To take into account sampling rate, normalize the noise power.
            % The SNR is defined per RE for each receive antenna (3GPP TS 38.101-4).
            SNRdB = randi([0 10]);  % Random SNR values between 0 and 10 dB
            SNR = 10^(SNRdB/10);    % Calculate linear SNR
            N0 = 1/sqrt(2.0*nRxAnts*double(waveformInfo.Nfft)*SNR);
            noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
            rxWaveform = rxWaveform + noise;

            % Perfect synchronization. Use information provided by the channel
            % to find the strongest multipath component
            pathFilters = getPathFilters(channel); % Get path filters for perfect channel estimation
            [offset,~] = nrPerfectTimingEstimate(pathGains,pathFilters);

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

            % Perform OFDM demodulation on the received data to recreate the
            % resource grid, including padding in case practical
            % synchronization results in an incomplete slot being demodulated
            rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
            [K,L,R] = size(rxGrid);
            if (L < carrier.SymbolsPerSlot)
                rxGrid = cat(2,rxGrid,zeros(K,carrier.SymbolsPerSlot-L,R));
            end

            % Perfect channel estimation, using the value of the path gains
            % provided by the channel. This channel estimate does not
            % include the effect of transmitter precoding
            estChannelGridPerfect = nrPerfectChannelEstimate(carrier,pathGains, ...
                pathFilters,offset,sampleTimes);

            % Linear interpolation
            dmrsRx = rxGrid(dmrsIndices);
            dmrsEsts = dmrsRx .* conj(dmrsSymbols);
            f = scatteredInterpolant(dmrsSubs(:,2),dmrsSubs(:,1),dmrsEsts);
            hest = f(l_hest,k_hest);

            % Split interpolated grid into real and imaginary components and
            % concatenate them along the third dimension, as well as for the
            % true channel response
            rx_grid = cat(3, real(hest), imag(hest));
            est_grid = cat(3, real(estChannelGridPerfect), ...
                imag(estChannelGridPerfect));

            % Add generated training example and label to the respective arrays
            trainData(:,:,:,i) = rx_grid;
            trainLabels(:,:,:,i) = est_grid;

            % Data generation tracker
            if mod(i,round(numExamples/25)) == 0
                fprintf('%3.2f%% complete\n',i/numExamples*100);
            end
        end
        fprintf('Data generation complete!\n')
    end

    function simParameters = hDeepLearningChanEstSimParameters()
    % Set simulation parameters for Deep Learning Data Synthesis for 5G Channel Estimation example
```

```
    % Carrier configuration
    simParameters.Carrier = nrCarrierConfig;
    simParameters.Carrier.NSizeGrid = 51;            % Bandwidth in number of resource blocks (51
    simParameters.Carrier.SubcarrierSpacing = 30;    % 15, 30, 60, 120, 240 (kHz)
    simParameters.Carrier.CyclicPrefix = 'Normal';   % 'Normal' or 'Extended' (Extended CP is rel
    simParameters.Carrier.NCellID = 2;               % Cell identity

    % Number of transmit and receive antennas
    simParameters.NTxAnts = 1;                      % Number of PDSCH transmission antennas
    simParameters.NRxAnts = 1;                      % Number of UE receive antennas

    % PDSCH and DM-RS configuration
    simParameters.PDSCH = nrPDSCHConfig;
    simParameters.PDSCH.PRBSet = 0:simParameters.Carrier.NSizeGrid-1; % PDSCH PRB allocation
    simParameters.PDSCH.SymbolAllocation = [0, simParameters.Carrier.SymbolsPerSlot];           %
    simParameters.PDSCH.MappingType = 'A';     % PDSCH mapping type ('A'(slot-wise),'B'(non slot-
    simParameters.PDSCH.NID = simParameters.Carrier.NCellID;
    simParameters.PDSCH.RNTI = 1;
    simParameters.PDSCH.VRBToPRBInterleaving = 0; % Disable interleaved resource mapping
    simParameters.PDSCH.NumLayers = 1;            % Number of PDSCH transmission layers
    simParameters.PDSCH.Modulation = '16QAM';                       % 'QPSK', '16QAM', '64QAM',

    % DM-RS configuration
    simParameters.PDSCH.DMRS.DMRSPortSet = 0:simParameters.PDSCH.NumLayers-1; % DM-RS ports to us
    simParameters.PDSCH.DMRS.DMRSTypeAPosition = 2;      % Mapping type A only. First DM-RS symbo
    simParameters.PDSCH.DMRS.DMRSLength = 1;             % Number of front-loaded DM-RS symbols (
    simParameters.PDSCH.DMRS.DMRSAdditionalPosition = 1; % Additional DM-RS symbol positions (max
    simParameters.PDSCH.DMRS.DMRSConfigurationType = 2;  % DM-RS configuration type (1,2)
    simParameters.PDSCH.DMRS.NumCDMGroupsWithoutData = 1;% Number of CDM groups without data
    simParameters.PDSCH.DMRS.NIDNSCID = 1;               % Scrambling identity (0...65535)
    simParameters.PDSCH.DMRS.NSCID = 0;                  % Scrambling initialization (0,1)
end

function plotChEstimates(interpChannelGrid,estChannelGrid,estChannelGridNN,estChannelGridPerfect,
                         interp_mse,practical_mse,neural_mse)
% Plot the different channel estimates and display the measured MSE

    figure;
    cmax = max(abs([estChannelGrid(:); estChannelGridNN(:); estChannelGridPerfect(:)]));

    subplot(1,4,1)
    imagesc(abs(interpChannelGrid));
    xlabel('OFDM Symbol');
    ylabel('Subcarrier');
    title({'Linear Interpolation', ['MSE: ', num2str(interp_mse)]});
    clim([0 cmax]);

    subplot(1,4,2)
    imagesc(abs(estChannelGrid));
    xlabel('OFDM Symbol');
    ylabel('Subcarrier');
    title({'Practical Estimator', ['MSE: ', num2str(practical_mse)]});
    clim([0 cmax]);

    subplot(1,4,3)
    imagesc(abs(estChannelGridNN));
    xlabel('OFDM Symbol');
    ylabel('Subcarrier');
```

```
    title({'Neural Network', ['MSE: ', num2str(neural_mse)]});
    clim([0 cmax]);

    subplot(1,4,4)
    imagesc(abs(estChannelGridPerfect));
    xlabel('OFDM Symbol');
    ylabel('Subcarrier');
    title({'Actual Channel'});
    clim([0 cmax]);

end
```

See Also

Functions

`nrPerfectChannelEstimate` | `nrChannelEstimate` | `predict` | `trainNetwork`

More About

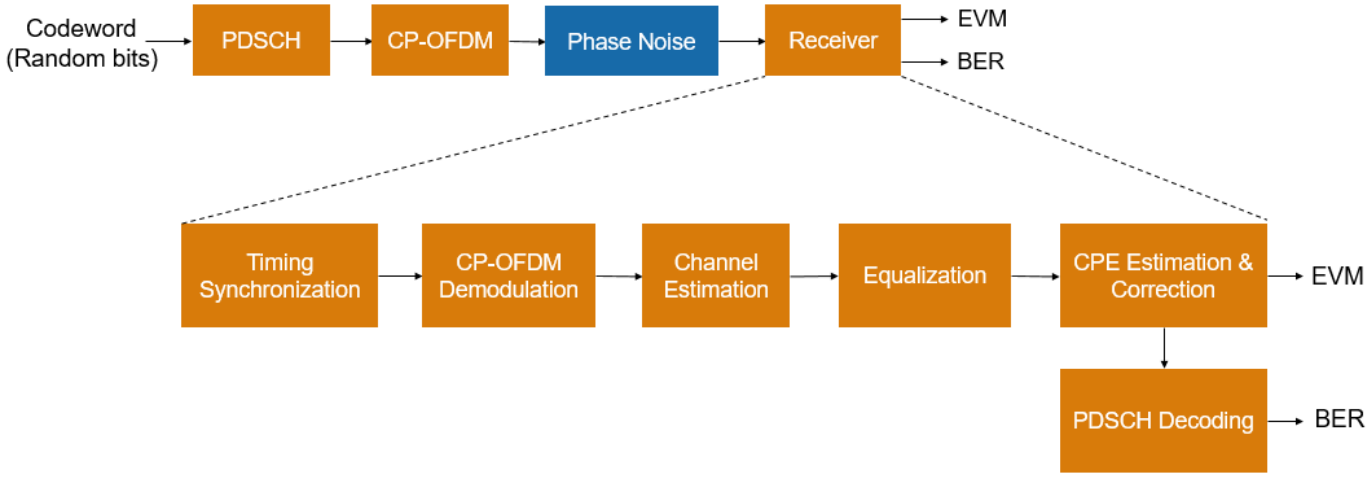
- “SNR Definition Used in Link Simulations” on page 5-86

NR Phase Noise Modeling and Compensation

This example demonstrates the impact of phase noise in a 5G OFDM system and shows how to use phase tracking reference signal (PT-RS) in compensating the common phase error (CPE). The example measures error vector magnitude (EVM) and bit error rate (BER) with and without CPE compensation.

Introduction

In 5G NR, 3GPP introduces a new reference signal, named phase tracking reference signal (PT-RS), to deal with oscillator noise. The noise incurred in the oscillators results in phase modulation of the information signal, leading to significant changes in the frequency spectrum and timing properties of the information signal. This phenomenon related to oscillators is called phase noise. Phase noise produced in local oscillators introduces a significant degradation at millimeter wave (mmWave) frequencies, depending on the power spectral density of phase noise. Phase noise leads to CPE and intercarrier interference (ICI). CPE leads to an identical rotation of a received symbol in each subcarrier. ICI leads to a loss of orthogonality between the subcarriers. Due to the distributed structure of PT-RS in frequency domain, the example primarily uses PT-RS to estimate and minimize the effect of CPE on system performance. The example applies phase noise on the waveform consisting of physical downlink shared channel (PDSCH) and shows the change in EVM and BER without and with CPE compensation. This figure shows the processing chain implemented in this example.

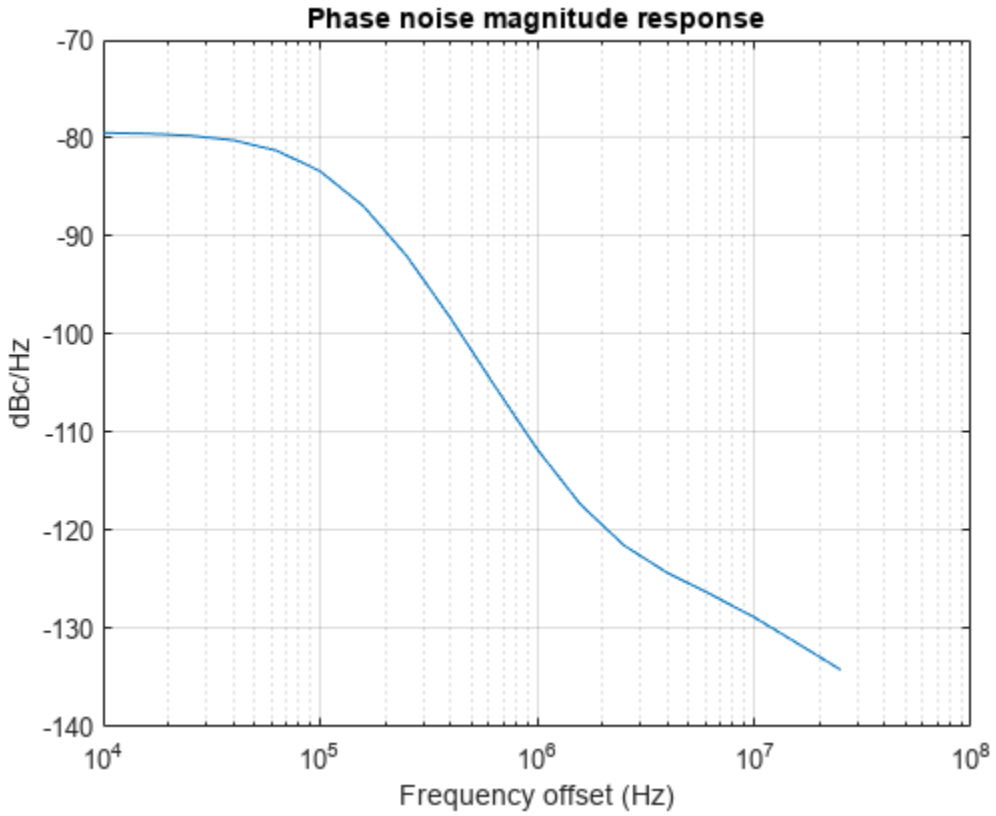


Phase Noise Modeling

The oscillator power spectral density models the phase noise. This example uses a multipole zero model to approximate the power spectral density of the oscillator. Use the `PNModel` field of the `simParameters` structure to select the phase noise model: 'A', 'B', or 'C'. Parameter sets 'A' and 'B' are obtained from practical oscillators operating at 30 GHz and 60 GHz, respectively, as described in TDoc R1-163984. The parameter set 'C' is obtained from the practical oscillator operating at 29.55 GHz, as described in TR 38.803 Section 6.1.10.

The example uses a carrier with a subcarrier spacing of 60 kHz for a transmission bandwidth of 50 MHz.

```
% Configure carrier
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 60;
carrier.CyclicPrefix = 'normal';
carrier.NSizeGrid = 66;

% Set the operating frequency and choose the phase noise model
simParameters = [];
simParameters.Fc = 30e9; % Frequency in Hz
simParameters.PNModel = 'A'; % 'A' (TDoc R1-163984 Set A), 'B' (TDoc R1-163984 Set B), 'C' (TR 38

% Get the sample rate
ofdmInfo = nrOFDMInfo(carrier);
sr = ofdmInfo.SampleRate;

% Phase noise level
foffsetLog = (4:0.2:log10(sr/2)); % Model offset from 1e4 Hz to sr/2 Hz
foffset = 10.^foffsetLog;         % Linear frequency offset
pn_PSD = hPhaseNoisePoleZeroModel(foffset,simParameters.Fc,simParameters.PNModel); % dBc/Hz

% Set phase noise level
pnoise = comm.PhaseNoise('FrequencyOffset',foffset,'Level',pn_PSD,'SampleRate',sr);
pnoise.RandomStream = "mt19937ar with seed";

% Visualize spectrum mask of phase noise
figure
semilogx(foffset,pn_PSD)
xlabel('Frequency offset (Hz)')
ylabel('dBc/Hz')
title('Phase noise magnitude response')
grid on
```



Configure PDSCH

The example configures PDSCH occupying the complete carrier with modulation scheme set to '64QAM' and number of layers set to 1. The example defaults to a single layer and a single codeword of random uncoded bits.

```
% Set PDSCH parameters
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:carrier.NSizeGrid-1;
pdsch.SymbolAllocation = [0 14];
pdsch.Modulation = '64QAM';
pdsch.NumLayers = 1;
pdsch.MappingType = 'A';
pdsch.NID = 1;
pdsch.RNTI = 2;

% Set DM-RS parameters
pdsch.DMRS.DMRSConfigurationType = 1;
pdsch.DMRS.DMRSTypeAPosition = 2;
pdsch.DMRS.DMRSAdditionalPosition = 0;
pdsch.DMRS.DMRSLength = 1;
pdsch.DMRS.DMRSPortSet = [];
pdsch.DMRS.NumCDMGroupsWithoutData = 1;
pdsch.DMRS.NIDNSCID = 1;
pdsch.DMRS.NSCID = 0;

% Set PT-RS parameters
```

```
pdsch.EnablePTRS = 1;
pdsch.PTRS.TimeDensity = 1;
pdsch.PTRS.FrequencyDensity = 2;
pdsch.PTRS.REOffset = '00';
pdsch.PTRS.PTRSPortSet = [];
```

Generate Waveform

The waveform is generated for 2 frames and the field `NumFrames` of `simParameters` structure controls the number of frames of the waveform. The steps involved are:

- Generate random codeword with the bit capacity of PDSCH
- Get the PDSCH symbols for the random codeword and map them to grid
- Generate and map DM-RS symbols to grid
- Generate and map PT-RS symbols to grid
- Perform OFDM modulation for the complete grid of all frames

```
% Number of frames to generate the waveform
simParameters.NumFrames = 2;

% Get the number of slots in the waveform and number of symbols in a slot
numSlots = carrier.SlotsPerFrame*simParameters.NumFrames;
nSlotSymb = carrier.SymbolsPerSlot;

% Initialize the grid for specified number of frames
txGrid = zeros(carrier.NSizeGrid*12,nSlotSymb*numSlots,pdsch.NumLayers);

% Processing loop
txbits = [];
rng('default')
for slotIdx = 0:numSlots - 1
    % Set slot number
    carrier.NSlot = slotIdx;

    % Get PDSCH indices and structural information
    [pdschInd,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);

    % Generate random codeword(s)
    numCW = pdsch.NumCodewords; % Number of codewords
    data = cell(1,numCW);
    for i = 1:numCW
        data{i} = randi([0 1],pdschIndicesInfo.G(i),1);
        txbits = [txbits; data{i}]; %#ok<AGROW>
    end

    % Get modulated symbols
    pdschSym = nrPDSCH(carrier,pdsch,data);

    % Get DM-RS symbols and indices
    dmrsSym = nrPDSCHDMRS(carrier,pdsch);
    dmrsInd = nrPDSCHDMRSIndices(carrier,pdsch);

    % Get PT-RS symbols and indices
    ptrsSym = nrPDSCHPTRS(carrier,pdsch);
    ptrsInd = nrPDSCHPTRSIndices(carrier,pdsch);
```

```
    % Resource element mapping to slot grid
    slotGrid = nrResourceGrid(carrier,pdsch.NumLayers);
    slotGrid(pdschInd) = pdschSym;
    slotGrid(dmrsInd) = dmrsSym;
    slotGrid(ptrsInd) = ptrsSym;

    % Generate txGrid for all frames by mapping slotGrid at respective
    % locations
    txGrid(:,slotIdx*nSlotSymb+1:(slotIdx+1)*(nSlotSymb),:) = slotGrid;
end

% Perform OFDM modulation
carrier.NSlot = 0; % Reset the slot number to 0 for OFDM modulation
txWaveform = nrOFDMModulate(carrier,txGrid);
```

Apply Phase Noise

Apply phase noise to the transmitted waveform. To clearly observe the impact of phase noise, the example does not apply any thermal noise or channel model in addition to phase noise. The example applies the same phase noise to all the layers.

```
rxWaveform = pnoise(txWaveform);
```

Receiver

Before returning the equalized PDSCH symbols and decoded bits, the receiver performs these steps:

- Timing synchronization
- OFDM demodulation
- Channel estimation
- Equalization
- CPE estimation and correction
- PDSCH decoding

For the CPE estimation and correction step, the receiver uses the logical field `CompensateCPE` of the `simParameters` structure. Because the example does not use a propagation channel, the steps of timing synchronization, channel estimation, and equalization are not strictly necessary. However, these steps help investigate the phase noise effects if you introduce a channel.

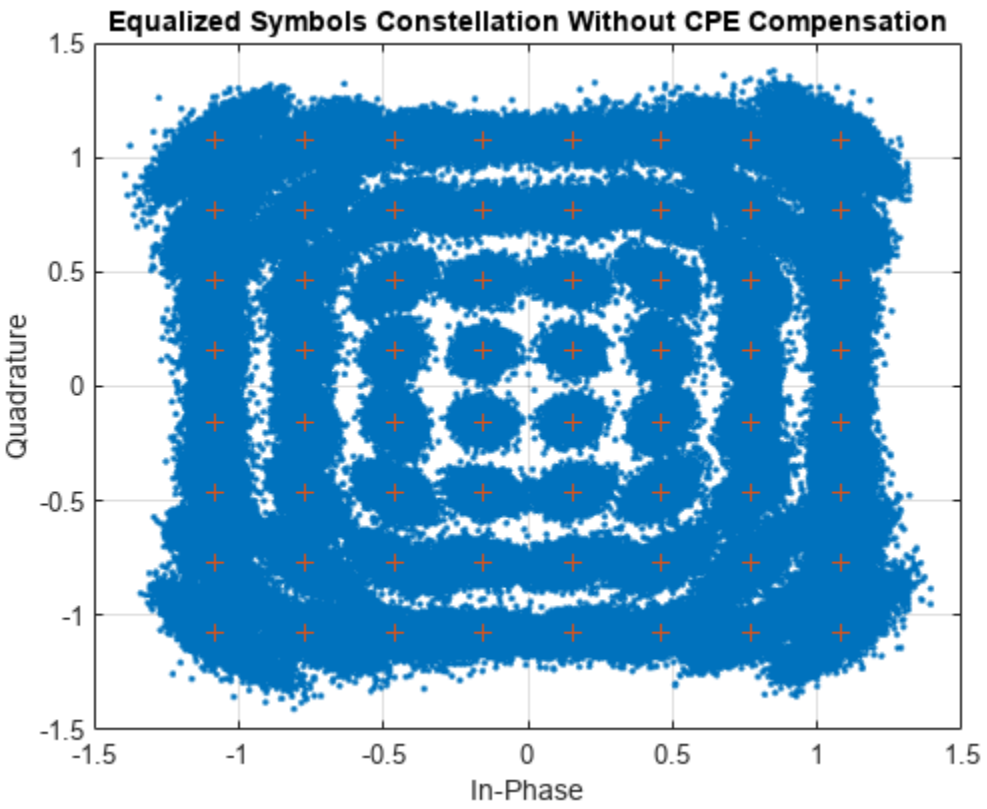
The example shows the equalized constellation symbols, EVM, and bit error rate, with and without CPE compensation.

Case 1: Without CPE Compensation

To disable the CPE compensation, set the field `CompensateCPE` of `simParameters` structure to 0.

```
simParameters.CompensateCPE = 0;
[eqSymbols,rxbits] = practicalReceiver(carrier,pdsch,simParameters,rxWaveform);
refSymbols = getConstellationPoints(pdsch);
% Display the constellation diagram
figure
plot(eqSymbols,'.')
hold on
plot(refSymbols,'+')
title('Equalized Symbols Constellation Without CPE Compensation')
```

```
grid on
xlabel('In-Phase')
ylabel('Quadrature')
```



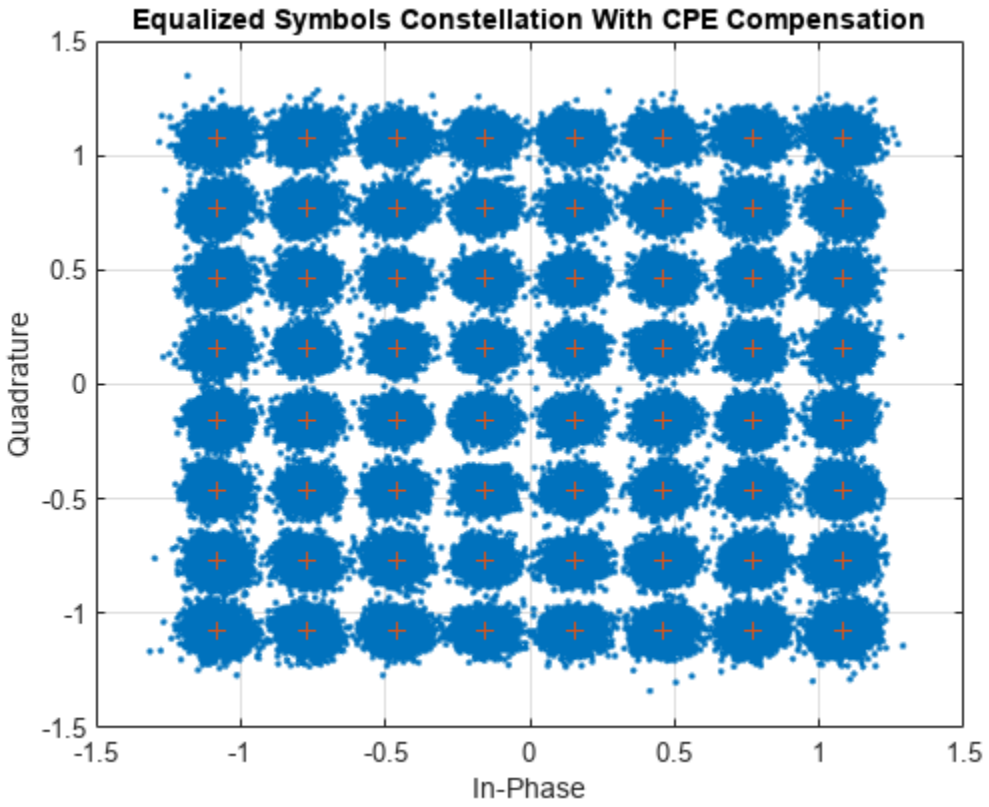

```
% Display RMS EVM
evm = comm.EVM('ReferenceSignalSource','Estimated from reference constellation','ReferenceConste
fprintf('RMS EVM (in percent) for equalized symbols without CPE compensation: %f%% \n',evm(eqSyml

RMS EVM (in percent) for equalized symbols without CPE compensation: 7.431048%

% Display bit error rate
errorRate = nnz(rxbits-txbits)/numel(txbits);
fprintf('Bit error rate without CPE compensation: %f \n',errorRate)

Bit error rate without CPE compensation: 0.005525
```

Case 2: With CPE Compensation

To enable the CPE compensation, set the field `CompensateCPE` of `simParameters` structure to 1. Use PT-RS to estimate the CPE at all the OFDM symbol locations in a slot. Correct the CPE in the OFDM symbol locations within the range of PT-RS OFDM symbols.

```
simParameters.CompensateCPE = 1;
[eqSymbolsCPE,rxbitsCPE] = practicalReceiver(carrier,pdsch,simParameters,rxWaveform);
% Display the constellation diagram
figure
plot(eqSymbolsCPE,'.')
hold on
```

```
plot(refSymbols,'+')
title('Equalized Symbols Constellation With CPE Compensation')
grid on
xlabel('In-Phase')
ylabel('Quadrature')
```




```
% Display RMS EVM
fprintf('RMS EVM (in percent) for equalized symbols with CPE compensation: %f%% \n',evm(eqSymbols

RMS EVM (in percent) for equalized symbols with CPE compensation: 4.557690%

% Display bit error rate
errorRateCPE = nnz(rxbitsCPE-txbits)/numel(txbits);
fprintf('Bit error rate with CPE compensation: %f \n',errorRateCPE)

Bit error rate with CPE compensation: 0.000052
```

Further Exploration

- To visualize the impact of phase noise, change the carrier frequency, subcarrier spacing, number of resource blocks, modulation scheme, and the number of frames.
- To see the effects of phase noise on the constellation, change the phase noise model.
- To analyze the effect of CPE compensation with different configurations, change the time and frequency density of PT-RS.
- Visualize the impacts of phase noise by including thermal noise and channel models.

Summary

This example demonstrates the impact of phase noise and shows how to estimate and correct CPE with PT-RS. The example also shows that CPE compensation reduces the EVM and improves bit error rate. The displayed constellation plot shows huge ICI in mmWave frequencies, indicating that ICI compensation needs to be performed in addition to CPE compensation.

Local Functions

```
function [eqSymbols,rxbits] = practicalReceiver(carrier,pdsch,params,rxWaveform)
% Returns equalized modulated symbols after performing the timing
% estimation, OFDM demodulation, channel estimation, MMSE equalization,
% CPE estimation and correction, and PDSCH decoding.

    % Get the current slot number, number of slots, number of symbols
    % per slot, and total number of symbols
    nSlot = carrier.NSlot;
    numSlots = carrier.SlotsPerFrame*params.NumFrames;
    nSlotSymb = carrier.SymbolsPerSlot;
    numTotalSymbols = numSlots*nSlotSymb;

    % Get reference grid with DM-RS symbols
    dmrsSymCell = cell(1,numSlots);
    dmrsIndCell = cell(1,numSlots);
    refGrid = zeros(carrier.NSizeGrid*12,numTotalSymbols,pdsch.NumLayers);
    for NSlot = 0:numSlots-1
        carrier.NSlot = NSlot;
        slotGrid = nrResourceGrid(carrier,pdsch.NumLayers);
        dmrsSymCell{NSlot+1} = nrPDSCHDMRS(carrier,pdsch);
        dmrsIndCell{NSlot+1} = nrPDSCHDMRSIndices(carrier,pdsch);
        slotGrid(dmrsIndCell{NSlot+1}) = dmrsSymCell{NSlot+1};
        refGrid(:,NSlot*nSlotSymb+1:(NSlot+1)*(nSlotSymb),:) = slotGrid;
    end

    % Perform timing estimation and correction
    carrier.NSlot = nSlot;
    offset = nrTimingEstimate(carrier,rxWaveform,refGrid);
    waveformSync = rxWaveform(1+offset:end,:);

    % Perform OFDM demodulation on the received data to recreate the
    % resource grid, including padding in the event that practical
    % synchronization results in an incomplete slots being demodulated
    rxGrid = nrOFDMDemodulate(carrier,waveformSync);
    [K,L,R] = size(rxGrid);
    if (L < numTotalSymbols)
        rxGrid = cat(2,rxGrid,zeros(K,numTotalSymbols-L,R));
    end

    % Declare storage variables
    eqSymbols = [];  % equalized symbols for constellation plot
    rxbits = [];

    for NSlot = 0:numSlots-1
        % Extract grid for current slot
        currentGrid = rxGrid(:,NSlot*nSlotSymb+(1:nSlotSymb),:);

        % Get the PDSCH resources
        carrier.NSlot = NSlot;
```

```
dmrsSymbols = dmrsSymCell{NSlot+1};
dmrsIndices = dmrsIndCell{NSlot+1};
ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);
ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
[pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(carrier,pdsch);

% Channel estimation
[estChannelGrid,noiseEst] = nrChannelEstimate(currentGrid,dmrsIndices,dmrsSymbols,"CDMLen

% Get PDSCH resource elements from the received grid
[pdschRx,pdschHest] = nrExtractResources(pdschIndices,currentGrid,estChannelGrid);

% Equalization
pdschEq = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

% Common phase error (CPE) estimation and correction
if params.CompensateCPE
    % Initialize temporary grid to store equalized symbols
    tempGrid = nrResourceGrid(carrier,pdsch.NumLayers);

    % Extract PT-RS symbols from received grid and estimated
    % channel grid
    [ptrsRx,ptrsHest,~,~,~,ptrsLayerIndices] = nrExtractResources(ptrsIndices,currentGrid

    % Equalize PT-RS symbols and map them to tempGrid
    ptrsEq = nrEqualizeMMSE(ptrsRx,ptrsHest,noiseEst);
    tempGrid(ptrsLayerIndices) = ptrsEq;

    % Estimate the residual channel at the PT-RS locations in
    % tempGrid
    cpe = nrChannelEstimate(tempGrid,ptrsIndices,ptrsSymbols);

    % Sum estimates across subcarriers, receive antennas, and
    % layers. Then, get the CPE by taking the angle of the
    % resultant sum
    cpe = angle(sum(cpe,[1 3 4]));

    % Map the equalized PDSCH symbols to tempGrid
    tempGrid(pdschIndices) = pdschEq;

    % Correct CPE in each OFDM symbol within the range of reference
    % PT-RS OFDM symbols
    if numel(pdschIndicesInfo.PTRSSymbolSet) > 0
        symLoc = pdschIndicesInfo.PTRSSymbolSet(1)+1:pdschIndicesInfo.PTRSSymbolSet(end)-
        tempGrid(:,symLoc,:) = tempGrid(:,symLoc,:).*exp(-1i*cpe(symLoc));
    end

    % Extract PDSCH symbols
    pdschEq = tempGrid(pdschIndices);
end

% Store the equalized symbols and output them for all the slots
eqSymbols = [eqSymbols; pdschEq]; %#ok<AGROW>

% Decode the PDSCH symbols and get the hard bits
eqbits = nrPDSCHDecode(carrier,pdsch,pdschEq);
for i = 1:numel(eqbits)
    rxbits = [rxbits; double(eqbits{i}<0)]; %#ok<AGROW>
```

```
        end

    end

end

function sym = getConstellationPoints(pdsch)
%getConstellationPoints Constellation points
%   SYM = getConstellationPoints(PDSCH) returns the constellation points
%   SYM based on modulation schemes provided in PDSCH configuration object.

    sym = [];
    modulation = string(pdsch.Modulation);  % Convert modulation scheme to string type
    ncw = pdsch.NumCodewords;               % Number of codewords
    if ncw == 2 && (numel(modulation) == 1)
        modulation(end+1) = modulation(1);
    end
    % Get the constellation points
    for cwIndex = 1:ncw
        qm = strcmpi(modulation(cwIndex),{'QPSK','16QAM','64QAM','256QAM'})*[2 4 6 8]';
        sym = [sym; nrSymbolModulate(int2bit((0:2^qm-1)',qm),modulation(cwIndex))]; %#ok<AGROW>
    end

end
```

See Also

More About

- “EVM Measurement of 5G NR Downlink Waveforms with RF Impairments” on page 7-79

Neural Network for Beam Selection

This example shows how to use a neural network to reduce the overhead in the beam selection task. In the example, you use only the location of the receiver rather than knowledge of the communication channels. Instead of an exhaustive beam search over all the beam pairs, you can reduce beam sweeping overhead by searching among the selected *K* beam pairs. Considering a system with a total of 16 beam pairs, simulation results in this example show the designed machine learning algorithm can achieve an accuracy of 90% by performing an exhaustive search over only half of the beam pairs.

Introduction

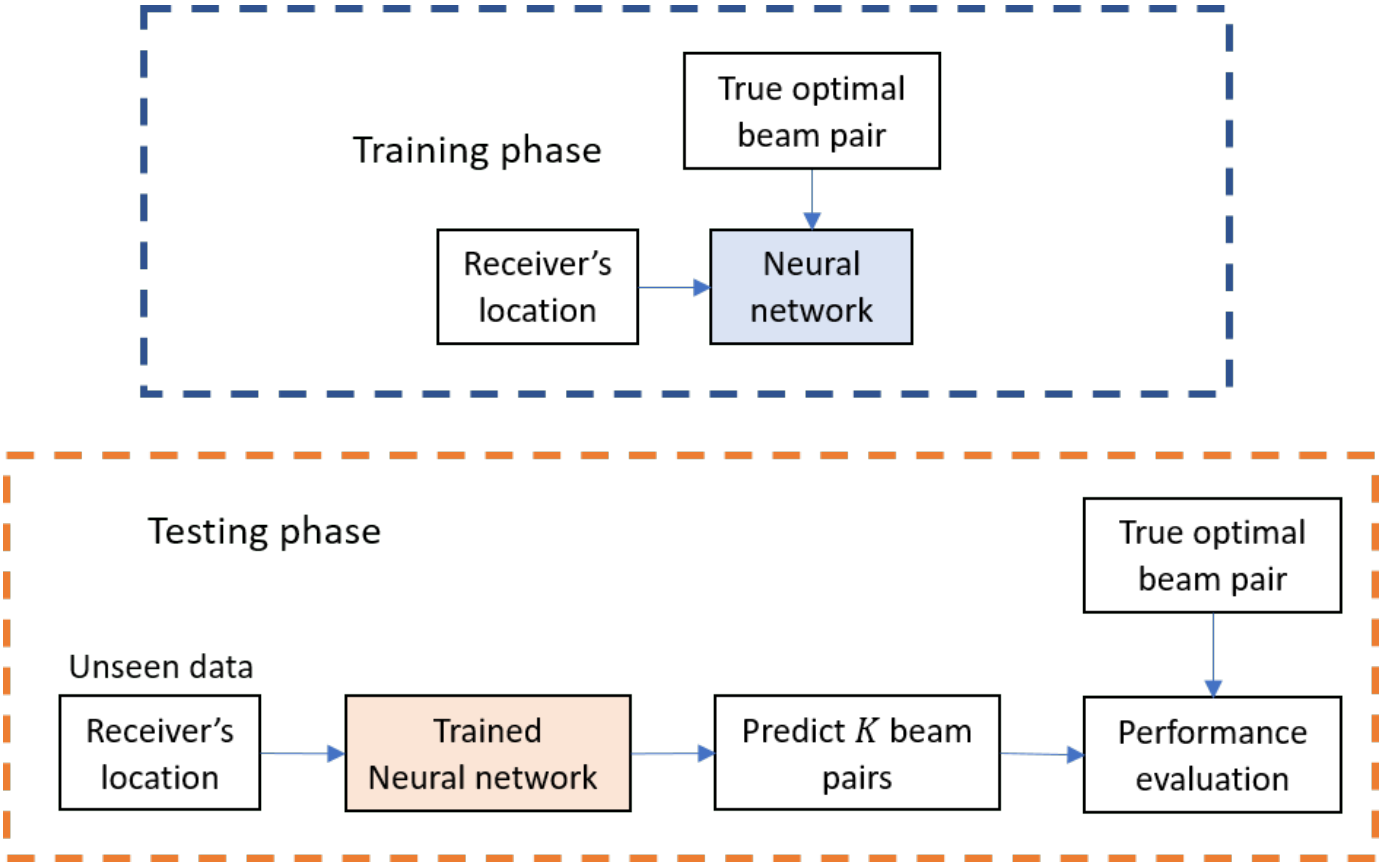
To enable millimeter wave (mmWave) communications, beam management techniques must be used due to the high pathloss and blockage experienced at high frequencies. Beam management is a set of Layer 1 (physical layer) and Layer 2 (medium access control) procedures to establish and retain an optimal beam pair (transmit beam and a corresponding receive beam) for good connectivity [1 on page 4-46]. For simulations of 5G New Radio (NR) beam management procedures, see the “NR SSB Beam Sweeping” on page 1-98 and “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113 examples.

This example considers beam selection procedures when a connection is established between the user equipment (UE) and access network node (gNB). In 5G NR, the beam selection procedure for initial access consists of beam sweeping, which requires exhaustive searches over all the beams on the transmitter and the receiver sides, and then selection of the beam pair offering the strongest reference signal received power (RSRP). Since mmWave communications require many antenna elements, implying many beams, an exhaustive search over all beams becomes computationally expensive and increases the initial access time.

To avoid repeatedly performing an exhaustive search and to reduce the communication overhead, machine learning has been applied to the beam selection problem. Typically, the beam selection problem is posed as a classification task, where the target output is the best beam pair index. The extrinsic information, including lidar, GPS signals, and roadside camera images, is used as input to the machine learning algorithms [2 on page 4-46]-[6 on page 4-46]. Specifically, given this out-of-band information, a trained machine learning model recommends a set of *K* good beam pairs. Instead of an exhaustive search over all the beam pairs, the simulation reduces beam sweeping overhead by searching only among the selected *K* beam pairs.

This example uses a neural network to perform beam selection using only the GPS coordinates of the receiver. Fixing the locations of the transmitter and the scatterers, the example generates a set of training samples: Each sample consists of a receiver location (GPS data) and the true optimal beam pair index (found by performing exhaustive search over all the beam pairs at transmit and receive ends). The example designs and trains a neural network that uses the location of the receiver as the input and the true optimal beam pair index as the correct label. During the testing phase, the neural network first outputs *K* good beam pairs. An exhaustive search over these *K* beam pairs is followed, and the beam pair with the highest average RSRP is selected as the final predicted beam pair by the neural network.

The example measures the effectiveness of the proposed method using two metrics: average RSRP and top-*K* accuracy [2 on page 4-46]-[6 on page 4-46]. This figure shows the main processing steps.




```
rng(211);                           % Set RNG state for repeatability
```

Generate Training Data

In the prerecorded data, receivers are randomly distributed on the perimeter of a 6-meter square and configured with 16 beam pairs (four beams on each end, analog beamformed with 1 RF chain). After setting up a MIMO scattering channel, the example considers 200 different receiver locations in the training set and 100 different receiver locations in the test sets. The prerecorded data uses 2-D location coordinates. Specifically, the third GPS coordinate of each sample is always zero. As in the `NR SSB Beam Sweeping` example, for each location, SSB-based beam sweeping is performed for an exhaustive search over all 16 beam pairs. Since AWGN is added during the exhaustive search, for each location, the example runs four different trials and determines the true optimal beam pair by picking the beam pair with the highest average RSRP.

To generate new training and test sets, you can adjust the `useSavedData` and `SaveData` logicals. Be aware that regenerating data takes a significant amount of time.

```
useSavedData = true;
saveData = false;

if useSavedData
    load nnBS_prm.mat;              % Load beam selection system parameters
    load nnBS_TrainingData.mat;     % Load prerecorded training samples
    %   (input: receiver's location; output: optimal beam pair indices)
    load nnBS_TestData.mat;         % Load prerecorded test samples
else
```

Configure Frequency and Beam Sweeping Angles

```
prm.NCellID = 1;                    % Cell ID
prm.FreqRange = 'FR1';              % Frequency range: 'FR1' or 'FR2'

prm.CenterFreq = 2.5e9;             % Hz
prm.SSBlockPattern = 'Case B';      % Case A/B/C/D/E
prm.SSBTransmitted = [ones(1,4) zeros(1,0)]; % 4/8 or 64 in length

prm.TxArraySize = [8 8];            % Transmit array size, [rows cols]
prm.TxAZlim = [-163 177];           % Transmit azimuthal sweep limits
prm.TxELlim = [-90 0];              % Transmit elevation sweep limits

prm.RxArraySize = [2 2];            % Receive array size, [rows cols]
prm.RxAZlim = [-177 157];           % Receive azimuthal sweep limits
prm.RxELlim = [0 90];               % Receive elevation sweep limits

prm.ElevationSweep = false;         % Enable/disable elevation sweep
prm.SNRdB = 30;                     % SNR, dB
prm.RSRPMode = 'SSSwDMRS';          % {'SSSwDMRS', 'SSSonly'}

prm = validateParams(prm);
```

Synchronization Signal Burst Configuration

```
txBurst = nrWavegenSSBurstConfig;
txBurst.BlockPattern = prm.SSBlockPattern;
txBurst.TransmittedBlocks = prm.SSBTransmitted;
txBurst.Period = 20;
txBurst.SubcarrierSpacingCommon = prm.SubcarrierSpacingCommon;
```

Scatterer Configuration

```
c = physconst('LightSpeed');   % Propagation speed
prm.lambda = c/prm.CenterFreq; % Wavelength

prm.rightCoorMax = 10;    % Maximum x-coordinate
prm.topCoorMax = 10;      % Maximum y-coordinate
prm.posTx = [3.5;4.2;0];  % Transmit array position, [x;y;z], meters

% Scatterer locations
% Generate scatterers at random positions
Nscat = 10;        % Number of scatterers
azRange = prm.TxAZlim(1):prm.TxAZlim(2);
elRange = -90:90;

% More evenly spaced scatterers
randAzOrder = round(linspace(1, length(azRange), Nscat));
azAngInSph = azRange(randAzOrder(1:Nscat));

% Consider a 2-D area, i.e., the elevation angle is zero
elAngInSph = zeros(size(azAngInSph));
r = 2;            % radius
[x,y,z] = sph2cart(deg2rad(azAngInSph),deg2rad(elAngInSph),r);
prm.ScatPos = [x;y;z] + [prm.rightCoorMax/2;prm.topCoorMax/2;0];
```

Antenna Array Configuration

```
% Transmit array
if prm.IsTxURA
    % Uniform rectangular array
    arrayTx = phased.URA(prm.TxArraySize,0.5*prm.lambda, ...
        'Element',phased.IsotropicAntennaElement('BackBaffled',true));
else
    % Uniform linear array
    arrayTx = phased.ULA(prm.NumTx, ...
        'ElementSpacing',0.5*prm.lambda, ...
        'Element',phased.IsotropicAntennaElement('BackBaffled',true));
end

% Receive array
if prm.IsRxURA
    % Uniform rectangular array
    arrayRx = phased.URA(prm.RxArraySize,0.5*prm.lambda, ...
        'Element',phased.IsotropicAntennaElement);
else
    % Uniform linear array
    arrayRx = phased.ULA(prm.NumRx, ...
        'ElementSpacing',0.5*prm.lambda, ...
        'Element',phased.IsotropicAntennaElement);
end
```

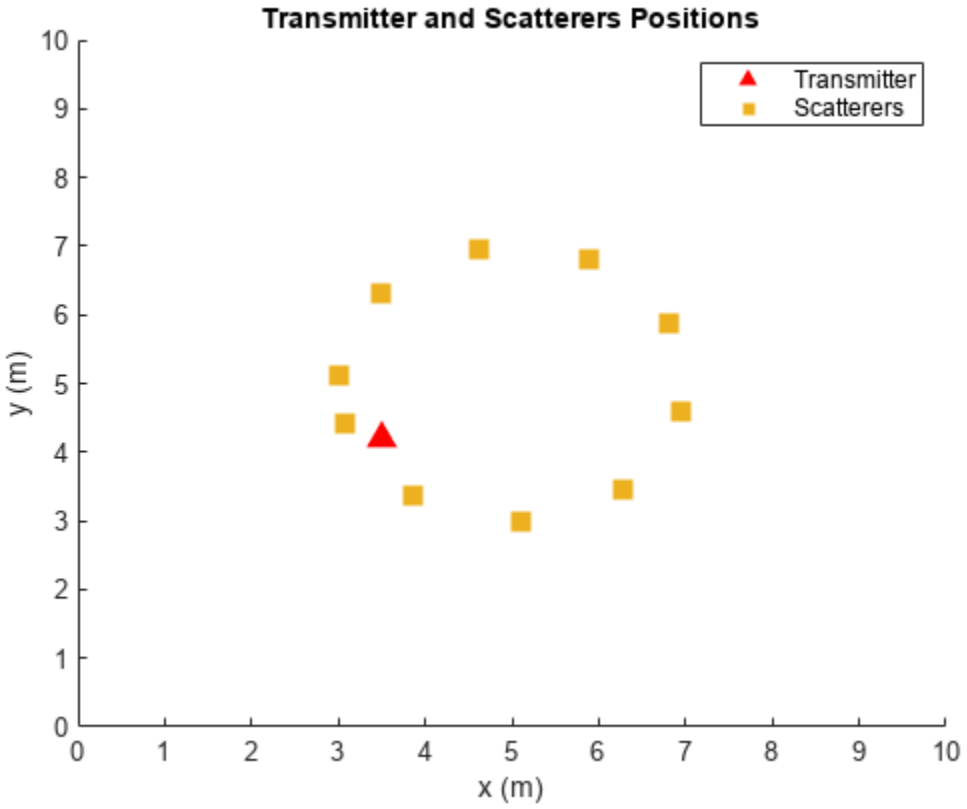
Determine Tx/Rx Positions

```
% Receiver locations
% Training data: X points around a rectangle: each side has X/4 random points
% X: X/4 for around square, X/10 for validation => lcm(4,10) = 20 smallest
NDiffLocTrain = 200;
pointsEachSideTrain = NDiffLocTrain/4;
prm.NDiffLocTrain = NDiffLocTrain;

locationX = 2*ones(pointsEachSideTrain, 1);
locationY = 2 + (8-2)*rand(pointsEachSideTrain, 1);

locationX = [locationX; 2 + (8-2)*rand(pointsEachSideTrain, 1)];
locationY = [locationY; 8*ones(pointsEachSideTrain, 1)];

locationX = [locationX; 8*ones(pointsEachSideTrain, 1)];
locationY = [locationY; 2 + (8-2)*rand(pointsEachSideTrain, 1)];

locationX = [locationX; 2 + (8-2)*rand(pointsEachSideTrain, 1)];
locationY = [locationY; 2*ones(pointsEachSideTrain, 1)];

locationZ = zeros(size(locationX));
locationMat = [locationX locationY locationZ];

% Fixing receiver's location, run repeated simulations to consider
% different realizations of AWGN
prm.NRepeatSameLoc = 4;

locationMatTrain = repelem(locationMat,prm.NRepeatSameLoc, 1);

% Test data: Y points around a rectangle: each side has Y/4 random points
% Different data than test, but a smaller number
```

```
    NDiffLocTest = 100;
    pointsEachSideTest = NDiffLocTest/4;
    prm.NDiffLocTest = NDiffLocTest;

    locationX = 2*ones(pointsEachSideTest, 1);
    locationY = 2 + (8-2)*rand(pointsEachSideTest, 1);

    locationX = [locationX; 2 + (8-2)*rand(pointsEachSideTest, 1)];
    locationY = [locationY; 8*ones(pointsEachSideTest, 1)];

    locationX = [locationX; 8*ones(pointsEachSideTest, 1)];
    locationY = [locationY; 2 + (8-2)*rand(pointsEachSideTest, 1)];

    locationX = [locationX; 2 + (8-2)*rand(pointsEachSideTest, 1)];
    locationY = [locationY; 2*ones(pointsEachSideTest, 1)];

    locationZ = zeros(size(locationX));
    locationMat = [locationX locationY locationZ];

    locationMatTest = repelem(locationMat,prm.NRepeatSameLoc,1);

    [optBeamPairIdxMatTrain,rsrpMatTrain] = hGenDataMIMOScatterChan('training',locationMatTrain,p
    [optBeamPairIdxMatTest,rsrpMatTest] = hGenDataMIMOScatterChan('test',locationMatTest,prm,txBu

    % Save generated data
    if saveData
        save('nnBS_prm.mat','prm');
        save('nnBS_TrainingData.mat','optBeamPairIdxMatTrain','rsrpMatTrain','locationMatTrain')
        save('nnBS_TestData.mat','optBeamPairIdxMatTest','rsrpMatTest','locationMatTest');
    end
end
```

Plot Transmitter and Scatterer Locations

```
figure
scatter(prm.posTx(1),prm.posTx(2),100,'r^','filled');
hold on;
scatter(prm.ScatPos(1,:),prm.ScatPos(2,:),100,[0.9290 0.6940 0.1250],'s','filled');
xlim([0 10])
ylim([0 10])
title('Transmitter and Scatterers Positions')
legend('Transmitter','Scatterers')
xlabel('x (m)')
ylabel('y (m)')
```



Data Processing and Visualization

Next, label the beam pair with the highest average RSRP as the true optimal beam pair. Convert one-hot encoding labels to categorical data to use for classification. Finally, augment the categorical data so that it has 16 classes total to match the possible number of beam pairs (although classes may have unequal number of elements). The augmentation is to ensure that the output of the neural network has the desired dimension 16.

Process Training Data

```
% Choose the best beam pair by picking the one with the highest average RSRP
% (taking average over NRepeatSameLoc different trials at each location)
avgOptBeamPairIdxCellTrain = cell(size(optBeamPairIdxMatTrain, 1)/prm.NRepeatSameLoc, 1);
avgOptBeamPairIdxScalarTrain = zeros(size(optBeamPairIdxMatTrain, 1)/prm.NRepeatSameLoc, 1);
for locIdx = 1:size(optBeamPairIdxMatTrain, 1)/prm.NRepeatSameLoc
    avgRsrp = squeeze(rsrpMatTrain(:,:,locIdx));
    [~, targetBeamIdx] = max(avgRsrp(:));
    avgOptBeamPairIdxScalarTrain(locIdx) = targetBeamIdx;
    avgOptBeamPairIdxCellTrain{locIdx} = num2str(targetBeamIdx);
end

% Even though there are a total of 16 beam pairs, due to the fixed topology
% (transmitter/scatterers/receiver locations), it is possible
% that some beam pairs are never selected as an optimal beam pair
%
% Therefore, we augment the categories so 16 classes total are in the data
% (although some classes may have zero elements)
```

```
allBeamPairIdxCell = cellstr(string((1:prm.numBeams^2)'));
avgOptBeamPairIdxCellTrain = categorical(avgOptBeamPairIdxCellTrain, allBeamPairIdxCell);
NBeamPairInTrainData = numel(categories(avgOptBeamPairIdxCellTrain)); % Should be 16
```

Process Testing Data

```
% Decide the best beam pair by picking the one with the highest avg. RSRP
avgOptBeamPairIdxCellTest = cell(size(optBeamPairIdxMatTest, 1)/prm.NRepeatSameLoc, 1);
avgOptBeamPairIdxScalarTest = zeros(size(optBeamPairIdxMatTest, 1)/prm.NRepeatSameLoc, 1);
for locIdx = 1:size(optBeamPairIdxMatTest, 1)/prm.NRepeatSameLoc
    avgRsrp = squeeze(rsrpMatTest(:,:,locIdx));
    [~, targetBeamIdx] = max(avgRsrp(:));
    avgOptBeamPairIdxScalarTest(locIdx) = targetBeamIdx;
    avgOptBeamPairIdxCellTest{locIdx} = num2str(targetBeamIdx);
end
% Augment the categories such that the data has 16 classes total
avgOptBeamPairIdxCellTest = categorical(avgOptBeamPairIdxCellTest, allBeamPairIdxCell);
NBeamPairInTestData = numel(categories(avgOptBeamPairIdxCellTest)); % Should be 16
```

Create Input/Output Data for Neural Network

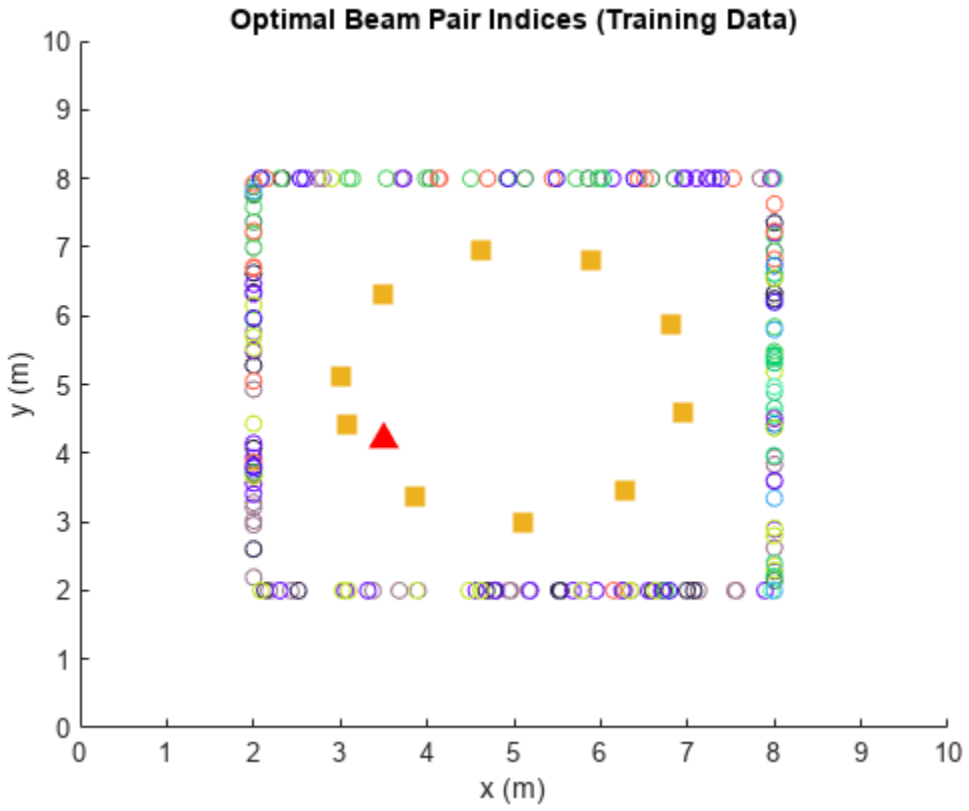
```
trainDataLen = size(locationMatTrain, 1)/prm.NRepeatSameLoc;
trainOut = avgOptBeamPairIdxCellTrain;
sampledLocMatTrain = locationMatTrain(1:prm.NRepeatSameLoc:end, :);
trainInput = sampledLocMatTrain(1:trainDataLen, :);

% Take 10% data out of test data as validation data
valTestDataLen = size(locationMatTest, 1)/prm.NRepeatSameLoc;
valDataLen = round(0.1*size(locationMatTest, 1))/prm.NRepeatSameLoc;
testDataLen = valTestDataLen-valDataLen;

% Randomly shuffle the test data such that the distribution of the
% extracted validation data is closer to test data
rng(111)
shuffledIdx = randperm(prm.NDiffLocTest);
avgOptBeamPairIdxCellTest = avgOptBeamPairIdxCellTest(shuffledIdx);
avgOptBeamPairIdxScalarTest = avgOptBeamPairIdxScalarTest(shuffledIdx);
rsrpMatTest = rsrpMatTest(:,:,shuffledIdx);

valOut = avgOptBeamPairIdxCellTest(1:valDataLen, :);
testOutCat = avgOptBeamPairIdxCellTest(1+valDataLen:end, :);

sampledLocMatTest = locationMatTest(1:prm.NRepeatSameLoc:end, :);
sampledLocMatTest = sampledLocMatTest(shuffledIdx, :);

valInput = sampledLocMatTest(1:valDataLen, :);
testInput = sampledLocMatTest(valDataLen+1:end, :);
```

Plot Optimal Beam Pair Distribution for Training Data

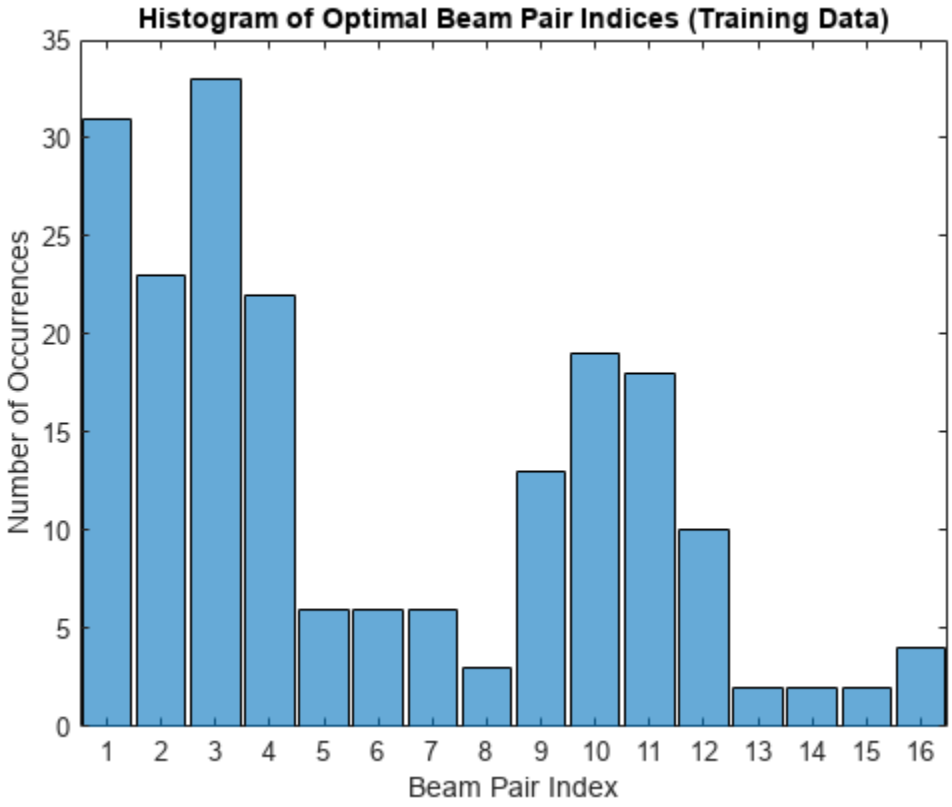
Plot the location and the optimal beam pair for each training sample (200 in total). Each color represents one beam pair index. In other words, the data points with the same color belong to the same class. Increase the training data set to possibly include each beam pair value, though the actual distribution of the beam pairs would depend on the scatterer and transmitter locations.

```
figure
rng(111)    % for colors in plot
color = rand(NBeamPairInTrainData, 3);
```

```
uniqueOptBeamPairIdx = unique(avgOptBeamPairIdxScalarTrain);
for n = 1:length(uniqueOptBeamPairIdx)
    beamPairIdx = find(avgOptBeamPairIdxScalarTrain == uniqueOptBeamPairIdx(n));
    locX = sampledLocMatTrain(beamPairIdx, 1);
    locY = sampledLocMatTrain(beamPairIdx, 2);
    scatter(locX, locY, [], color(n, :));
    hold on;
end
scatter(prm.posTx(1),prm.posTx(2),100,'r^','filled');
scatter(prm.ScatPos(1,:),prm.ScatPos(2,:),100,[0.9290 0.6940 0.1250],'s','filled');
hold off
xlabel('x (m)')
ylabel('y (m)')
xlim([0 10])
ylim([0 10])
title('Optimal Beam Pair Indices (Training Data)')
```

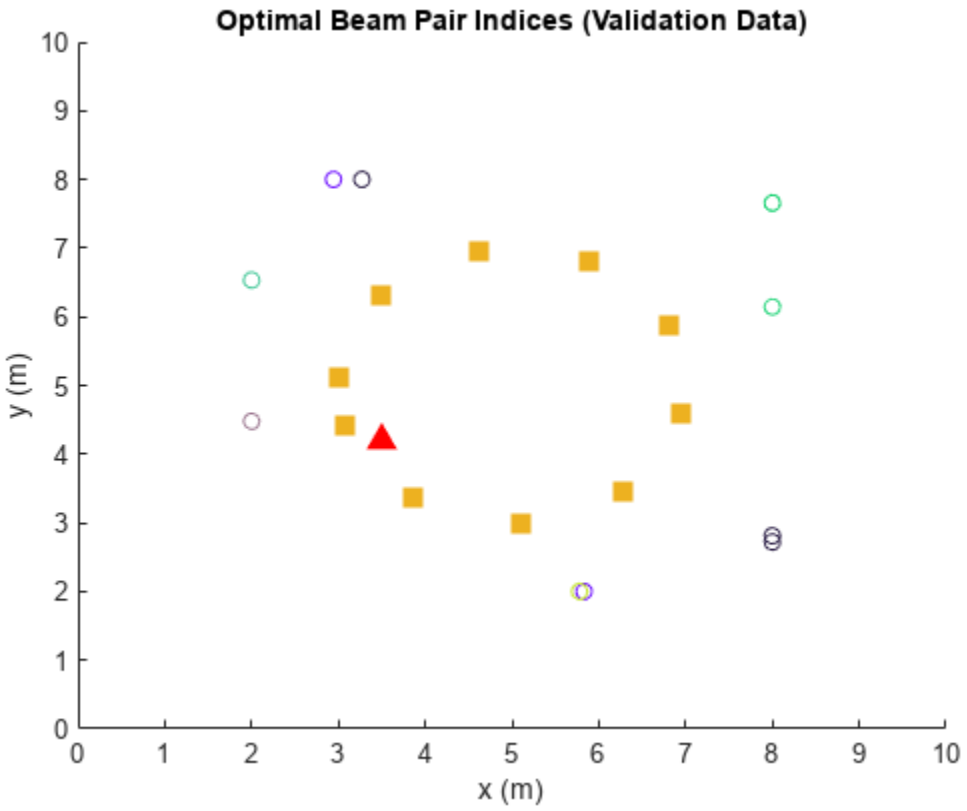



```
figure
histogram(trainOut)
title('Histogram of Optimal Beam Pair Indices (Training Data)')
xlabel('Beam Pair Index')
ylabel('Number of Occurrences')
```

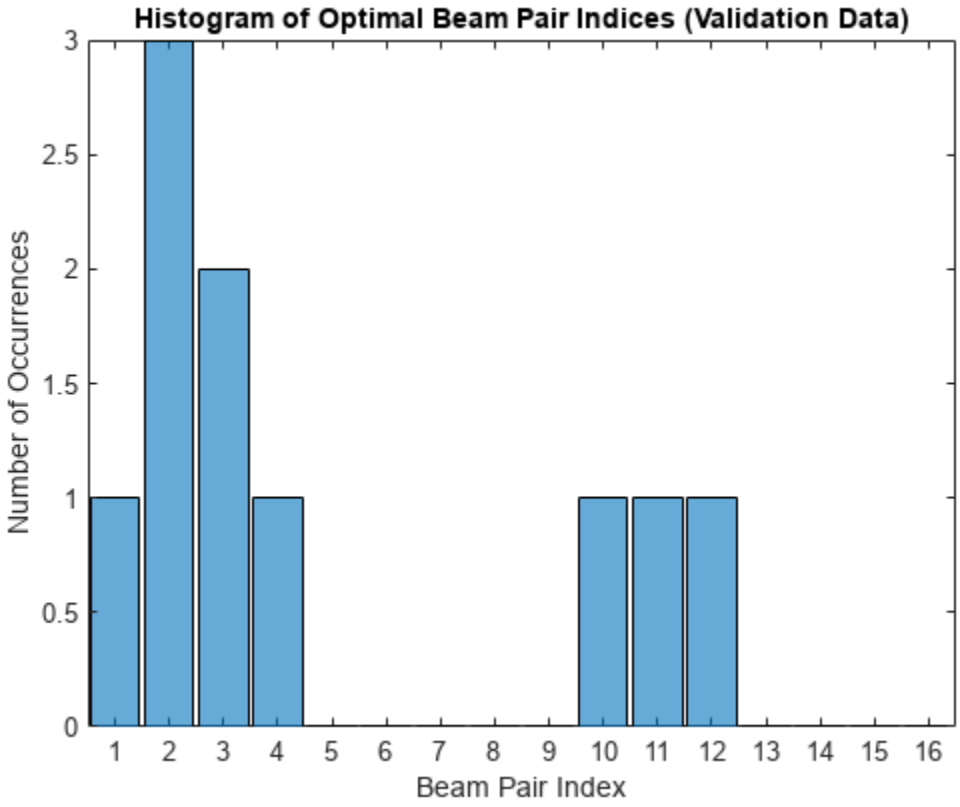


Plot Optimal Beam Pair Distribution for Validation Data

```
figure
rng(111)    % for colors in plot
color = rand(NBeamPairInTestData, 3);
uniqueOptBeamPairIdx = unique(avgOptBeamPairIdxScalarTest(1:valDataLen));
for n = 1:length(uniqueOptBeamPairIdx)
    beamPairIdx = find(avgOptBeamPairIdxScalarTest(1:valDataLen) == uniqueOptBeamPairIdx(n));
    locX = sampledLocMatTest(beamPairIdx, 1);
    locY = sampledLocMatTest(beamPairIdx, 2);
    scatter(locX, locY, [], color(n, :));
    hold on;
end
scatter(prm.posTx(1),prm.posTx(2),100,'r^','filled');
scatter(prm.ScatPos(1,:),prm.ScatPos(2,:),100,[0.9290 0.6940 0.1250],'s','filled');
hold off
xlabel('x (m)')
ylabel('y (m)')
xlim([0 10])
ylim([0 10])
title('Optimal Beam Pair Indices (Validation Data)')
```

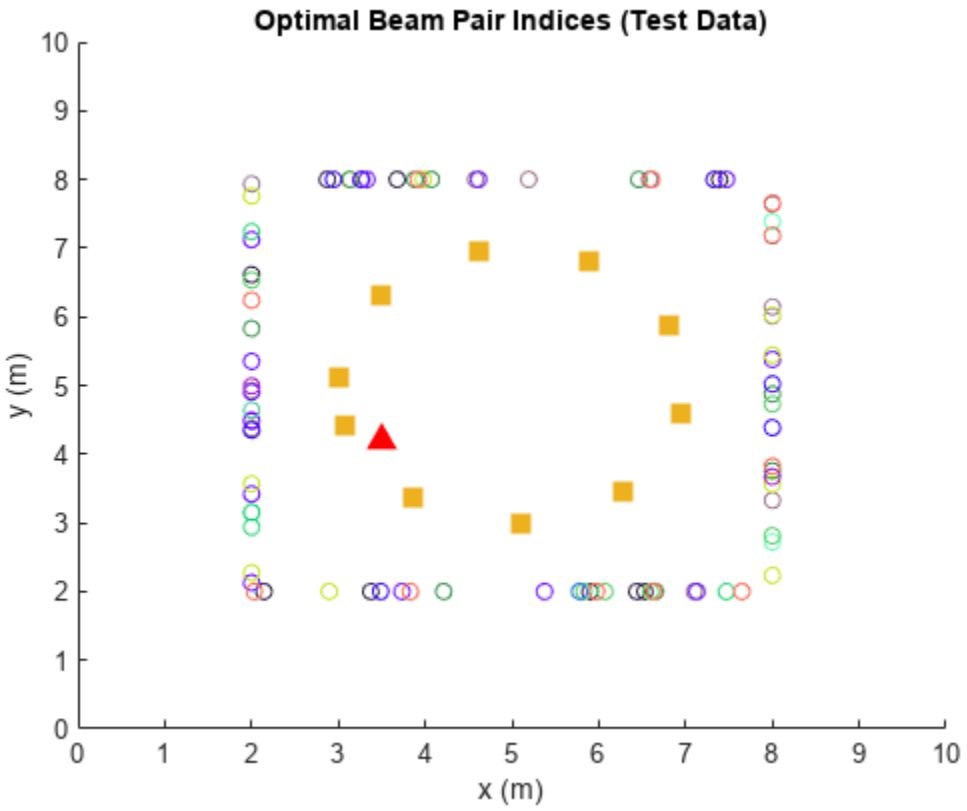



```
figure
histogram(valOut)
title('Histogram of Optimal Beam Pair Indices (Validation Data)')
xlabel('Beam Pair Index')
ylabel('Number of Occurrences')
```

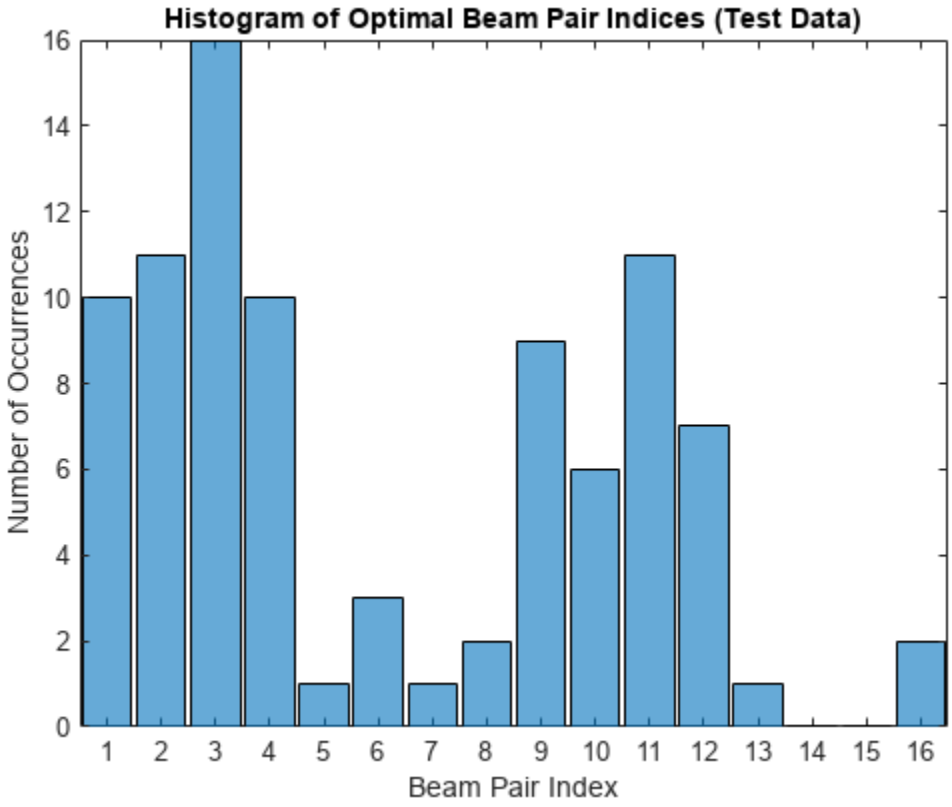


Plot Optimal Beam Pair Distribution for Test Data

```
figure
rng(111)    % for colors in plots
color = rand(NBeamPairInTestData, 3);
uniqueOptBeamPairIdx = unique(avgOptBeamPairIdxScalarTest(1+valDataLen:end));
for n = 1:length(uniqueOptBeamPairIdx)
    beamPairIdx = find(avgOptBeamPairIdxScalarTest(1+valDataLen:end) == uniqueOptBeamPairIdx(n));
    locX = sampledLocMatTest(beamPairIdx, 1);
    locY = sampledLocMatTest(beamPairIdx, 2);
    scatter(locX, locY, [], color(n, :));
    hold on;
end
scatter(prm.posTx(1),prm.posTx(2),100,'r^','filled');
scatter(prm.ScatPos(1,:),prm.ScatPos(2,:),100,[0.9290 0.6940 0.1250],'s','filled');
hold off
xlabel('x (m)')
ylabel('y (m)')
xlim([0 10])
ylim([0 10])
title('Optimal Beam Pair Indices (Test Data)')
```




```
figure
histogram(testOutCat)
title('Histogram of Optimal Beam Pair Indices (Test Data)')
xlabel('Beam Pair Index')
ylabel('Number of Occurrences')
```



Design and Train Neural Network

Train a neural network with four hidden layers. The design is motivated by [3 on page 4-46] (four hidden layers) and [5 on page 4-46] (two hidden layers with 128 neurons in each layer) in which the receiver locations are also considered as the input to the neural network. To enable training, adjust the `doTraining` logical.

This example also provides an option to weight the classes. Classes that occur more frequently have smaller weights and classes that occur less frequently have larger weights. To use class weighting, adjust the `useDiffClassWeights` logical.

Modify the network to experiment with different designs. If you modify one of the provided data sets, you must retrain the network with the modified data sets. Retraining the network can take a significant amount of time. Adjust the `saveNet` logical to use the trained network in subsequent runs.

```
doTraining = false;
useDiffClassWeights = false;
saveNet = false;

if doTraining
    if useDiffClassWeights
        catCount = countcats(trainOut);
        catFreq = catCount/length(trainOut);
        nnzIdx = (catFreq ~= 0);
        medianCount = median(catFreq(nnzIdx));
        classWeights = 10*ones(size(catFreq));
        classWeights(nnzIdx) = medianCount./catFreq(nnzIdx);
```

```
            filename = 'nnBS_trainedNetwWeighting.mat';
        else
            classWeights = ones(1,NBeamPairInTestData);
            filename = 'nnBS_trainedNet.mat';
        end

        % Neural network design
        layers = [ ...
            featureInputLayer(3,'Name','input','Normalization','rescale-zero-one')

            fullyConnectedLayer(96,'Name','linear1')
            leakyReluLayer(0.01,'Name','leakyRelu1')

            fullyConnectedLayer(96,'Name','linear2')
            leakyReluLayer(0.01,'Name','leakyRelu2')

            fullyConnectedLayer(96,'Name','linear3')
            leakyReluLayer(0.01,'Name','leakyRelu3')

            fullyConnectedLayer(96,'Name','linear4')
            leakyReluLayer(0.01,'Name','leakyRelu4')

            fullyConnectedLayer(NBeamPairInTrainData,'Name','linear5')
            softmaxLayer('Name','softmax')
            classificationLayer('ClassWeights',classWeights,'Classes',allBeamPairIdxCell,'Name','outp

        maxEpochs = 1000;
        miniBatchSize = 256;

        options = trainingOptions('adam', ...
            'MaxEpochs',maxEpochs, ...
            'MiniBatchSize',miniBatchSize, ...
            'InitialLearnRate',1e-4, ...
            'ValidationData',{valInput,valOut}, ...
            'ValidationFrequency',500, ...
            'OutputNetwork', 'best-validation-loss', ...
            'Shuffle','every-epoch', ...
            'Plots','training-progress', ...
            'ExecutionEnvironment','cpu', ...
            'Verbose',0);

        % Train the network
        net = trainNetwork(trainInput,trainOut,layers,options);

        if saveNet
            save(filename,'net');
        end
    else
        if useDiffClassWeights
            load 'nnBS_trainedNetwWeighting.mat';
        else
            load 'nnBS_trainedNet.mat';
        end
    end
```

Compare Different Approaches: Top-*K* Accuracy

This section tests the trained network with unseen test data considering the top-K accuracy metric. The top-K accuracy metric has been widely used in the neural network-based beam selection task [2 on page 4-46]-[6 on page 4-46].

Given a receiver location, the neural network first outputs K recommended beam pairs. Then it performs an exhaustive sequential search on these K beam pairs and selects the one with the highest average RSRP as the final prediction. If the true optimal beam pair is the final selected beam pair, then a successful prediction occurs. Equivalently, a success occurs when the true optimal beam pair is one of the K recommended beam pairs by the neural network.

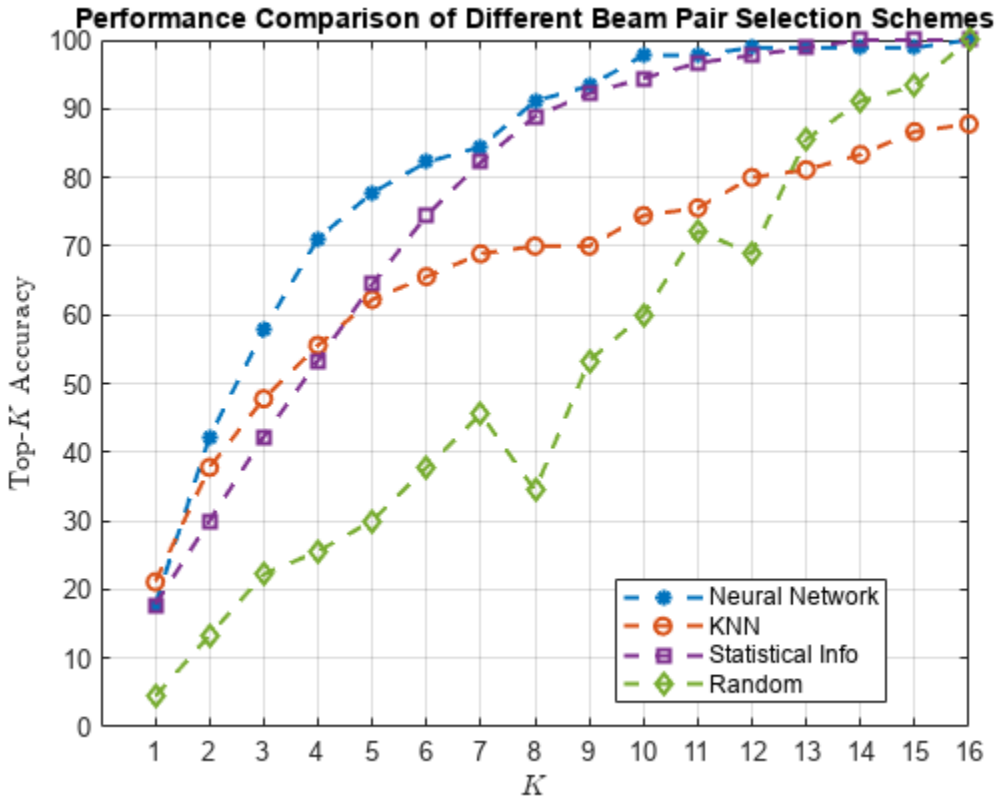
Three benchmarks are compared. Each scheme produces the K recommended beam pairs.

1. KNN - For a test sample, this method first collects K closest training samples based on GPS coordinates. The method then recommends all the beam pairs associated with these K training samples. Since each training sample has a corresponding optimal beam pair, the number of beam pairs recommended is at most K(some beam pairs might be the same).
2. Statistical Info [5 on page 4-46] - This method first ranks all the beam pairs according to their relative frequency in the training set, and then always selects the first K beam pairs.
3. Random [5 on page 4-46] - For a test sample, this method randomly chooses K beam pairs.

The plot shows that for $K = 8$, the accuracy is already more than 90%, which highlights the effectiveness of using the trained neural network for the beam selection task. When $K = 16$, every scheme (except KNN) is relaxed to the exhaustive search over all the 16 beam pairs, and hence achieves an accuracy of 100%. However, when $K = 16$, KNN considers 16 closest training samples, and the number of *distinct* beam pairs from these samples is often less than 16. Hence, KNN does not achieve an accuracy of 100%.

```
rng(111)    % for repeatability of the "Random" policy
testOut = avgOptBeamPairIdxScalarTest(1+valDataLen:end, :);
statisticCount = countcats(testOutCat);
predTestOutput = predict(net,testInput,'ExecutionEnvironment','cpu');

K = prm.numBeams^2;
accNeural = zeros(1,K);
accKNN = zeros(1,K);
accStatistic = zeros(1,K);
accRandom = zeros(1,K);
for k = 1:K
    predCorrectNeural = zeros(testDataLen,1);
    predCorrectKNN = zeros(testDataLen,1);
    predCorrectStats = zeros(testDataLen,1);
    predCorrectRandom = zeros(testDataLen,1);
    knnIdx = knnsearch(trainInput,testInput,'K',k);

    for n = 1:testDataLen
        trueOptBeamIdx = testOut(n);

        % Neural Network
        [~, topKPredOptBeamIdx] = maxk(predTestOutput(n, :),k);
        if sum(topKPredOptBeamIdx == trueOptBeamIdx) > 0
            % if true, then the true correct index belongs to one of the K predicted indices
            predCorrectNeural(n,1) = 1;
        end
```

```
            % KNN
            neighborsIdxInTrainData = knnIdx(n,:);
            topKPredOptBeamIdx= avgOptBeamPairIdxScalarTrain(neighborsIdxInTrainData);
            if sum(topKPredOptBeamIdx == trueOptBeamIdx) > 0
                % if true, then the true correct index belongs to one of the K predicted indices
                predCorrectKNN(n,1) = 1;
            end

            % Statistical Info
            [~, topKPredOptBeamIdx] = maxk(statisticCount,k);
            if sum(topKPredOptBeamIdx == trueOptBeamIdx) > 0
                % if true, then the true correct index belongs to one of the K predicted indices
                predCorrectStats(n,1) = 1;
            end

            % Random
            topKPredOptBeamIdx = randperm(prm.numBeams*prm.numBeams,k);
            if sum(topKPredOptBeamIdx == trueOptBeamIdx) > 0
                % if true, then the true correct index belongs to one of the K predicted indices
                predCorrectRandom(n,1) = 1;
            end

        end

        accNeural(k)    = sum(predCorrectNeural)/testDataLen*100;
        accKNN(k)       = sum(predCorrectKNN)/testDataLen*100;
        accStatistic(k) = sum(predCorrectStats)/testDataLen*100;
        accRandom(k)    = sum(predCorrectRandom)/testDataLen*100;

    end

    figure
    lineWidth = 1.5;
    colorNeural = [0 0.4470 0.7410];
    colorKNN = [0.8500 0.3250 0.0980];
    colorStats = [0.4940 0.1840 0.5560];
    colorRandom = [0.4660 0.6740 0.1880];
    plot(1:K,accNeural,'--*','LineWidth',lineWidth,'Color',colorNeural)
    hold on
    plot(1:K,accKNN,'--o','LineWidth',lineWidth,'Color',colorKNN)
    plot(1:K,accStatistic,'--s','LineWidth',lineWidth,'Color',colorStats)
    plot(1:K,accRandom,'--d','LineWidth',lineWidth,'Color',colorRandom)
    hold off
    grid on
    xticks(1:K)
    xlabel('$K$','interpreter','latex')
    ylabel('Top-$K$ Accuracy','interpreter','latex')
    title('Performance Comparison of Different Beam Pair Selection Schemes')
    legend('Neural Network','KNN','Statistical Info','Random','Location','best')
```

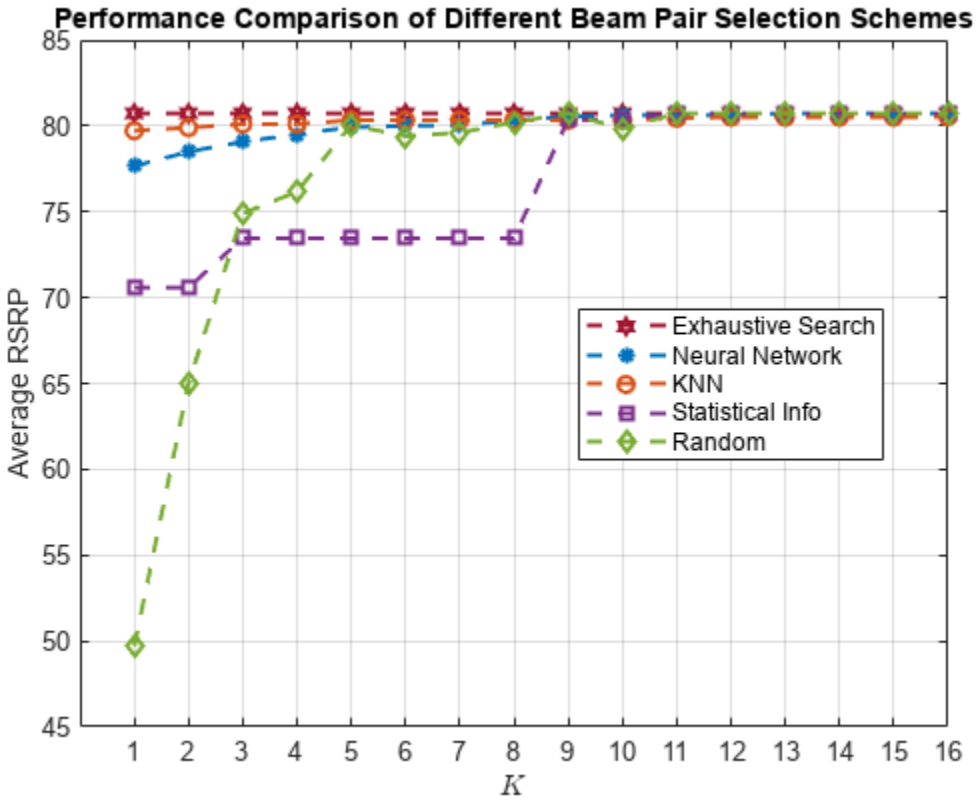


Compare Different Approaches: Average RSRP

Using unseen test data, compute the average RSRP achieved by the neural network and the three benchmarks. The plot shows that using the trained neural network results in an average RSRP close to the optimal exhaustive search.

```
rng(111)    % for repeatability of the "Random" policy
K = prm.numBeams^2;
rsrpOptimal = zeros(1,K);
rsrpNeural = zeros(1,K);
rsrpKNN = zeros(1,K);
rsrpStatistic = zeros(1,K);
rsrpRandom = zeros(1,K);
for k = 1:K
    rsrpSumOpt = 0;
    rsrpSumNeural = 0;
    rsrpSumKNN = 0;
    rsrpSumStatistic = 0;
    rsrpSumRandom = 0;

    knnIdx = knnsearch(trainInput,testInput,'K',k);

    for n = 1:testDataLen
        % Exhaustive Search
        trueOptBeamIdx = testOut(n);
        rsrp = rsrpMatTest(:,:,valDataLen+n);
        rsrpSumOpt = rsrpSumOpt + rsrp(trueOptBeamIdx);
```

```
            % Neural Network
            [~, topKPredOptCatIdx] = maxk(predTestOutput(n, :),k);
            rsrpSumNeural = rsrpSumNeural + max(rsrp(topKPredOptCatIdx));

            % KNN
            neighborsIdxInTrainData = knnIdx(n,:);
            topKPredOptBeamIdxKNN = avgOptBeamPairIdxScalarTrain(neighborsIdxInTrainData);
            rsrpSumKNN = rsrpSumKNN + max(rsrp(topKPredOptBeamIdxKNN));

            % Statistical Info
            [~, topKPredOptCatIdxStat] = maxk(statisticCount,k);
            rsrpSumStatistic = rsrpSumStatistic + max(rsrp(topKPredOptCatIdxStat));

            % Random
            topKPredOptBeamIdxRand = randperm(prm.numBeams*prm.numBeams,k);
            rsrpSumRandom = rsrpSumRandom + max(rsrp(topKPredOptBeamIdxRand));
        end
        rsrpOptimal(k)  = rsrpSumOpt/testDataLen/prm.NRepeatSameLoc;
        rsrpNeural(k)   = rsrpSumNeural/testDataLen/prm.NRepeatSameLoc;
        rsrpKNN(k)      = rsrpSumKNN/testDataLen/prm.NRepeatSameLoc;
        rsrpStatistic(k) = rsrpSumStatistic/testDataLen/prm.NRepeatSameLoc;
        rsrpRandom(k)   = rsrpSumRandom/testDataLen/prm.NRepeatSameLoc;
    end

    figure
    lineWidth = 1.5;
    plot(1:K,rsrpOptimal,'--h','LineWidth',lineWidth,'Color',[0.6350 0.0780 0.1840]);
    hold on
    plot(1:K,rsrpNeural,'--*','LineWidth',lineWidth,'Color',colorNeural)
    plot(1:K,rsrpKNN,'--o','LineWidth',lineWidth,'Color',colorKNN)
    plot(1:K,rsrpStatistic,'--s','LineWidth',lineWidth,'Color',colorStats)
    plot(1:K,rsrpRandom,'--d','LineWidth',lineWidth, 'Color',colorRandom)
    hold off
    grid on
    xticks(1:K)
    xlabel('$K$','interpreter','latex')
    ylabel('Average RSRP')
    title('Performance Comparison of Different Beam Pair Selection Schemes')
    legend('Exhaustive Search','Neural Network','KNN','Statistical Info','Random','Location','best')
```



Compare the end values for the optimal, neural network, and KNN approaches.

```
[rsrpOptimal(end-3:end); rsrpNeural(end-3:end); rsrpKNN(end-3:end);]
```

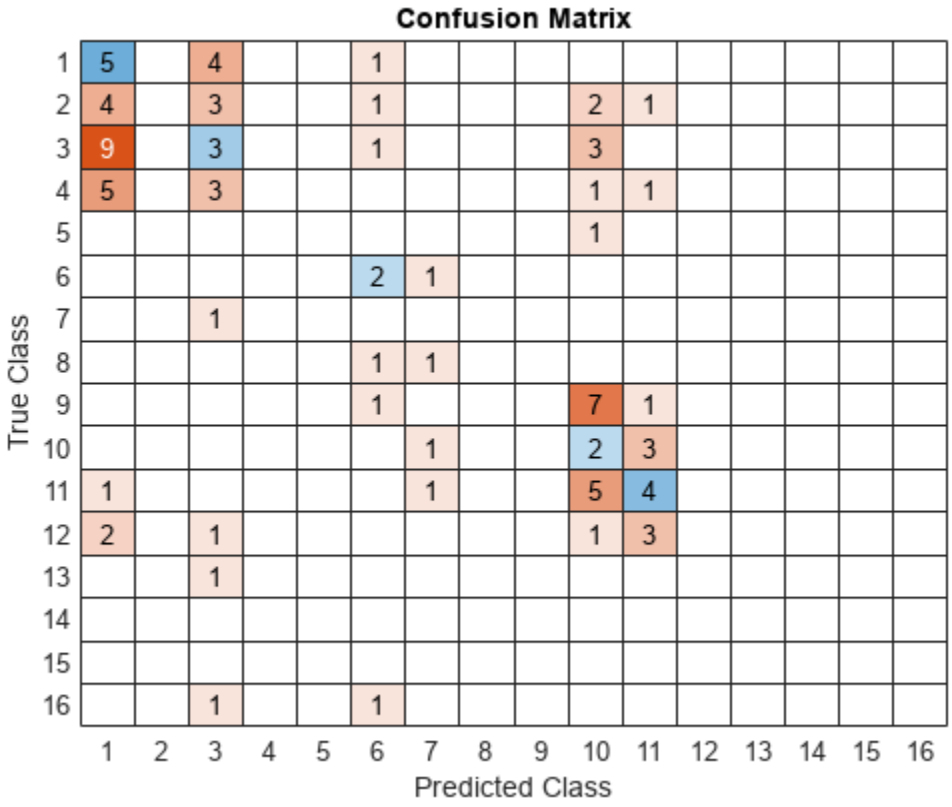
```
ans = 3×4

   80.7363   80.7363   80.7363   80.7363
   80.7363   80.7363   80.7363   80.7363
   80.5067   80.5068   80.5069   80.5212
```

The performance gap between KNN and the optimal methods indicates that the KNN might not perform well even when a larger set of beam pairs is considered, say, 256.

Plot Confusion Matrix

We observe that the classes with fewer elements are negatively impacted with the trained network. Using different weights for different classes could avoid this. Explore the same with the `useDiffClassWeights` logical and specify custom weights per class.

```
predLabels = classify(net,testInput,'ExecutionEnvironment','cpu');
figure;
cm = confusionchart(testOutCat,predLabels);
title('Confusion Matrix')
```



Conclusion and Further Exploration

This example describes the application of a neural network to the beam selection task for a 5G NR system. You can design and train a neural network that outputs a set of K good beam pairs. Beam sweeping overhead can be reduced by an exhaustive search only on those selected K beam pairs.

The example allows you to specify the scatterers in a MIMO channel. To see the impact of the channel on the beam selection, experiment with different scenarios. The example also provides presaved datasets that can be used to experiment with different network structures and training hyperparameters.

From simulation results, for the prerecorded MIMO scattering channel for 16 beam pairs, the proposed algorithm can achieve a top-K accuracy of 90% when $K = 8$. This indicates with the neural network it is sufficient to perform an exhaustive search over only half of all the beam pairs, reducing the beam sweeping overhead by 50%. Experiment with varying other system parameters to see the efficacy of the network by regenerating data, then retraining and retesting the network.

References

1 3GPP TR 38.802, "Study on New Radio access technology physical layer aspects." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

2 Klautau, A., González-Prelcic, N., and Heath, R. W., "LIDAR data for deep learning-based mmWave beam-selection," IEEE Wireless Communications Letters, vol. 8, no. 3, pp. 909–912, Jun. 2019.

3 Heng, Y., and Andrews, J. G., "Machine Learning-Assisted Beam Alignment for mmWave Systems," 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9013296.

4 Klautau, A., Batista, P., González-Prelcic, N., Wang, Y., and Heath, R. W., "5G MIMO Data for Machine Learning: Application to Beam-Selection Using Deep Learning," 2018 Information Theory and Applications Workshop (ITA), 2018, pp. 1-9, doi: 10.1109/ITA.2018.8503086.

5 Matteo, Z., <https://github.com/ITU-AI-ML-in-5G-Challenge/PS-012-ML5G-PHY-Beam-Selection_BEAMSOUP> (This is the team achieving the highest test score in the ITU Artificial Intelligence/Machine Learning in 5G Challenge in 2020).

6 Sim, M. S., Lim, Y., Park, S. H., Dai, L., and Chae, C., "Deep Learning-Based mmWave Beam Selection for 5G NR/6G With Sub-6 GHz Channel Information: Algorithms and Prototype Validation," IEEE Access, vol. 8, pp. 51634-51646, 2020.

Local Function

```
function prm = validateParams(prm)
% Validate user specified parameters and return updated parameters
%
% Only cross-dependent checks are made for parameter consistency.

    if strcmpi(prm.FreqRange,'FR1')
        if prm.CenterFreq > 7.125e9 || prm.CenterFreq < 410e6
            error(['Specified center frequency is outside the FR1 ', ...
                   'frequency range (410 MHz - 7.125 GHz).']);
        end
        if strcmpi(prm.SSBlockPattern,'Case D') ||  ...
           strcmpi(prm.SSBlockPattern,'Case E')
            error(['Invalid SSBlockPattern for selected FR1 frequency ' ...
                'range. SSBlockPattern must be one of ''Case A'' or ' ...
                '''Case B'' or ''Case C'' for FR1.']);
        end
        if ~((length(prm.SSBTransmitted)==4) || ...
             (length(prm.SSBTransmitted)==8))
            error(['SSBTransmitted must be a vector of length 4 or 8', ...
                   'for FR1 frequency range.']);
        end
        if (prm.CenterFreq <= 3e9) && (length(prm.SSBTransmitted)~=4)
            error(['SSBTransmitted must be a vector of length 4 for ' ...
                   'center frequency less than or equal to 3GHz.']);
        end
        if (prm.CenterFreq > 3e9) && (length(prm.SSBTransmitted)~=8)
            error(['SSBTransmitted must be a vector of length 8 for ', ...
                   'center frequency greater than 3GHz and less than ', ...
                   'or equal to 7.125GHz.']);
        end
    else % 'FR2'
        if prm.CenterFreq > 52.6e9 || prm.CenterFreq < 24.25e9
            error(['Specified center frequency is outside the FR2 ', ...
                   'frequency range (24.25 GHz - 52.6 GHz).']);
        end
        if ~(strcmpi(prm.SSBlockPattern,'Case D') || ...
                strcmpi(prm.SSBlockPattern,'Case E'))
            error(['Invalid SSBlockPattern for selected FR2 frequency ' ...
                'range. SSBlockPattern must be either ''Case D'' or ' ...
                '''Case E'' for FR2.']);
        end
```

```
            if length(prm.SSBTransmitted)~=64
                error(['SSBTransmitted must be a vector of length 64 for ', ...
                       'FR2 frequency range.']);
            end
        end

        % Number of beams at transmit/receive ends
        prm.numBeams = sum(prm.SSBTransmitted);

        prm.NumTx = prod(prm.TxArraySize);
        prm.NumRx = prod(prm.RxArraySize);
        if prm.NumTx==1 || prm.NumRx==1
            error(['Number of transmit or receive antenna elements must be', ...
                   ' greater than 1.']);
        end
        prm.IsTxURA = (prm.TxArraySize(1)>1) && (prm.TxArraySize(2)>1);
        prm.IsRxURA = (prm.RxArraySize(1)>1) && (prm.RxArraySize(2)>1);

        if ~( strcmpi(prm.RSRPMode,'SSSonly') || ...
              strcmpi(prm.RSRPMode,'SSSwDMRS') )
            error(['Invalid RSRP measuring mode. Specify either ', ...
                   '''SSSonly'' or ''SSSwDMRS'' as the mode.']);
        end

        % Select SCS based on SSBlockPattern
        switch lower(prm.SSBlockPattern)
            case 'case a'
                scs = 15;
                cbw = 10;
                scsCommon = 15;
            case {'case b', 'case c'}
                scs = 30;
                cbw = 25;
                scsCommon = 30;
            case 'case d'
                scs = 120;
                cbw = 100;
                scsCommon = 120;
            case 'case e'
                scs = 240;
                cbw = 200;
                scsCommon = 120;
        end
        prm.SCS = scs;
        prm.ChannelBandwidth = cbw;
        prm.SubcarrierSpacingCommon = scsCommon;
    end
```

See Also

Functions
`trainNetwork` | `trainingOptions`

Objects
`featureInputLayer` | `reluLayer` | `fullyConnectedLayer`

Related Examples

- “Deep Learning in MATLAB” (Deep Learning Toolbox)
- “NR SSB Beam Sweeping” on page 1-98

Train DQN Agent for Beam Selection

This example shows how to train a deep Q-network (DQN) reinforcement learning agent to accomplish the beam selection task in a 5G New Radio (NR) communications system. Instead of an exhaustive beam search over all the beam pairs, the trained agent increases beam selection accuracy by selecting the beam with highest signal strength while reducing the beam transition cost. When you use an access network node (gNB) with four beams, simulation results in this example show the trained agent selects beams with greater than 90% maximum possible signal strengths.

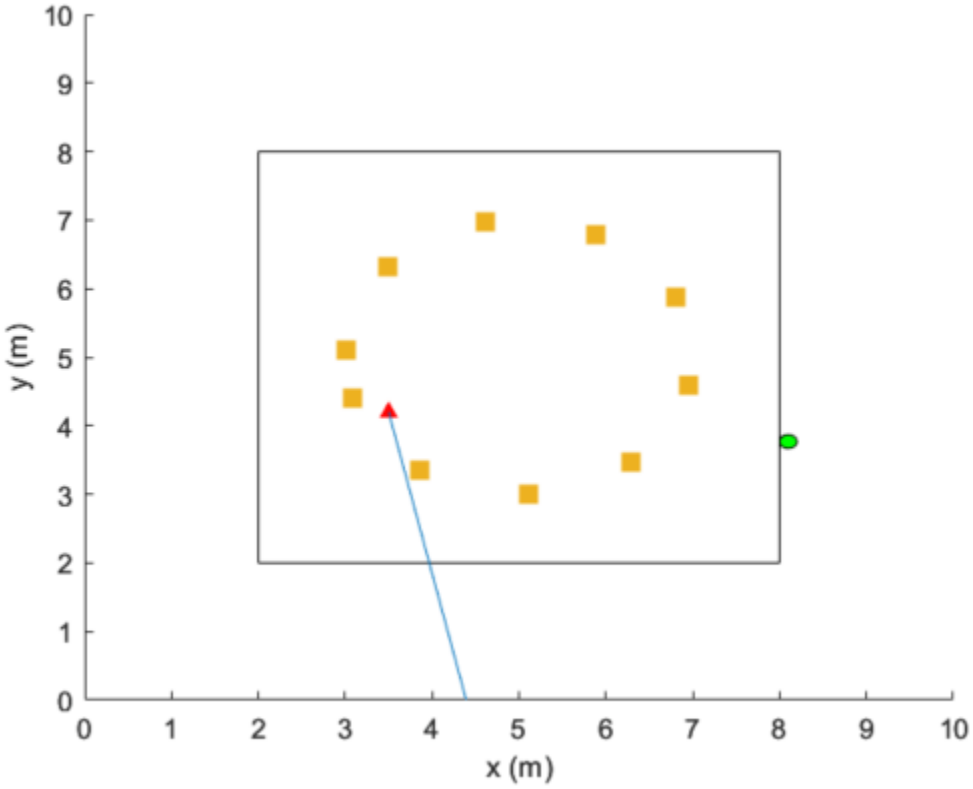
Introduction

To enable millimeter wave (mmWave) communications, beam management techniques must be used due to the high pathloss and blockage experienced at high frequencies. Beam management is a set of Layer 1 (physical layer) and Layer 2 (medium access control) procedures to establish and retain an optimal beam pair (transmit beam and a corresponding receive beam) for good connectivity [1 on page 4-55]. For examples of NR beam management procedures, see “NR SSB Beam Sweeping” on page 1-98 and “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113.

This example considers beam selection procedures when a connection is established between the user equipment (UE) and gNB. In 5G NR, the beam selection procedure for initial access consists of beam sweeping, which requires exhaustive searches over all the beams on the transmitter and the receiver sides, and then selection of the beam pair offering the strongest reference signal received power (RSRP). Since mmWave communications require many antenna elements, implying many beams, an exhaustive search over all beams becomes computationally expensive and increases the initial access time.

To avoid repeatedly performing an exhaustive search and to reduce the communication overhead, this example uses a reinforcement learning (RL) agent to perform beam selection using the GPS coordinates of the receiver and the current beam angle while the UE moves around a track.

In this figure, the square represents the track that the UE (green circle) moves around, the red triangle represents the location of the base station (gNB), the yellow squares represent the channel scatterers, and the blue line represents the selected beam.



For more information on DQN reinforcement learning agents, see “Deep Q-Network (DQN) Agents” (Reinforcement Learning Toolbox).

Define Environment

To train a reinforcement learning agent, you must define the environment with which it will interact. The reinforcement learning agent selects actions given observations. The goal of the reinforcement learning algorithm is to find optimal actions that maximize the expected cumulative long-term reward received from the environment during the task. For more information about reinforcement learning agents, see “Reinforcement Learning Agents” (Reinforcement Learning Toolbox).

For the beam selection environment:

- The observations are represented by UE position information and the current beam selection.
- The actions are a selected beam out of four total beam angles from the gNB.
- The reward r_t at time step t is given by:

$$r_t = r_{\text{rsrp}} + r_\theta$$
$$r_{\text{rsrp}} = 0.9 \times \text{rsrp}$$
$$r_\theta = -0.1 \times |\theta_t - \theta_{t-1}| .$$

r_{rsrp} is a reward for the signal strength measured from the UE (rsrp) and r_θ is a penalty for control effort. θ is the beam angle in degrees.

The environment is created from the RSRP data generated from in “Neural Network for Beam Selection” on page 4-27. In the prerecorded data, receivers are randomly distributed on the perimeter of a 6-meter square and configured with 16 beam pairs (four beams on each end, analog beamformed with one RF chain). Using a MIMO scattering channel, the example considers 200 receiver locations in the training set (`nnBS_TrainingData.mat`) and 100 receiver locations in the test sets (`nnBS_TestData.mat`). The prerecorded data uses 2-D location coordinates.

The `nnBS_TrainingData.mat` file contains a matrix of receiver locations, `locationMatTrain`, and an RSRP measurements of 16 beam pairs, `rsrpMatTrain`. Since receiver beam selection does not significantly affect signal strength, you compute the mean RSRP for each base station antenna beam for each UE location. Thus, the action space is four beam angles. You reorder the recorded data to imitate the receiver moving clockwise around the base station.

To generate new training and test sets, set `useSavedData` to `false`. Be aware that regenerating data can take up to a few hours.

```
% Set the random generator seed for reproducibility
rng(0)

useSavedData = true;
if useSavedData
    % Load data generated from Neural Network for Beam Selection example
    load nnBS_TrainingData
    load nnBS_TestData
    load nnBS_position
else
    % Generate data
    helperNNBSGenerateData(); %#ok
    position.posTX = prm.posTx;
    position.ScatPos = prm.ScatPos;
end
locationMat = locationMatTrain(1:4:end,:);

% Sort location in clockwise order
secLen = size(locationMat,1)/4;
[~,b1] = sort(locationMat(1:secLen,2));
[~,b2] = sort(locationMat(secLen+1:2*secLen,1));
[~,b3] = sort(locationMat(2*secLen+1:3*secLen,2),"descend");
[~,b4] = sort(locationMat(3*secLen+1:4*secLen,1),"descend");
idx = [b1;secLen+b2;2*secLen+b3;3*secLen+b4];

locationMat =  locationMat(idx,:);

% Compute average RSRP for each gNB beam and sort in clockwise order
avgRsrpMatTrain = rsrpMatTrain/4;    % prm.NRepeatSameLoc=4;
avgRsrpMatTrain = 100*avgRsrpMatTrain./max(avgRsrpMatTrain, [],"all");
avgRsrpMatTrain = avgRsrpMatTrain(:,:,idx);
avgRsrpMatTrain = mean(avgRsrpMatTrain,1);

% Angle rotation matrix: update for nBeams>4
txBeamAng = [-78,7,92,177];
rotAngleMat = [
    0 85 170 105
    85 0 85 170
    170 85 0 85
    105 170 85 0];
rotAngleMat = 100*rotAngleMat./max(rotAngleMat,[],"all");
```

```
% Create training environment using generated data
envTrain = BeamSelectEnv(locationMat,avgRsrpMatTrain,rotAngleMat,position);
```

The environment is defined in the `BeamSelectEnv` supporting class, which is created using the `rlCreateEnvTemplate` class. `BeamSelectEnv.m` is located in this example folder. The reward and penalty functions are defined within and are updated as the agent interacts with the environment.

Create Agent

A DQN agent approximates the long-term reward for the given observations and actions by using a `rlVectorQValueFunction` (Reinforcement Learning Toolbox) critic. Vector Q-value function approximators have observations as inputs and state-action values as outputs. Each output element represents the expected cumulative long-term reward for taking the corresponding discrete action from the state indicated by the observation inputs.

The example uses the default critic network structures for the given observation and action specification.

```
obsInfo = getObservationInfo(envTrain);
actInfo = getActionInfo(envTrain);
agent = rlDQNAgent(obsInfo,actInfo);
```

View the critic neural network.

```
criticNetwork = getModel(getCritic(agent));
analyzeNetwork(criticNetwork)
```

To foster expoloration, the DQN agent in this example optimizes with a learning rate of 0.001 and an epsilon decay factor of 0.0001. For a full list of DQN hyperparameters and their descriptions, see `rlDQNAgentOptions` (Reinforcement Learning Toolbox).

Specify the agent hyperparameters for training.

```
agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e-3;
agent.AgentOptions.EpsilonGreedyExploration.EpsilonDecay = 1e-4;
```

Train Agent

To train the agent, first specify the training options using `rlTrainingOptions` (Reinforcement Learning Toolbox). For this example, run each training session for at most 500 episodes, with each episode lasting at most 200 time steps, corresponding to one full loop of the track.

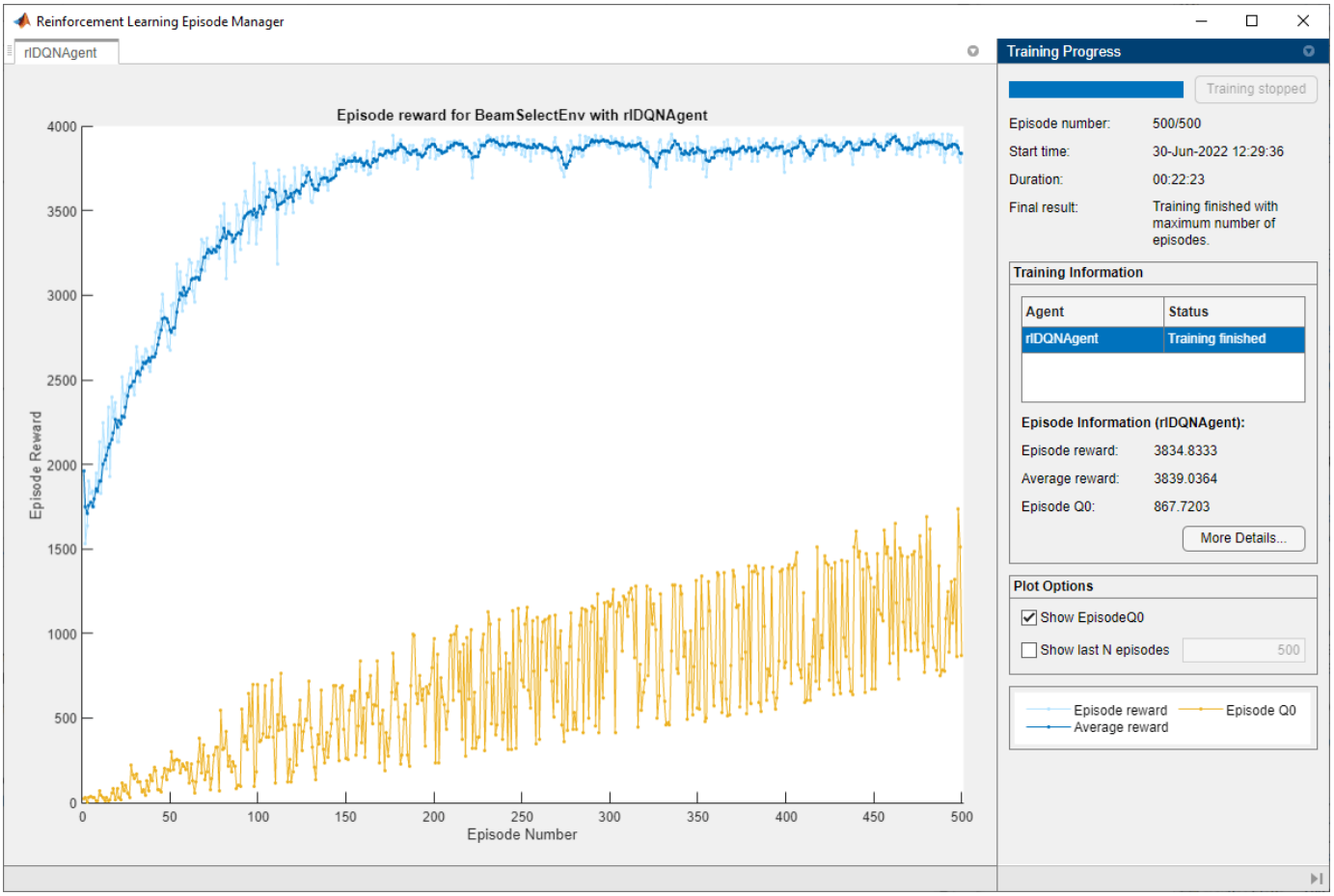
```
trainOpts = rlTrainingOptions(...
    MaxEpisodes=500, ...
    MaxStepsPerEpisode=200, ...          % training data size = 200
    StopTrainingCriteria="AverageSteps", ...
    StopTrainingValue=500, ...
    Plots="training-progress");
```

Train the agent using the `train` (Reinforcement Learning Toolbox) function. Training this agent is a computationally intensive process that takes several minutes to complete. To save time while running this example, load a pretrained agent by setting `doTraining` to `false`. To train the agent yourself, set `doTraining` to `true`.

```
doTraining = false;
if doTraining
```

```
    trainingStats = train(agent,envTrain,trainOpts); %#ok
else
    load("nnBS_RLAgent.mat")
end
```

This figure shows the progression of the training. You can expect different results due to randomness inherent to the training process.



Simulate Trained Agent

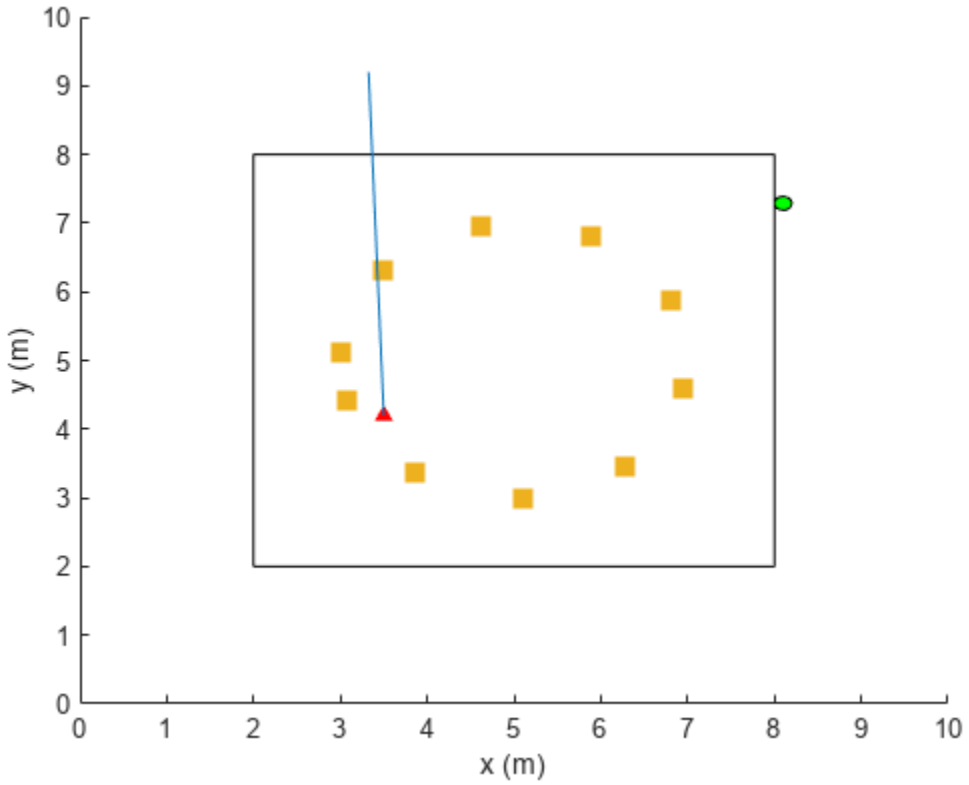
To validate the trained agent, rfirst set up a test environment with UE locations that the agent has not seen in the training process.

```
locationMat = locationMatTest(1:4:end,:);

% Sort location in clockwise order
secLen = size(locationMat,1)/4;
[~,b1] = sort(locationMat(1:secLen,2));
[~,b2] = sort(locationMat(secLen+1:2*secLen,1));
[~,b3] = sort(locationMat(2*secLen+1:3*secLen,2),"descend");
[~,b4] = sort(locationMat(3*secLen+1:4*secLen,1),"descend");
idx = [b1;secLen+b2;2*secLen+b3;3*secLen+b4];

locationMat =  locationMat(idx,:);
```

```
% Compute average RSRP
avgRsrpMatTest = rsrpMatTest/4;  % 4 = prm.NRepeatSameLoc;
avgRsrpMatTest = 100*avgRsrpMatTest./max(avgRsrpMatTest, [],"all");
avgRsrpMatTest = avgRsrpMatTest(:,:,idx);
avgRsrpMatTest = mean(avgRsrpMatTest,1);

% Create test environment
envTest = BeamSelectEnv(locationMat,avgRsrpMatTest,rotAngleMat,position);
```

Simulate the environment with the trained agent. For more information on agent simulation, see `rlSimulationOptions` (Reinforcement Learning Toolbox) and `sim` (Reinforcement Learning Toolbox).

```
plot(envTest)
sim(envTest,agent,rlSimulationOptions("MaxSteps",100))
```




```
maxPosibleRsrp = sum(max(squeeze(avgRsrpMatTest)));
rsrpSim =  envTest.EpisodeRsrp;
disp("Agent RSRP/Maximum RSRP = " + rsrpSim/maxPosibleRsrp*100 +"%")
```

```
Agent RSRP/Maximum RSRP = 94.9399%
```

References

[1] 3GPP TR 38.802. "Study on New Radio Access Technology Physical Layer Aspects." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] Sutton, Richard S., and Andrew G. Barto. Reinforcement Learning: An Introduction. Second edition. Cambridge, MA: MIT Press, 2020.

See Also

Related Examples

- “Neural Network for Beam Selection” on page 4-27
- “What Is Reinforcement Learning?” (Reinforcement Learning Toolbox)
- “Train Reinforcement Learning Agents” (Reinforcement Learning Toolbox)
- “Deep Learning in MATLAB” (Deep Learning Toolbox)

5

End-To-End Simulation

Transmission Over MIMO Channel Model with Delay Profile TDL

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

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

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

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

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

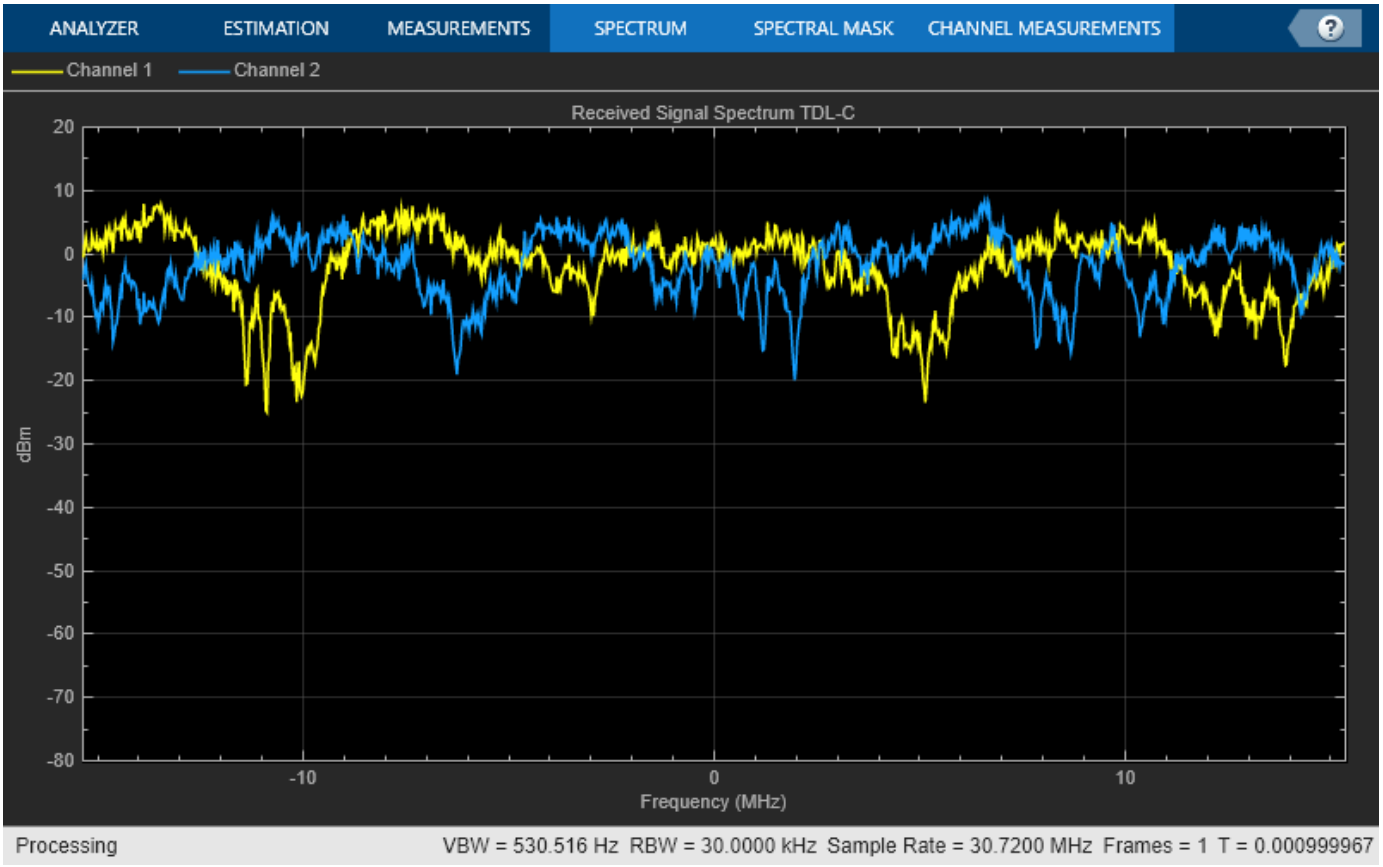
```
SR = 30.72e6;
T = SR * 1e-3;
tdl.SampleRate = SR;
tdlinfo = info(tdl);
Nt = tdlinfo.NumTransmitAntennas;

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

Transmit the input waveform through the channel.

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

Plot the received waveform spectrum.

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



See Also

Objects
`nrTDLChannel`

Plot Path Gains for TDL-E Delay Profile with SISO

Plot the path gains of a tapped delay line (TDL) single-input/single-output (SISO) channel using an `nrTDLChannel` System object.

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

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

Configure the transmit and receive antenna arrays for SISO operation.

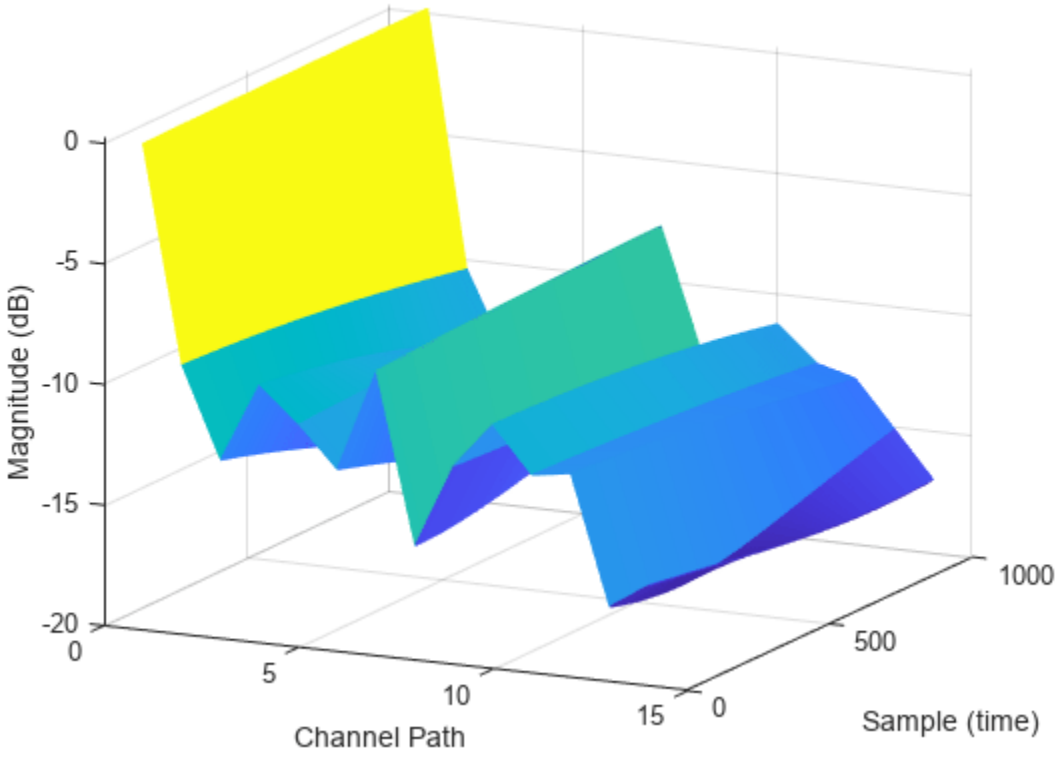
```
tdl.NumTransmitAntennas = 1;
tdl.NumReceiveAntennas = 1;
```

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

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

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

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



See Also

Objects
`nrTDLChannel`

Reconstruct Channel Impulse Response Using CDL Channel Path Filters

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

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

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

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

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

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

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

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

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

Transmit the input waveform through the channel.

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

Obtain the path filters used in channel filtering.

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

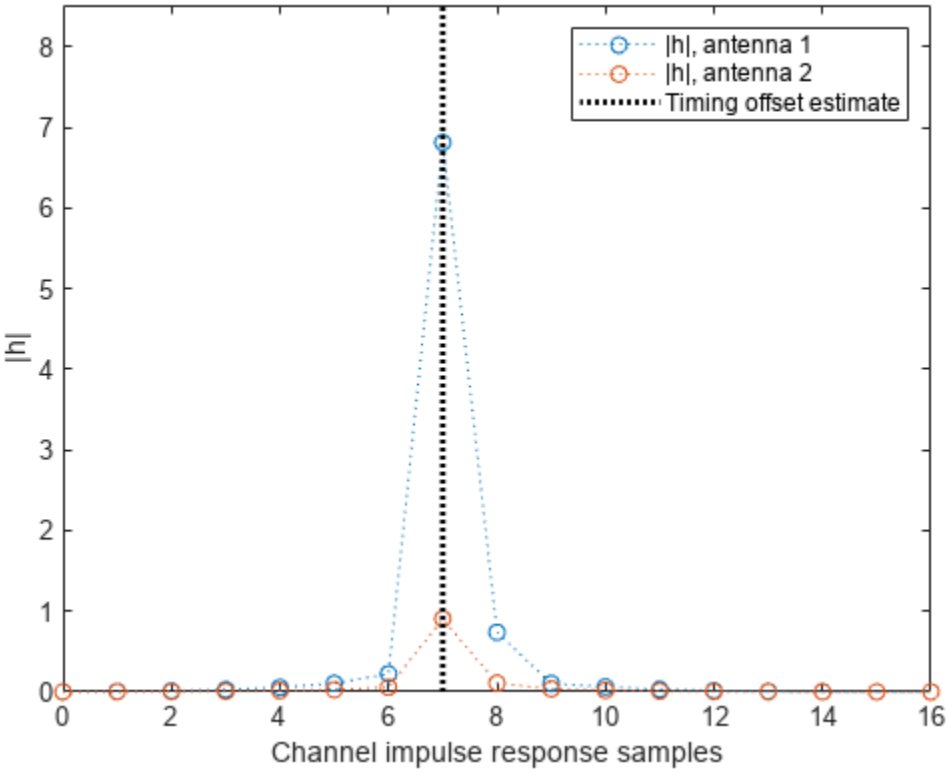
Perform timing offset estimation using `nrPerfectTimingEstimate`.

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

Plot the magnitude of the channel impulse response.

```
[Nh,Nr] = size(mag);
plot(0:(Nh-1),mag,'o:');
hold on;
plot([offset offset],[0 max(mag(:))*1.25],'k:','LineWidth',2);
```

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



See Also

Functions

`nrPerfectTimingEstimate`

Visualize CDL Channel Model Characteristics

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

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

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

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

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

Configure the vertical and horizontal element spacing and the vertical and horizontal panel spacing, in wavelength, as a vector of the form $[\lambda_v\,\lambda_h\,dg_v\,dg_h]$. Because panel spacing is measured from the center of the panels, to avoid panel overlapping, set dg_h to a value greater than 1 wavelength. To ensure uniform antenna element spacing across vertically and horizontally separated panels, configure panel spacings as $dg_v = \lambda_v \times M$ and $dg_h = \lambda_h \times N$, respectively.

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

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

```
cdl.TransmitArrayOrientation = [0 15 0]';
```

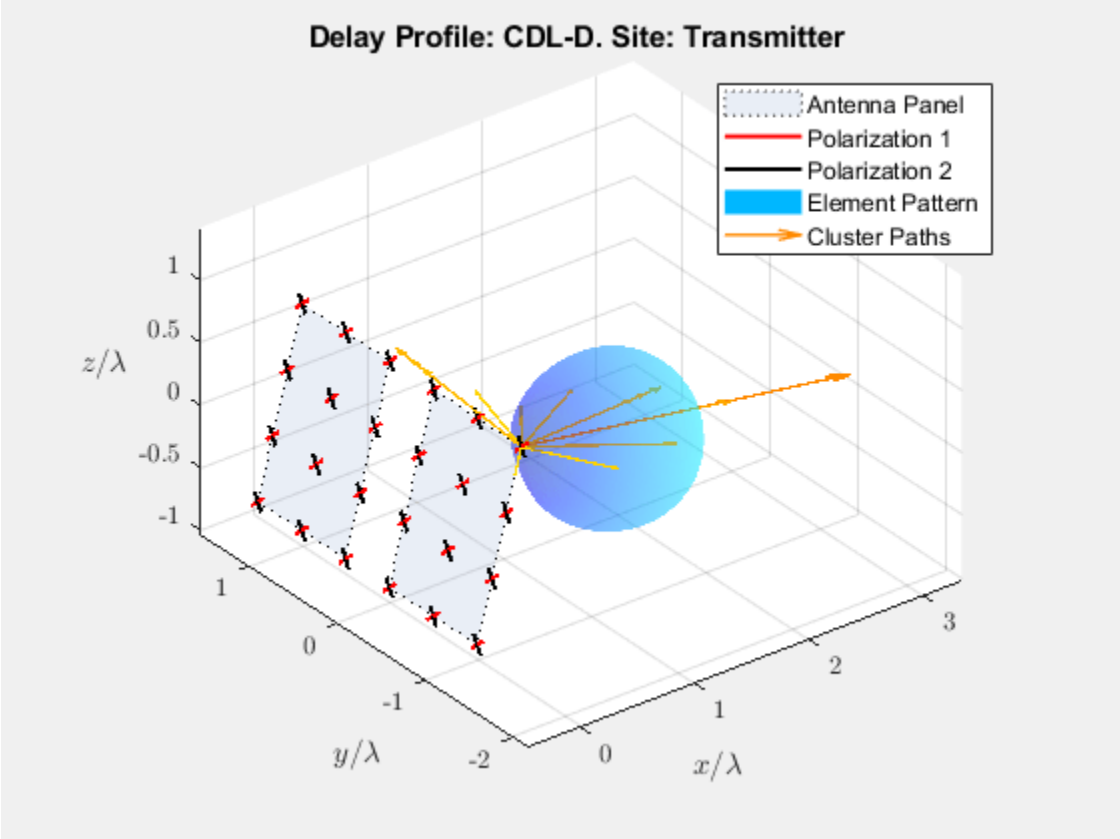
For an overview of all other transmit antenna array properties, see the “TransmitAntennaArray” property of the `nrCDLChannel` System object.

Display the channel characteristics at the transmitter end.

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

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

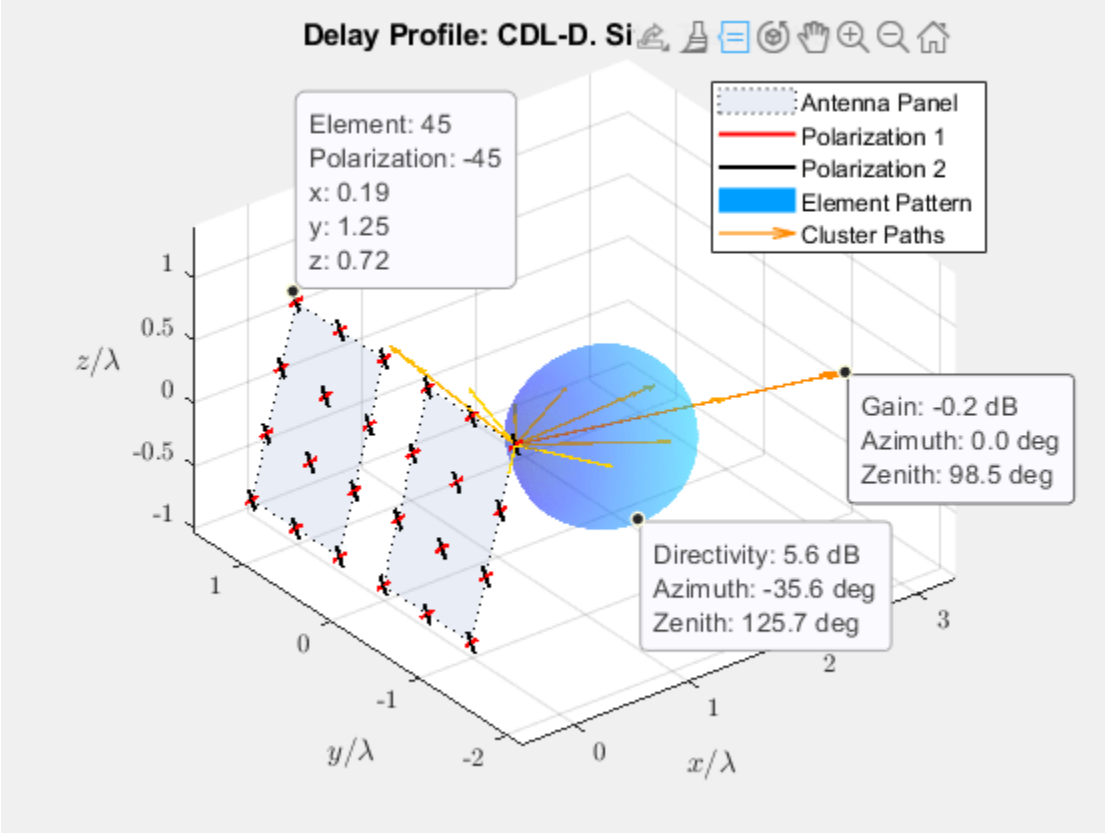
```
datacursormode on;
```



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

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

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

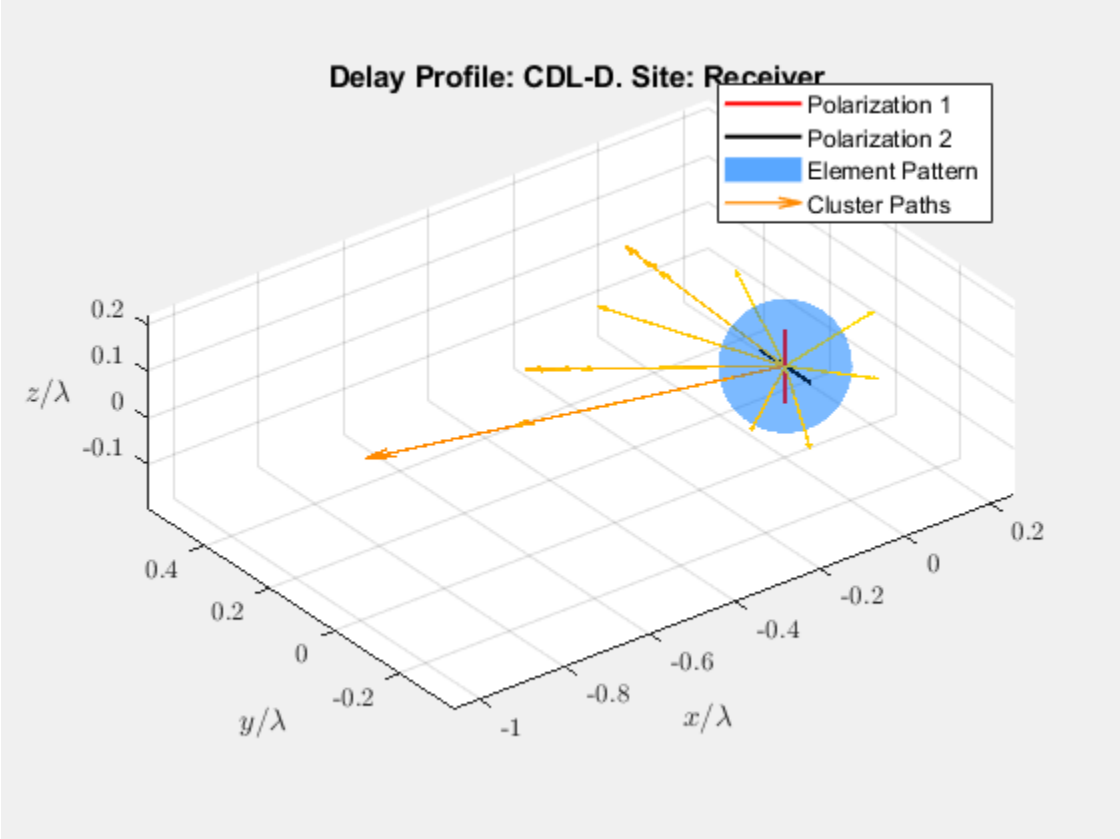


Visualize and explore channel characteristics at the receiver end. To customize the receive antenna array, use the “ReceiveAntennaArray” property of the `nrCDLChannel` System object. Then, display the channel characteristics at the receiver end by calling the `displayChannel` function with the `'LinkEnd','Rx'`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

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

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

```
datacursormode on;
```



See Also

Functions
`info` | `displayChannel`

Objects
`nrCDLChannel`

More About

- “Interactively Explore Plotted Data”

NR PUCCH Block Error Rate

This example shows how to measure the block error rate (BLER) of uplink control information (UCI) transmitted on the physical uplink control channel (PUCCH) in a 5G NR link using 5G Toolbox™ features.

Introduction

This example measures the BLER of UCI transmitted on PUCCH format 3 of a 5G link.

The UCI BLER is defined as the probability of incorrectly decoding the UCI when the UCI is transmitted.

$$\mathrm{BLER}_{\mathrm{UCI}} = \frac{\#(\text{false UCI})}{\#(\text{UCI})}$$

- #(false UCI) is the number of instances when transmitted UCI is incorrectly decoded.
- #(UCI) is the number of instances when UCI is transmitted.

These 5G NR features are modeled in this example.

- UCI encoding and decoding
- PUCCH and associated demodulation reference signal (DM-RS)
- OFDM modulation and demodulation
- Tapped delay line (TDL) propagation channel

Other features of the simulation in this example are:

- Perfect or practical synchronization and channel estimation
- Equalization

To reduce the total simulation time, you can use Parallel Computing Toolbox™ features to execute the signal to noise ratio (SNR) points of the SNR loop in parallel.

Simulation Length and SNR Points

Set the length of the simulation in terms of the number of 10 ms frames. A large number for `NFrames` must be used to produce meaningful throughput results. Set the SNR points to simulate. The SNR is defined per resource element (RE) and applies to each receive antenna. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
simParameters = struct;         % Create simParameters structure
simParameters.NFrames = 5;      % Number of 10 ms frames
simParameters.SNRIn = -14:2:-4; % SNR range (dB)
```

Set the `displaySimulationInformation` variable to `true` to display the information of BLER simulation at each SNR point.

`displaySimulationInformation = true;`

Carrier and PUCCH Configuration

Set carrier with these properties:

- Physical layer cell identity
- Subcarrier spacing (SCS) in kHz
- Cyclic prefix
- Bandwidth in resource blocks (RBs). Each RB contains 12 subcarriers
- Starting RB of the carrier with respect to the common resource block 0 (CRB 0)

Set PUCCH format 3 configuration with these properties:

- Allocated set of RBs
- Symbol allocation ([S L])
- Modulation scheme
- Frequency hopping configuration
- Second hop starting RB
- Group hopping configuration
- Scrambling identities
- Additional DM-RS configuration

Set these additional simulation-wide parameters:

- Number of transmit antennas
- Number of receive antennas

```
% Set carrier resource grid properties (15 kHz SCS and 10 MHz bandwidth)
carrier = nrCarrierConfig;
carrier.NCellID = 0 ;
carrier.SubcarrierSpacing = 15 kHz ;
carrier.CyclicPrefix = normal ;
carrier.NSizeGrid = 52;
carrier.NStartGrid = 0;

% Set PUCCH format 3 properties
pucch = nrPUCCH3Config;
pucch.PRBSet = 0;
pucch.SymbolAllocation = [0 14];
pucch.Modulation = QPSK ;
pucch.FrequencyHopping = intraSlot ;
pucch.SecondHopStartPRB = (carrier.NSizeGrid-1) - (numel(pucch.PRBSet)-1);
pucch.GroupHopping = neither ;
pucch.HoppingID = 0;
pucch.NID = [];
pucch.RNTI = 1 ;
pucch.AdditionalDMRS = 0 ;

% Set number of transmit and receive antennas
```

```
simParameters.NTxAnts = 1;
simParameters.NRxAnts = 2;
```

UCI Configuration

The field `NumUCIBits` indicates the number of random UCI bits used for generation of the UCI payload. The number of UCI bits must be in the range from 3 to 1706.

```
simParameters.NumUCIBits = 16; % Number of UCI bits
```

Channel Estimator Configuration

The logical variable `perfectChannelEstimator` controls the channel estimation and synchronization behavior. When you set the value to `true`, the perfect channel estimation and synchronization is used. Otherwise, practical channel estimation and synchronization is used, based on the values of the received PUCCH DM-RS.

```
perfectChannelEstimator = true;
```

Propagation Channel Model Configuration

Create the TDL channel model with the configuration of 'TDLC300-100 Low', specified in Annex G, Table G.2.1.1-4 of TS 38.104.

```
% Set up TDL channel
channel = nrTDLChannel;
channel.DelayProfile = 'TDLC300';
channel.MaximumDopplerShift = 100; % in Hz
channel.MIMOCorrelation = 'low';
channel.TransmissionDirection = 'Uplink';
channel.NumTransmitAntennas = simParameters.NTxAnts;
channel.NumReceiveAntennas = simParameters.NRxAnts;
```

Set the sampling rate for the channel model by using the value returned from the `nrOFDMInfo` function.

```
waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;
```

Processing Loop and Results

To determine the BLER of UCI at each SNR point, the UCI transmitted on the PUCCH is analyzed per transmission instance using these steps.

1. **Generate the resource grid:** The `nrUCIEncode` function encodes the UCI. The `nrPUCCH` function modulates the encoded UCI. An implementation-specific multiple-input-multiple-output (MIMO) precoding is applied to the modulated symbols. These modulated symbols along with reference signal are mapped to the resource grid.
2. **Generate the waveform:** The `nrOFDMModulate` function OFDM-modulates the generated grid to get the time domain waveform.
3. **Model and apply a noisy channel:** The generated waveform is passed through a TDL fading channel to get the faded waveform. Then, additive white Gaussian noise (AWGN) is added to the faded waveform. The SNR for each layer is defined per RE and per receive antenna. For an

explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

4. **Perform synchronization and OFDM demodulation:** For perfect synchronization, the path gains and path filters of the channel are used. For practical synchronization, the received waveform is correlated with the PUCCH DM-RS. The `nrOFDMDemodulate` function then OFDM-demodulates the synchronized signal.
5. **Perform channel estimation:** For perfect channel estimation, the path gains, path filters, and the sample times of channel snapshots are used. For practical channel estimation, the PUCCH DM-RS is used.
6. **Extract the PUCCH and perform equalization:** The `nrExtractResources` function extracts the REs corresponding to the PUCCH allocation from the received OFDM resource grid and the estimated channel grid. The `nrEqualizeMMSE` function then equalizes the received PUCCH REs.
7. **Decode the PUCCH:** The equalized PUCCH symbols, along with a noise estimate, are demodulated and descrambled to obtain an estimate of the received codeword.
8. **Decode the UCI:** The decoded codeword is passed through the `nrUCIDecode` function, and the number of instances of incorrect UCI decoding is recorded.

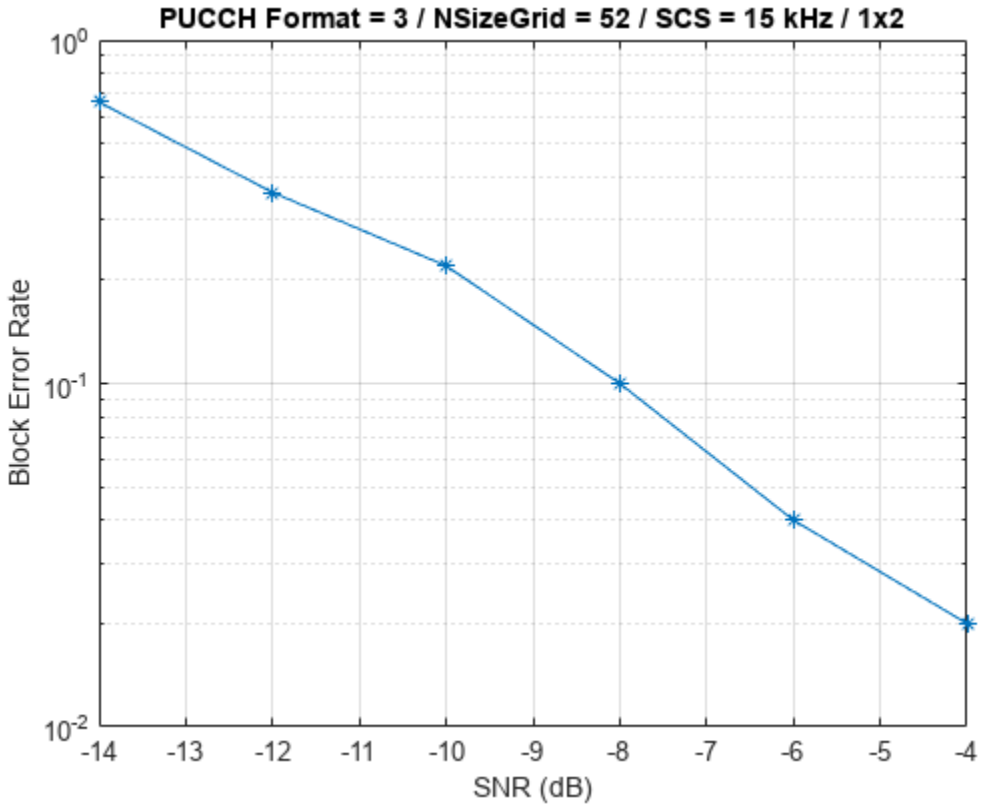
```
% Obtain channel information
chInfo = info(channel);

% Specify array to store output(s) for all SNR points
blerUCI = zeros(length(simParameters.SNRIn),1);

% Assign temporary variables for parallel simulation
nTxAnts = simParameters.NTxAnts;
nRxAnts = simParameters.NRxAnts;
snrIn = simParameters.SNRIn;
nFrames = simParameters.NFrames;
ouci = simParameters.NumUCIBits;
nFFT = waveformInfo.Nfft;
symbolsPerSlot = carrier.SymbolsPerSlot;
slotsPerFrame = carrier.SlotsPerFrame;

% Validate number of frames
validateattributes(nFrames,{'double'},{'scalar','positive','integer'},'','simParameters.NFrames')

% Validate SNR range
validateattributes(snrIn,{'double'},{'real','vector','finite'},'','simParameters.SNRIn')

% Validate PUCCH configuration
classPUCCH = validatestring(class(pucch),{'nrPUCCH2Config','nrPUCCH3Config','nrPUCCH4Config'},''
formatPUCCH = classPUCCH(8);

% The temporary variables carrier_init and pucch_init are used to
% create the temporary variables carrier and pucch in the SNR loop
% to create independent instances in case of parallel simulation.
carrier_init = carrier;
pucch_init = pucch;

for snrIdx = 1:numel(snrIn) % Comment out for parallel computing
% parfor snrIdx = 1:numel(snrIn) % Uncomment for parallel computing
    % To reduce the total simulation time, you can execute this loop in
    % parallel by using Parallel Computing Toolbox features. Comment out the
    % for-loop statement and uncomment the parfor-loop statement. If
    % Parallel Computing Toolbox is not installed, parfor-loop defaults to
```

```
% a for-loop statement. Because the parfor-loop iterations are executed
% in parallel in a nondeterministic order, the simulation information
% displayed for each SNR point can be intertwined. To switch off the
% simulation information display, set the displaySimulationInformation
% variable (defined earlier in this example) to false.

% Reset the random number generator and channel so that each SNR point
% experiences the same noise and channel realizations.
rng(0,'twister')
reset(channel)

% Initialize variables for this SNR point (required when using
% Parallel Computing Toolbox)
carrier = carrier_init;
pucch = pucch_init;
pathFilters = [];

% Get operating SNR value
SNRdB = snrIn(snrIdx);

% Get total number of slots in the simulation period
NSlots = nFrames*slotsPerFrame;

% Set timing offset, which is updated in every slot for perfect
% synchronization and when correlation is strong for practical
% synchronization
offset = 0;

% Set variable to store block errors for each SNR point with 0
ucierr = 0;
for nslot = 0:NSlots-1

    % Update carrier slot number to account for new slot transmission
    carrier.NSlot = nslot;

    % Get PUCCH resources
    [pucchIndices,pucchIndicesInfo] = nrPUCCHIndices(carrier,pucch);
    dmrsIndices = nrPUCCHDMRSIndices(carrier,pucch);
    dmrsSymbols = nrPUCCHDMRS(carrier,pucch);

    % Create random UCI bits
    uci = randi([0 1],ouci,1);

    % Perform UCI encoding
    codedUCI = nrUCIEncode(uci,pucchIndicesInfo.G);

    % Perform PUCCH modulation
    pucchSymbols = nrPUCCH(carrier,pucch,codedUCI);

    % Create resource grid associated with PUCCH transmission antennas
    pucchGrid = nrResourceGrid(carrier,nTxAnts);

    % Perform implementation-specific PUCCH MIMO precoding and mapping
    F = eye(1,nTxAnts);
    [~,pucchAntIndices] = nrExtractResources(pucchIndices,pucchGrid);
    pucchGrid(pucchAntIndices) = pucchSymbols*F;

    % Perform implementation-specific PUCCH DM-RS MIMO precoding and mapping
```

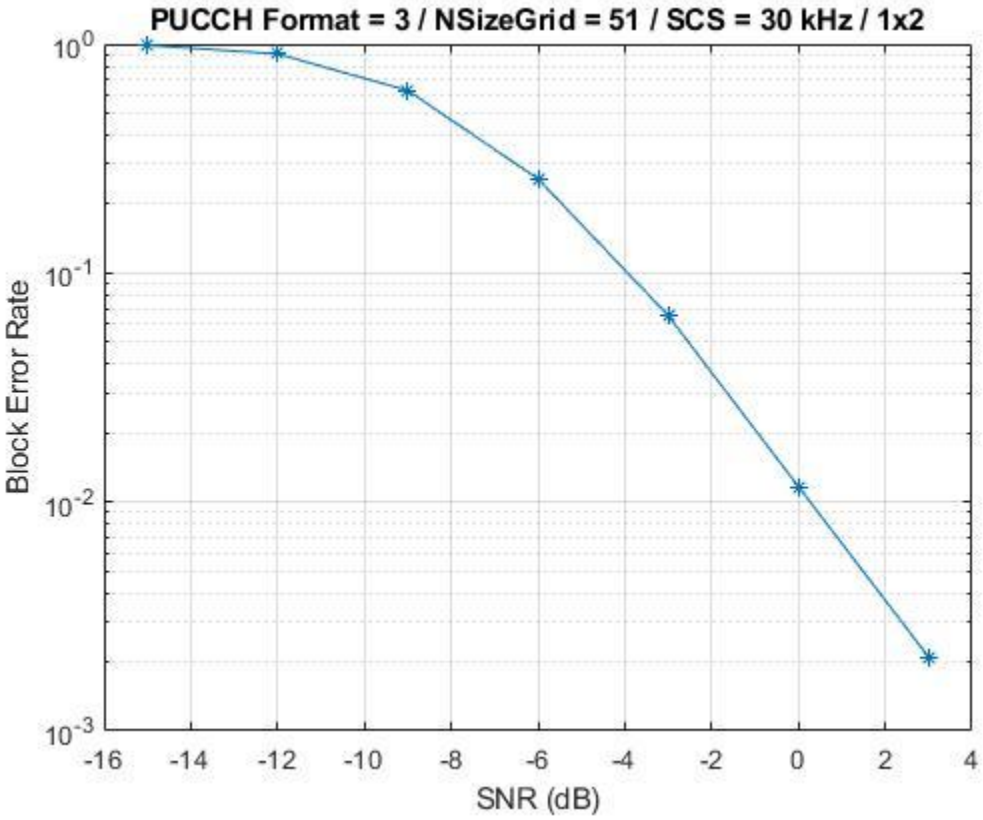
```
[~,dmrsAntIndices] = nrExtractResources(dmrsIndices,pucchGrid);
pucchGrid(dmrsAntIndices) = dmrsSymbols*F;

% Perform OFDM modulation
txWaveform = nrOFDMModulate(carrier,pucchGrid);

% Pass data through the channel model. Append zeros at the end of
% the transmitted waveform to flush the channel content. These
% zeros take into account any delay introduced in the channel. This
% delay is a combination of the multipath delay and implementation
% delay. This value can change depending on the sampling rate,
% delay profile, and delay spread.
txWaveformChDelay = [txWaveform; zeros(chInfo.MaximumChannelDelay,size(txWaveform,2))];
[rxWaveform,pathGains,sampleTimes] = channel(txWaveformChDelay);

% Add AWGN to the received time domain waveform. Normalize the
% noise power by the size of the inverse fast Fourier transform
% (IFFT) used in OFDM modulation, because the OFDM modulator
% applies this normalization to the transmitted waveform. Also,
% normalize the noise power by the number of receive antennas,
% because the default behavior of the channel model is to apply
% this normalization to the received waveform.
SNR = 10^(SNRdB/20);
N0 = 1/(sqrt(2.0*nRxAnts*nFFT)*SNR);
noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

% Perform synchronization
if perfectChannelEstimator == 1
    % For perfect synchronization, use the information provided by
    % the channel to find the strongest multipath component.
    pathFilters = getPathFilters(channel);
    [offset,mag] = nrPerfectTimingEstimate(pathGains,pathFilters);
else
    % For practical synchronization, correlate the received
    % waveform with the PUCCH DM-RS to give timing offset estimate
    % t and correlation magnitude mag. The function hSkipWeakTimingOffset
    % is used to update the receiver timing offset. If the correlation
    % peak in mag is weak, the current timing estimate t is ignored
    % and the previous estimate offset is used.
    [t,mag] = nrTimingEstimate(carrier,rxWaveform,dmrsIndices,dmrsSymbols);
    offset = hSkipWeakTimingOffset(offset,t,mag);

    % Display a warning if the estimated timing offset exceeds the
    % maximum channel delay
    if offset > chInfo.MaximumChannelDelay
        warning(['Estimated timing offset (%d) is greater than the maximum channel delay
            ' This will result in a decoding failure. This may be caused by low SNR,' ..
            ' or not enough DM-RS symbols to synchronize successfully.'],offset,chInfo.Ma
    end
end
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation on the received data to recreate the
% resource grid. Include zero padding in the event that practical
% synchronization results in an incomplete slot being demodulated.
rxGrid = nrOFDMDemodulate(carrier,rxWaveform);
[K,L,R] = size(rxGrid);
```

```
            if (L < symbolsPerSlot)
                rxGrid = cat(2,rxGrid,zeros(K,symbolsPerSlot-L,R));
            end

            % Perform channel estimation
            if perfectChannelEstimator == 1
                % For perfect channel estimation, use the value of the path
                % gains provided by the channel.
                estChannelGrid = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offset,sample

                % Get the perfect noise estimate (from the noise realization).
                noiseGrid = nrOFDMDemodulate(carrier,noise(1+offset:end,:));
                noiseEst = var(noiseGrid(:));

                % Apply MIMO deprecoding to estChannelGrid to give an
                % estimate per transmission layer.
                K = size(estChannelGrid,1);
                estChannelGrid = reshape(estChannelGrid,K*symbolsPerSlot*nRxAnts,nTxAnts);
                estChannelGrid = estChannelGrid*F.';
                estChannelGrid = reshape(estChannelGrid,K,symbolsPerSlot,nRxAnts,[]);
            else
                % For practical channel estimation, use PUCCH DM-RS.
                [estChannelGrid,noiseEst] = nrChannelEstimate(carrier,rxGrid,dmrsIndices,dmrsSymbols
            end

            % Get PUCCH REs from received grid and estimated channel grid
            [pucchRx,pucchHest] = nrExtractResources(pucchIndices,rxGrid,estChannelGrid);

            % Perform equalization
            [pucchEq,csi] = nrEqualizeMMSE(pucchRx,pucchHest,noiseEst);

            % Decode PUCCH symbols
            [uciLLRs,rxSymbols] = nrPUCCHDecode(carrier,pucch,ouci,pucchEq,noiseEst);

            % Decode UCI
            decucibits = nrUCIDecode(uciLLRs{1},ouci);

            % Store values to calculate BLER
            ucierr = ucierr + (~isequal(decucibits,uci));

        end

        % Calculate UCI BLER for each SNR point
        blerUCI(snrIdx) = ucierr/NSlots;

        % Display results dynamically
        if displaySimulationInformation == 1
            fprintf(['UCI BLER of PUCCH format ' formatPUCCH ' for ' num2str(nFrames) ' frame(s) at S
        end
    end

UCI BLER of PUCCH format 3 for 5 frame(s) at SNR -14 dB: 0.66
UCI BLER of PUCCH format 3 for 5 frame(s) at SNR -12 dB: 0.36
UCI BLER of PUCCH format 3 for 5 frame(s) at SNR -10 dB: 0.22
UCI BLER of PUCCH format 3 for 5 frame(s) at SNR -8 dB: 0.1
UCI BLER of PUCCH format 3 for 5 frame(s) at SNR -6 dB: 0.04
UCI BLER of PUCCH format 3 for 5 frame(s) at SNR -4 dB: 0.02
```

```
% Plot results
figure
semilogy(snrIn,blerUCI,'-*')
grid on
xlabel('SNR (dB)')
ylabel('Block Error Rate')
title(sprintf('PUCCH Format = %s / NSizeGrid = %d / SCS = %d kHz / %dx%d',...
    formatPUCCH,carrier_init.NSizeGrid,carrier_init.SubcarrierSpacing,nTxAnts,nRxAnts))
```



This next figure shows the BLER results obtained by simulating 1500 frames (`NFrames = 1500`, `SNRIn = -15:3:3`) for a carrier with 30 kHz SCS occupying a 20 MHz transmission bandwidth. The simulation setup includes the default PUCCH format 3 configuration placed in the example with the number of UCI bits set to 16 and the `perfectChannelEstimator` variable set to `false`.



Further Exploration

- To analyze the UCI BLER at each SNR point, toggle the value of `perfectChannelEstimator` variable and change the range of SNR values.
- To check the BLER performance of different scenarios, change the carrier numerology, the number of transmit and receive antennas, and the channel model.
- To observe the BLER performance of PUCCH formats 2 and 4, use the `nrPUCCH2Config` object and the `nrPUCCH4Config` object, respectively, for the `pucch` variable in section Carrier and PUCCH Configuration on page 5-12.

Summary

The example demonstrates how to measure the UCI BLER when UCI is transmitted on PUCCH format 3. The example also displays and plots the BLER as a function of SNR.

References

1 3GPP TS 38.211. "NR; Physical channels and modulation (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

2 3GPP TS 38.212. "NR; Multiplexing and channel coding (Release 15)." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

3 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception (Release 15).” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Functions
`nrUCIDecode` | `nrUCIEncode` | `nrOFDMModulate` | `nrOFDMDemodulate`

Objects
`nrTDLChannel` | `nrCarrierConfig`

More About

- “SNR Definition Used in Link Simulations” on page 5-86

CDL Channel Model Customization with Ray Tracing

This example shows how to customize the CDL channel model parameters by using the output of a ray tracing analysis. The example shows how to:

- Specify the location of a transmitter and a receiver in a 3D environment.
- Use ray tracing to calculate the geometric aspects of the channel: number of rays, angles, delays and attenuations.
- Configure the CDL channel model with the result of ray tracing analysis.
- Specify the channel antenna arrays using Phased Array System Toolbox™.
- Visualize the transmit and receive array radiation patterns based on singular value decomposition of a perfect channel estimate.

Base Station and UE Configuration

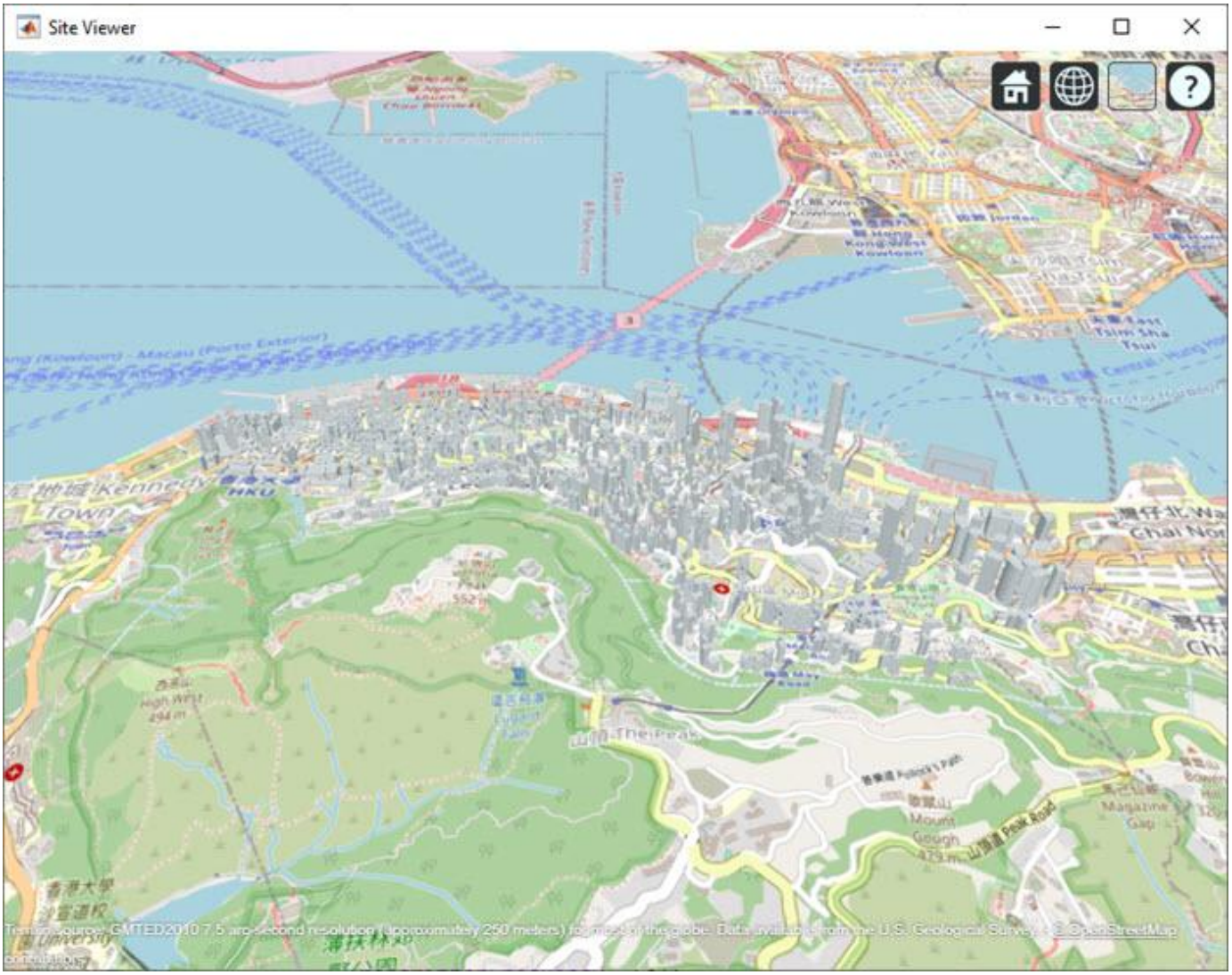
The example assumes that both the base station and the UE use rectangular arrays. The array orientations are specified as a pair of values representing azimuth and elevation. Both angles are in degrees.

```
fc = 6e9;                               % carrier frequency (Hz)
bsPosition = [22.287495, 114.140706]; % lat, lon
bsAntSize = [8 8];                      % number of rows and columns in rectangular array (base sta
bsArrayOrientation = [-30 0].';         % azimuth (0 deg is East, 90 deg is North) and elevation (p
uePosition = [22.287323,114.140859];  % lat, lon
ueAntSize = [2 2];                      % number of rows and columns in rectangular array (UE).
ueArrayOrientation = [180 45].';        % azimuth (0 deg is East, 90 deg is North) and elevation (p
reflectionsOrder = 1;                   % number of reflections for ray tracing analysis (0 for LOS

% Bandwidth configuration, required to set the channel sampling rate and for perfect channel esti
SCS = 15; % subcarrier spacing
NRB = 52; % number of resource blocks, 10 MHz bandwidth
```

Import and Visualize 3-D Environment with Buildings for Ray Tracing

Launch Site Viewer with buildings in Hong Kong. For more information about the osm file, see [1] on page 5-30.

```
if exist('viewer','var') && isvalid(viewer) % viewer handle exists and viewer window is open
    viewer.clearMap();
else
    viewer = siteviewer("Basemap","openstreetmap","Buildings","hongkong.osm");
end
```



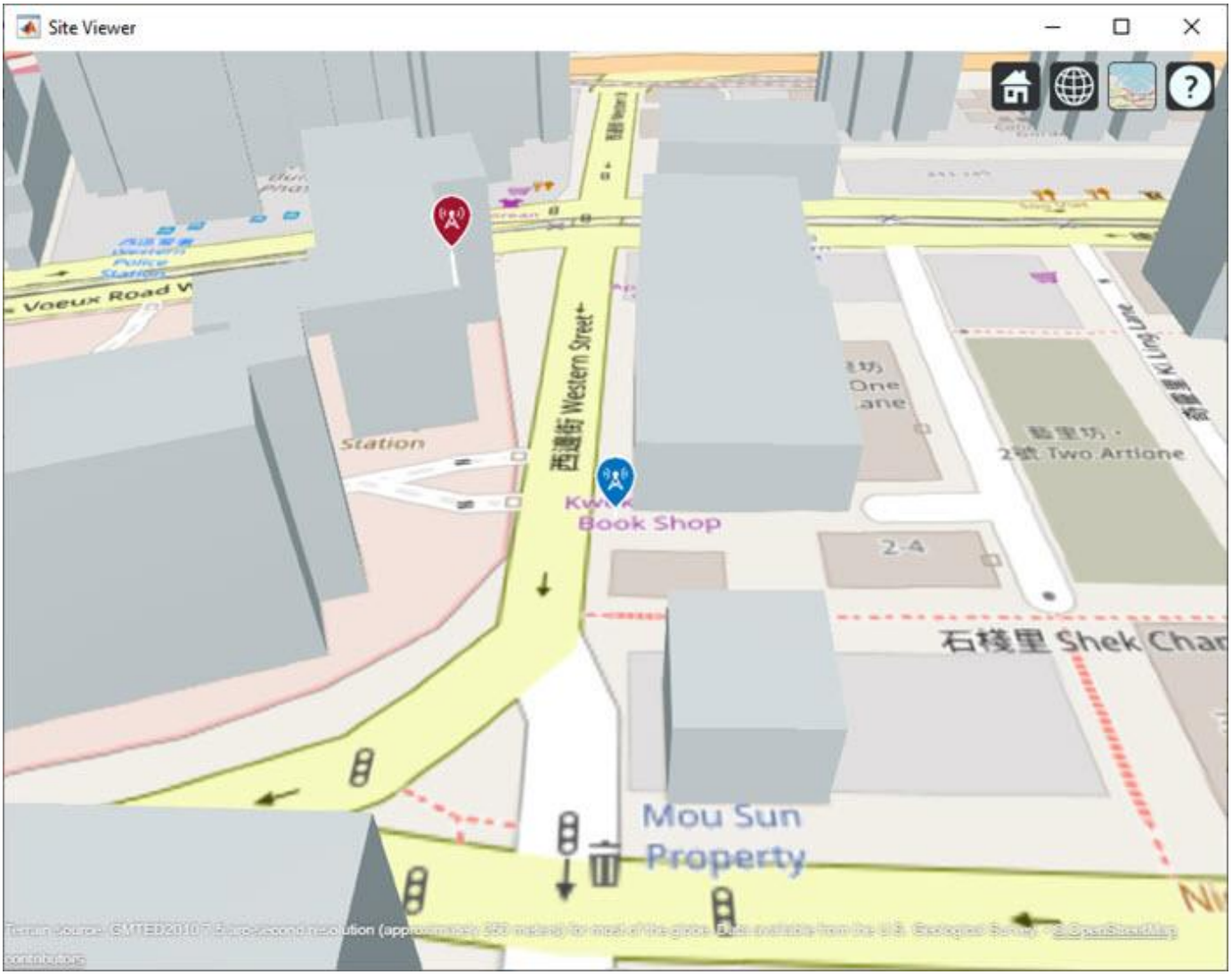
Create Base Station and UE

Locate the base station and the UE on the map.

```
bsSite = txsite("Name","Base station", ...
    "Latitude",bsPosition(1),"Longitude",bsPosition(2),...
    "AntennaAngle",bsArrayOrientation(1:2),...
    "AntennaHeight",4,...  % in m
    "TransmitterFrequency",fc);

ueSite = rxsite("Name","UE", ...
    "Latitude",uePosition(1),"Longitude",uePosition(2),...
    "AntennaHeight",1,... % in m
    "AntennaAngle",ueArrayOrientation(1:2));
```

Visualize the location of the base station and the UE. The base station is on top of a building.

```
bsSite.show();
ueSite.show();
```



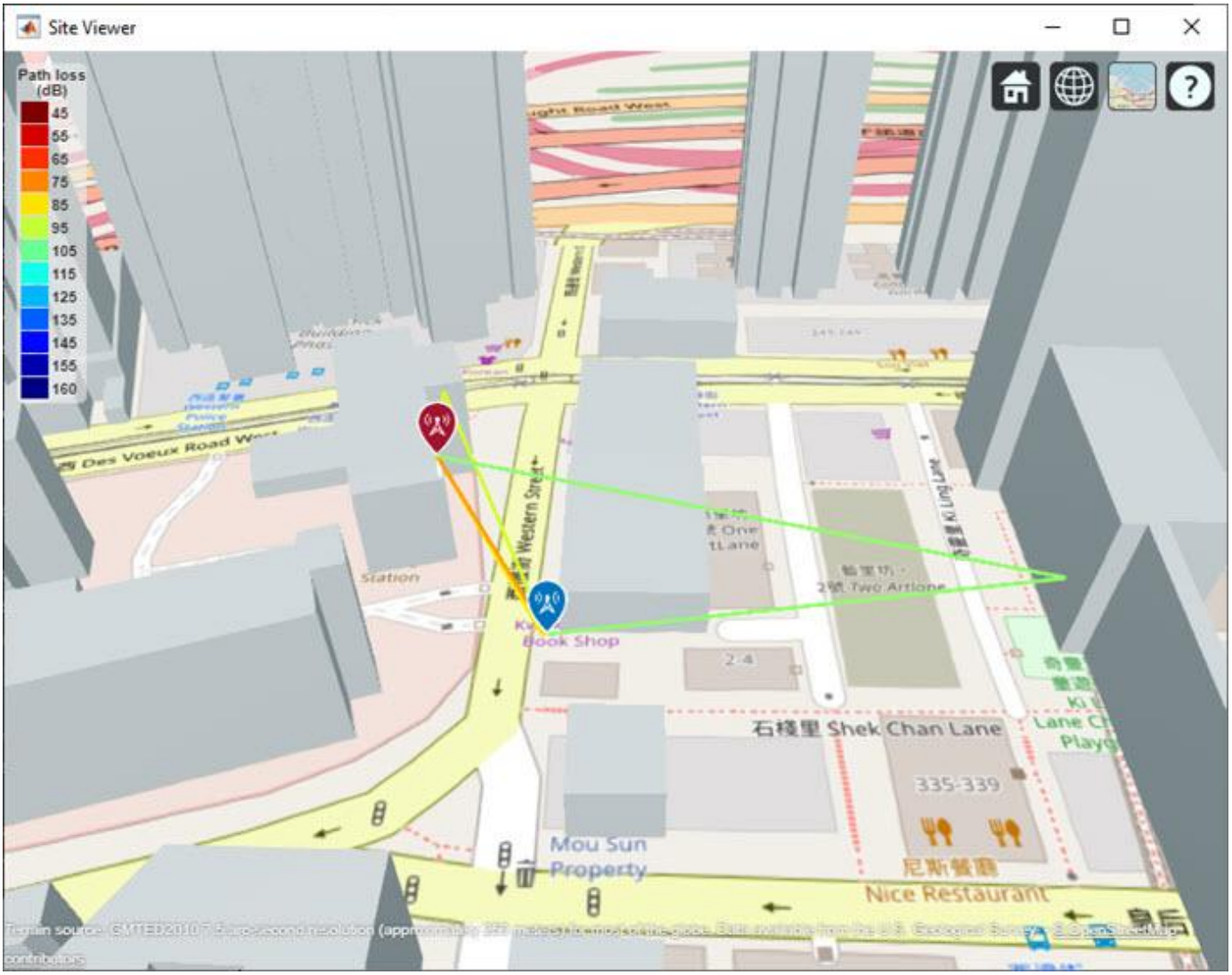
Ray Tracing Analysis

Perform ray tracing analysis using the shooting and bouncing rays (SBR) method. The SBR method includes effects from surface reflections and diffractions but does not include effects from refraction or scattering.

```
pm = propagationModel("raytracing","Method","sbr","MaxNumReflections",reflectionsOrder);
rays = raytrace(bsSite,ueSite,pm,"Type","pathloss");
```

Display the rays in Site Viewer.

```
plot(rays{1})
```



From the obtained rays, get the times of arrival, the average path gains, and the angles of departure and arrival. For simplicity, normalize the propagation delay so that the first path occurs at time 0 sec, corresponding to no delay. Use the path loss to obtain the average path gains.

```
pathToAs = [rays{1}.PropagationDelay]-min([rays{1}.PropagationDelay]);  % Time of arrival of each
avgPathGains  = -[rays{1}.PathLoss];                                    % Average path gains of e
pathAoDs = [rays{1}.AngleOfDeparture];                                  % AoD of each ray
pathAoAs = [rays{1}.AngleOfArrival];                                    % AoA of each ray
isLOS = any([rays{1}.LineOfSight]);                                     % Line of sight flag
```

Set Up CDL Channel Model

Configure the CDL channel model with the information generated by the ray tracing analysis. Set the `DelayProfile` property to `'Custom'` to specify the path delays, the average path gains and the angles of arrival and departure (both in azimuth and zenith).

When configuring the channel model, take into account that:

- The ray tracer finds individual rays between the base station and the UE, while the CDL channel models clusters of rays whose properties are determined by the cluster average path gain (`AveragePathGains`), average angles of arrival and departure (`AnglesAoA`, `AnglesZoA`, `AnglesAoD`, and `AnglesZoD`), and the spread of the rays in the cluster (`AngleSpreads`). The information retrieved from the ray tracing analysis for individual rays configures the cluster average values of the CDL channel.
- The ray tracer performs a static analysis, while the CDL channel models UE movement. Therefore, the CDL channel introduces small-scale fading.
- The path gains obtained from ray tracing are considered as average path gains. Therefore, because of fading, instantaneous channel path gains will differ from the average values, but over long simulations their mean value will match the specified average path gains when using isotropic antennas.
- The CDL channel uses zenith angles, while the ray tracing analysis returns elevation angles, therefore you must convert between the two.
- If any of the calculated rays is a line of sight (LOS) ray (no reflection), set the `HasLOSCluster` CDL channel property to `true`. For LOS cases, the CDL model splits the first path into two components, one being LOS and the other having a Rayleigh fading characteristic. This results in a combined Rician fading characteristic. Therefore, in LOS cases, when you specify N rays, the CDL channel models $N+1$ paths internally.

```
channel = nrCDLChannel;
channel.DelayProfile = 'Custom';
channel.PathDelays = pathToAs;
channel.AveragePathGains = avgPathGains;
channel.AnglesAoD = pathAoDs(1,:);       % azimuth of departure
channel.AnglesZoD = 90-pathAoDs(2,:);    % channel uses zenith angle, rays use elevation
channel.AnglesAoA = pathAoAs(1,:);       % azimuth of arrival
channel.AnglesZoA = 90-pathAoAs(2,:);    % channel uses zenith angle, rays use elevation
channel.HasLOSCluster = isLOS;
channel.CarrierFrequency = fc;
channel.NormalizeChannelOutputs = false; % do not normalize by the number of receive antennas, th
channel.NormalizePathGains = false;      % set to false to retain the path gains
```

Specify the channel antenna arrays by using Phased Array System Toolbox array objects. The array orientation properties of the CDL channel model use azimuth and downtilt while the `ueArrayOrientation` and `bsArrayOrientation` objects use azimuth and elevation. Therefore, convert the elevation to downtilt by changing the sign.

```
c = physconst('LightSpeed');
lambda = c/fc;

% UE array (single panel)
ueArray = phased.NRRectangularPanelArray('Size',[ueAntSize(1:2) 1 1],'Spacing', [0.5*lambda*[1 1]
ueArray.ElementSet = {phased.IsotropicAntennaElement};   % isotropic antenna element
channel.ReceiveAntennaArray = ueArray;
channel.ReceiveArrayOrientation = [ueArrayOrientation(1); (-1)*ueArrayOrientation(2); 0];  % the

% Base station array (single panel)
bsArray = phased.NRRectangularPanelArray('Size',[bsAntSize(1:2) 1 1],'Spacing', [0.5*lambda*[1 1]
bsArray.ElementSet = {phased.NRAntennaElement('PolarizationAngle',-45) phased.NRAntennaElement('P
channel.TransmitAntennaArray = bsArray;
channel.TransmitArrayOrientation = [bsArrayOrientation(1); (-1)*bsArrayOrientation(2); 0];   % th
```

Set Channel Sampling Rate

The signal going through the channel determines the channel sampling rate. Consider a signal with subcarrier spacing of 15 kHz and 52 resource blocks (RBs), equivalent to a bandwidth of 10 MHz. To obtain the sampling rate, call the `nrOFDMInfo` function.

```
ofdmInfo = nrOFDMInfo(NRB,SCS);

channel.SampleRate = ofdmInfo.SampleRate;
```

Channel Estimation

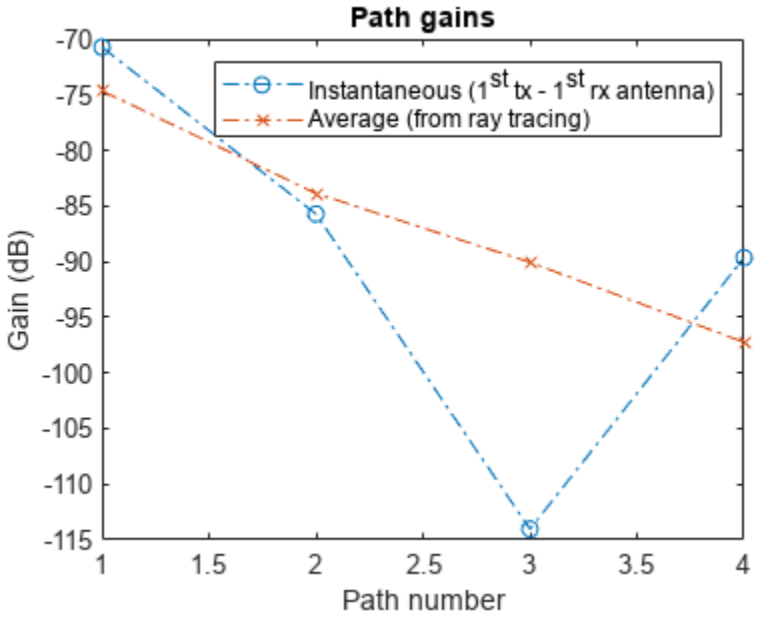
For simplicity, this example assumes perfect channel estimation. Setting the `ChannelFiltering` property to `false` allows you to get the channel path gains without sending a signal through the channel.

```
channel.ChannelFiltering = false;
[pathGains,sampleTimes] = channel();
```

Plot the path gains returned by the channel. Compare the results with the specified average path gains obtained from the ray attenuation values.

- For LOS cases, because the first two paths correspond to the first ray, the first two paths must be added together.
- The CDL channel model is a statistical channel model and considers UE motion. Therefore, the returned path gains are the instantaneous gains. The path gains from the ray tracing analysis are interpreted as average path gains by the channel model.
- The instantaneous path gains returned by the channel model include the gain of the antenna element in the direction of each ray. The custom path gains obtained from the ray tracing analysis do not include the antenna element gain. Therefore, in the mean, the channel path gains match the average gains only for isotropic antenna elements.

```
pg=permute(pathGains,[2 1 3 4]); % first dimension is the number of paths
if isLOS
    % in LOS cases sum the first to paths, they correspond to the LOS ray
    pg = [sum(pg(1:2,:,:,:)); pg(3:end,:,:,:)];
end
pg = abs(pg).^2;
plot(pow2db(pg(:,1,1,1)),'o-.');hold on
plot(avgPathGains,'x-.');hold off
legend("Instantaneous (1^{st} tx - 1^{st} rx antenna)","Average (from ray tracing)")
xlabel("Path number"); ylabel("Gain (dB)")
title('Path gains')
```

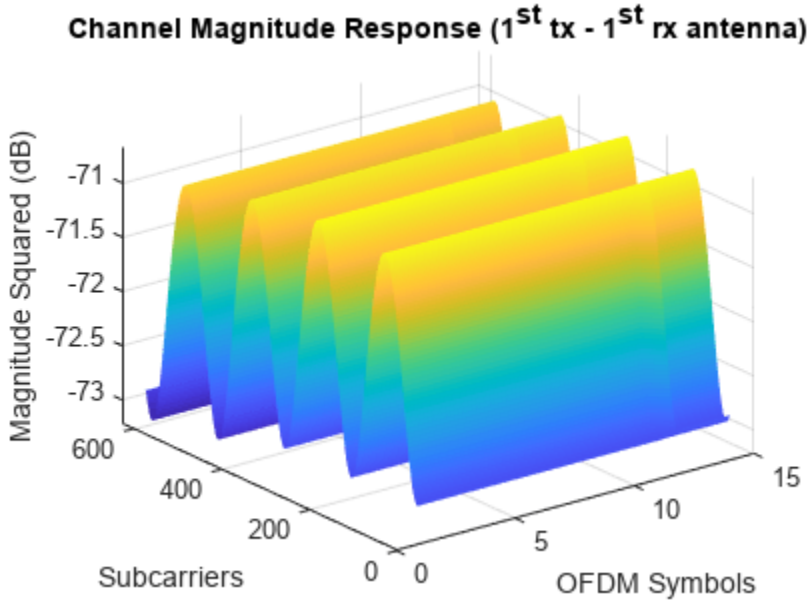


Obtain a perfect channel estimate for slot 0.

```
pathFilters = getPathFilters(channel);
nSlot = 0;
[offset,~] = nrPerfectTimingEstimate(pathGains,pathFilters);
hest = nrPerfectChannelEstimate(pathGains,pathFilters,NRB,SCS,nSlot,offset,sampleTimes);
```

Plot the channel response in time and frequency between the first transmit and the first receive antenna. This plot shows how the channel behaves in time and frequency. For low Doppler shifts, the channel doesn't change much during the observation period of one slot.

```
surf(pow2db(abs(hest(:,:,1,1)).^2));
shading('flat');
xlabel('OFDM Symbols');ylabel('Subcarriers');zlabel('Magnitude Squared (dB)');
title('Channel Magnitude Response (1^{st} tx - 1^{st} rx antenna)');
```



Get Beamforming Weights

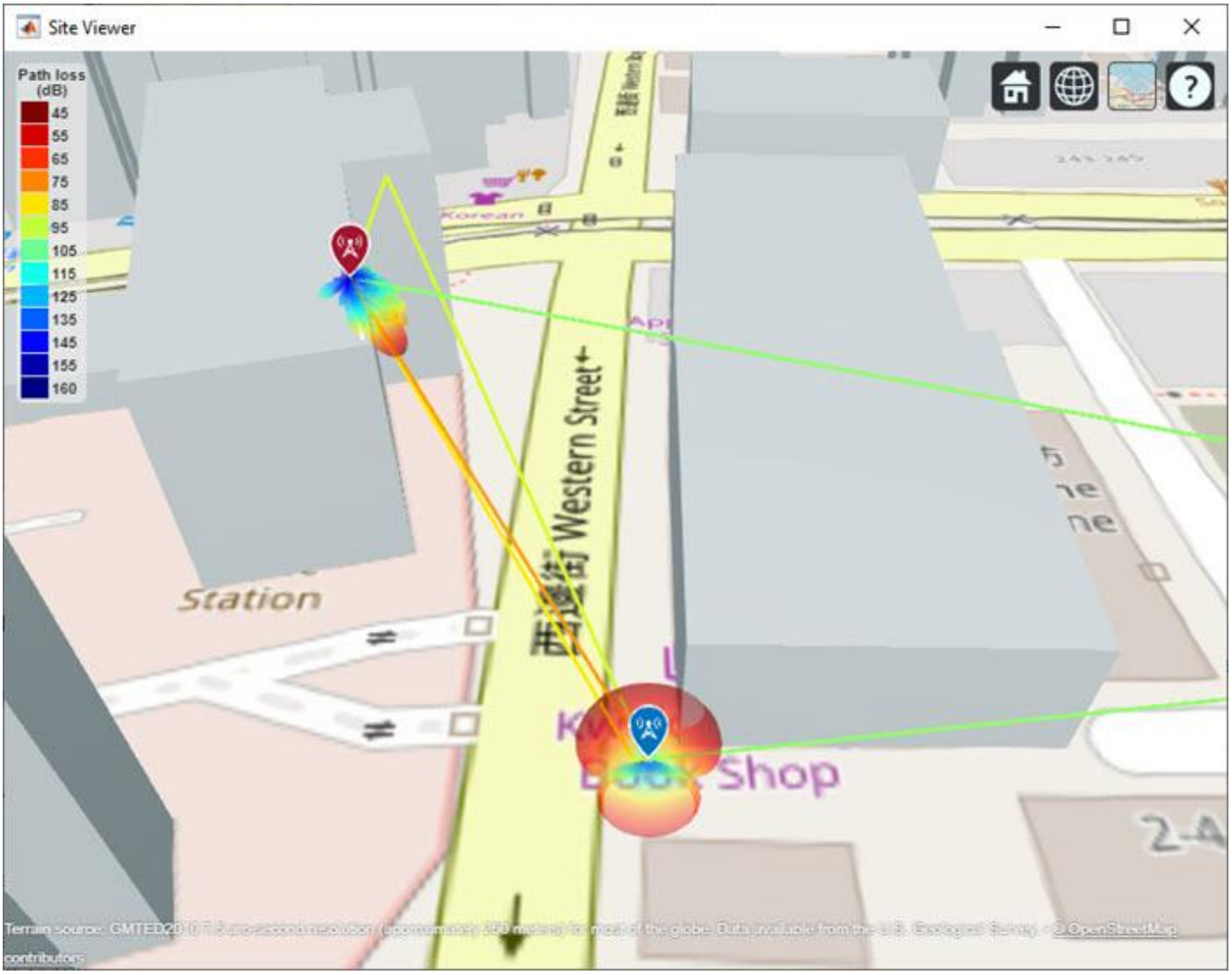
Calculate the beamforming weights using singular value decomposition (SVD). Assume 1 layer. The `getBeamformingWeights` function averages the channel conditions over a number of resource blocks, starting at an offset from the edge of the band (first subcarrier), enabling subband beamforming.

```
nLayers = 1;
scOffset = 0;   % no offset
noRBs = 1;      % average channel conditions over 1 RB to calculate beamforming weights
[wbs,wue,~] = getBeamformingWeights(hest,nLayers,scOffset,noRBs);
```

Plot Radiation Patterns

Plot the radiation patterns obtained for the UE and the base station.

```
% Plot UE radiation pattern
ueSite.Antenna = clone(channel.ReceiveAntennaArray); % need a clone, otherwise setting the Taper
ueSite.Antenna.Taper = wue;
pattern(ueSite,fc,"Size",4);

% Plot BS radiation pattern
bsSite.Antenna = clone(channel.TransmitAntennaArray); % need a clone, otherwise setting the Tape
bsSite.Antenna.Taper = wbs;
pattern(bsSite,fc,"Size",5);
```



References

[1] The osm file is downloaded from https://www.openstreetmap.org, which provides access to crowd-sourced map data all over the world. The data is licensed under the Open Data Commons Open Database License (ODbL), https://opendatacommons.org/licenses/odbl/.

Local Functions

```
function [wtx,wrx,D] = getBeamformingWeights(hEst,nLayers,scOffset,noRBs)
% Get beamforming weights given a channel matrix hEst and the number of
% layers nLayers. One set of weights is provided for the whole bandwidth.
% The beamforming weights are calculated using singular value (SVD)
% decomposition.
%
% Only part of the channel estimate is used to get the weights, this is
% indicated by an offset SCOFFSET (offset from the first subcarrier) and a
% width in RBs (NORBS).

% Average channel estimate
```

```
[~,~,R,P] = size(hEst);
%H = permute(mean(reshape(hEst,[],R,P)),[2 3 1]);

scNo = scOffset+1;
hEst = hEst(scNo:scNo+(12*noRBs-1),:,:,:);
H = permute(mean(reshape(hEst,[],R,P)),[2 3 1]);

% SVD decomposition
[U,D,V] = svd(H);
wtx = V(:,1:nLayers).';
wrx = U(:,1:nLayers)';
end
```

See Also

Functions
nrPerfectChannelEstimate | nrOFDMInfo

Objects
nrCDLChannel | phased.URA

TDD Reciprocity-Based PDSCH MU-MIMO Using SRS

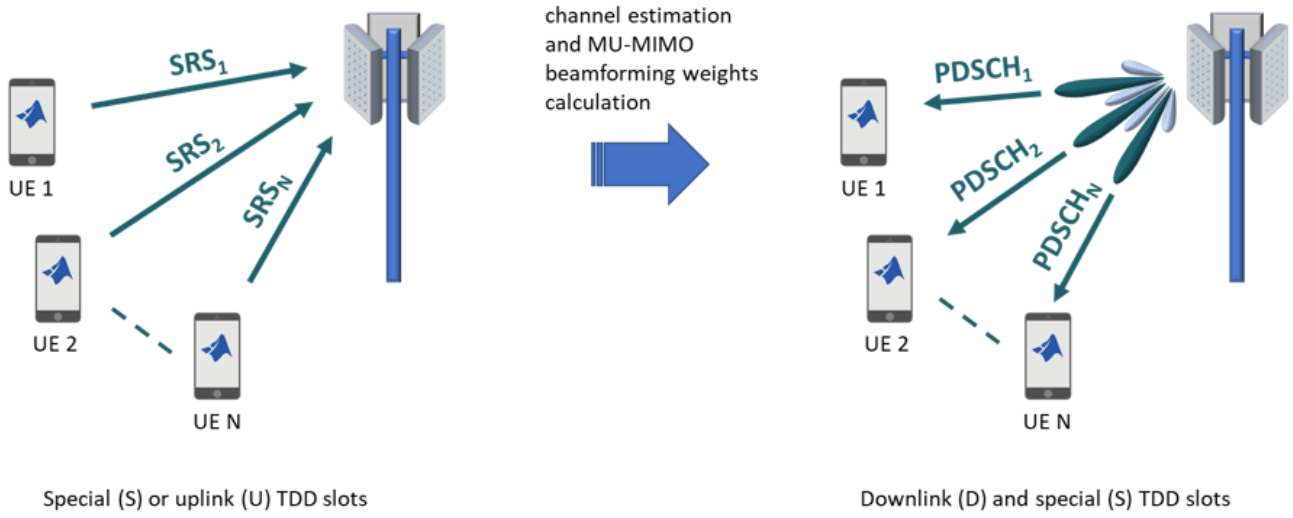
This example implements downlink multiuser multiple-input multiple-output (MU-MIMO) by exploiting channel reciprocity in a time division duplex (TDD) scenario. The example shows how to determine beamforming weights for physical downlink shared channel (PDSCH) transmission by using channel estimates based on uplink sounding reference signals (SRS) transmitted for each user, and how to schedule PDSCHs for multiple users in the same time and frequency resources.

Introduction

TDD systems use the same frequency band for uplink (UL) and downlink (DL) transmissions. The radio channel is reciprocal because it has the same characteristics in both UL and DL directions. Exploiting this reciprocity, you can use a UL transmission to obtain a channel estimate and then use this channel estimate to calculate parameters, including beamforming, for a DL transmission. This method is known as reciprocity-based beamforming.

This example implements downlink MU-MIMO by calculating a channel estimate for multiple users based on their SRS transmissions. Assuming reciprocity, the example then uses these channel estimates to select a set of users to be scheduled for PDSCH transmission and calculates DL beamforming weights for PDSCH transmissions to those users. When the base station has a sufficient number of antennas, it is possible to beamform PDSCH transmissions for a set of users in the same time and frequency resources such that the users suffer little interference from each other.

This example schedules SRS transmissions for all UEs in the UL part of the special slot, and schedules PDSCH transmissions for UEs chosen by the user selection algorithm in DL slots and the DL part of special slots.



The example offers a choice of MU-MIMO precoding algorithms and allows the number of layers scheduled in the same time and frequency resources to be configured. Finally, the example measures the capacity of the overall cell relative to a single-layer user, showing that the capacity is higher than could be achieved by scheduling a single user.

Simulation Assumptions

The example uses these assumptions.

- The SRS configuration allows for up to 16 UEs to be scheduled.
- The number of layers assigned to a given UE is random (either 1, 2, or 4) and is fixed for the duration of the simulation.
- The number of SRS antenna ports is equal to the number of layers assigned.
- For the zero forcing, regularized zero forcing, and block diagonalization precoding methods, the number of SRS antenna ports is equal to the number of physical antennas, that is, no spatial filtering of SRS ports onto a larger set of physical antennas. For joint spatial division multiplexing precoding, hybrid precoding is used so the number of SRS antenna ports is equal to the number of RF chains, which is less than the number of physical antennas.
- The number of UE transmit antennas is equal to the number of SRS antenna ports, that is, full channel sounding is used.
- The precoding resource block group (PRG) bundle size is the same for all UEs.
- PDSCH is not scheduled in the downlink slots of the first frame occurring prior to the first special slot, because no channel estimate is yet available.
- Demodulaton reference signal (DM-RS) configuration type is type 1 and DM-RS length is 2, that is, 8 orthogonal DM-RS ports can be scheduled. For scheduling more users, the example employs nonorthogonal DM-RSs by changing the scrambling identity.
- HARQ is not supported.
- All UEs are synchronized to within the cyclic prefix length based on the initial acquisiton and transmission timing adjustment procedures, as described in TS 38.213 Sections 8 and 4.2 [1 on page 5-42]. A timing offset within the cyclic prefix length manifests as a phase shift in the frequency domain and is compensated by the channel estimation and equalization.
- All UEs are received with equal power after averaging across channel variations due to fading.

Simulation Parameters

Specify the number of UEs, the number of frames to simulate, and the SNR for the downlink and the uplink, assuming that the SNR is the same for all users. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
numUEs = 12;        % Total number of UEs served by the base station
numFrames = 1;      % Total number of frames to simulate
SNRdBDL = 25;       % Downlink SNR in dB
SNRdBUL = 20;       % Uplink SNR in dB
```

Base Station Configuration

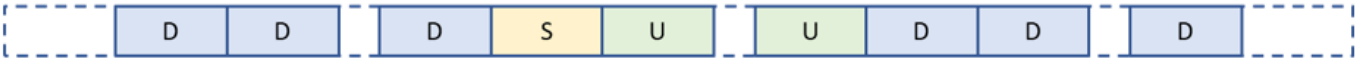
Carrier Configuration

Set the carrier parameters, specifying 51 resource blocks at 30 kHz subcarrier spacing, which corresponds to a channel bandwidth of 20 MHz.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 51;
carrier.SubcarrierSpacing = 30;
```

TDD Configuration

The example considers a TDD configuration consisting of a number of downlink slots (D), followed by a special slot (S), followed by a number of uplink slots (U). The special slot represents a switchpoint from downlink to uplink transmission and therefore contains downlink symbols, followed by empty guard symbols, followed by uplink symbols. The number of dowlink or uplink symbols in the special slot can be zero. This example schedules SRS transmissions in the uplink part of the special slot, and schedules PDSCH transmissions in downlink slots and in the downlink part of special slots.



Represent the TDD slot pattern by an array of strings specifying the slot type.

```
tddPattern = ["D" "D" "D" "D" "D" "D" "D" "S" "U" "U"];

disp("Cyclic slot pattern:")
```

```
Cyclic slot pattern:
```

```
disp("Slot " + string((0:length(tddPattern)-1)') + ": " + tddPattern(:))
```

```
    "Slot 0: D"
    "Slot 1: D"
    "Slot 2: D"
    "Slot 3: D"
    "Slot 4: D"
    "Slot 5: D"
    "Slot 6: D"
    "Slot 7: S"
    "Slot 8: U"
    "Slot 9: U"
```

Represent the special slot configuration by a three-element vector specifying the number of downlink symbols, guard symbols, and uplink symbols in the special slot.

```
specialSlot = [6 4 4];
```

The total number of symbols in the special slot configuration must match the number of symbols in a slot given by `carrier.SymbolsPerSlot`.

```
if(sum(specialSlot)~=carrier.SymbolsPerSlot)
    error(['specialSlot must contain ' num2str(carrier.SymbolsPerSlot) ' symbols.']);
end
```

Antenna Array Size

Specify the antenna array size for the base station (BS). Assume a rectangular array. The antenna array size is a vector `[M N P]`, where `M` is the number of rows, `N` is the number of columns and `P` is the number of polarizations in the antenna array.

```
bsAntSize = [2 8 2];
```

UE Configuration

Spatial Configuration

Specify if UE locations are randomly scattered throughout the cell sector or clustered into groups. To scatter randomly, set `numGroups` to 1. If clustering into groups, the number of UEs will be evenly split as much as possible amongst the groups. The number of groups must not exceed the number of UEs.

```
numGroups = 1;

% Assign each UE a spatial group
g = (1:numGroups).*ones(ceil(numUEs/numGroups),1);
groups = g(1:numUEs);
```

Number of Layers and Antenna Array Sizes

Randomly select the number of layers per UE (1, 2, or 4).

```
% Reset random generator for reproducibility
rng('default');

% Number of layers for each UE
numLayers = pow2(randi([0 2],[1 numUEs]))
```

```
numLayers = 1×12

     4     4     1     4     2     1     1     2     4     4     1     4
```

The number of layers determines the antenna array size for each UE according to this table.

Number of Layers	UE Antenna Array Size
1	[1 1 1]
2	[1 1 2]
4	[1 2 2]

```
% Number of rows, columns and polarizations in rectangular array for each UE
ueAntSizes = 1 + (numLayers.' > [4 2 1])
```

```
ueAntSizes = 12×3

     1     2     2
     1     2     2
     1     1     1
     1     2     2
     1     1     2
     1     1     1
     1     1     1
     1     1     2
     1     2     2
     1     2     2
  ⋮
```

SRS Configuration

Configure the SRS parameters for each UE. The SRSs for all UEs are configured to be transmitted in a special slot, or in an uplink slot if there is no special slot (or if it contains no uplink symbols). The example performs time-division multiplexing (TDM) of SRSs by using different OFDM symbol numbers and frequency-division multiplexing (FDM) by using different comb offsets. For each UE, the number of SRS ports is configured to be equal to the number of layers for that UE.

```
SRSs = hMultiUserSRS(tddPattern,specialSlot,carrier,numLayers);
```

For more information on SRS parameterization, see the “NR SRS Configuration” on page 2-59 example.

PDSCH and DL-SCH Configuration

Configure the PDSCH and DL-SCH parameters for each UE with these parameters.

- Rate 3/4 coding and 64-point quadrature amplitude modulation (64-QAM)
- DM-RS configuration type 1 and DM-RS length 2, which provides 8 orthogonal DM-RS ports, see TS 38.211 Table 7.4.1.1.2-1 [5 on page 5-43]
- PRG bundle size of 2, see TS 38.214 Section 5.1.2.3 [6 on page 5-43]

Configure the appropriate number of layers for each UE and set the RNTI equal to the UE number (`1...numUEs`).

```
PDSCHs = hMultiUserPDSCH(numLayers);
```

You can configure further PDSCH parameters for each transmission during the user selection process by using the `hMultiUserSelection` function.

For more information on PDSCH parameterization, see the “NR PDSCH Resource Allocation and DM-RS and PT-RS Reference Signals” on page 1-15 example.

Algorithmic Parameters

MU-MIMO Precoding Method

The example supports these precoding methods.

- `'BD'`: Block diagonalization, as described by Spencer, Q.H., et al. in [2 on page 5-43]. Note that the implementation uses the `blkdiagbfweights` function.
- `'ZF'`: Zero forcing, as described by Peel, C.B., et al. in [3 on page 5-43].
- `'RZF'`: Regularized zero forcing, as described by Peel, C.B., et al. in [3 on page 5-43].
- `'JSDM-JGP'`: Joint spatial division multiplexing with joint group processing (JGP), as described by Adhikary, A., et al. in [7 on page 5-43].
- `'JSDM-PGP'`: Joint spatial division multiplexing with per-group processing (PGP), as described by Adhikary, A., et al. in [7 on page 5-43].

Configure the precoding method that is used to determine the beamforming weights for each PDSCH during user selection.

```
algParameters = struct;
algParameters.PrecodingMethod = 'RZF';
```

Configure the number of layers that the user selection algorithm schedules in each PRG in each slot. This value is the total number of layers scheduled across all users in any given PRG in this slot. The user selection algorithm is based on proportional fair (PF) scheduling considering each PRG separately.

```
algParameters.ScheduledLayers = 8;
```

Hybrid Precoding using Joint Spatial Division Multiplexing (JSDM)

This example provides an option to use JSDM for the MU-MIMO precoding method. JSDM enables massive MIMO by using a hybrid precoding architecture where the number of RF chains is much less than the number of physical antennas. In this example, the number of RF chains equals the number of scheduled layers when JSDM is selected. For more information on JSDM, see the “Massive MIMO Hybrid Beamforming” example. Note that the implementation uses the `jsdmrfweights` (Phased Array System Toolbox) and `jsdmbbweights` (Phased Array System Toolbox) functions which require the Phased Array System Toolbox.

To accommodate the needs of JSDM, the following parameters changes are recommended:

- Increase the number of antennas. Massive MIMO generally implies antenna arrays larger than 64 elements.
- Limit the number of groups to 2-4 (at least two groups are required). JSDM allocates a fixed number of layers to each group, which may affect PDSCH scheduling and cause some UEs to not receive data due to the PF scheduling if there aren't enough layers in the group. In this example, the number of layers assigned to each group is split evenly from the number of scheduled layers (the last group will receive the leftover layers if the layers cannot be split evenly).
- JGP has better performance than PGP, because the PGP precoding matrix is truncated to reflect less CSI.
- Limit the number of scheduled layers to be less than the total number of UEs (for PGP) or less than the total number of UE layers (for JGP). JSDM RF beamforming is sensitive to the rank measurements of the group spatial covariances. In this example, the measured rank of each group is limited to the number of users.
- Adjusting the modulation order can lower the bit error rate in exchange for lower throughput. The default is set to 64-QAM.

Upon selection of JSDM, the example will enable channel filtering and disable perfect channel estimation.

Channel Estimator Configuration

The logical variable `PerfectChannelEstimator` controls channel estimation behavior. When you set this variable to `true`, the example uses perfect channel estimation for SRS and PDSCH reception. Otherwise, the example uses practical channel estimation based on the values of the received SRS and PDSCH DM-RS.

```
algParameters.PerfectChannelEstimator = false;
```

CDL Channel Models

Configure a propagation channel model for each UE. The `hMultiUserChannels` function uses the `nrCDLChannel` object to configure clustered delay line (CDL) channels. Use this function to create an initial channel object with CDL-A delay profile, 100 ns delay spread, and 5 Hz Doppler shift. The function then configures different channels for each UE by modifying the initial object to:

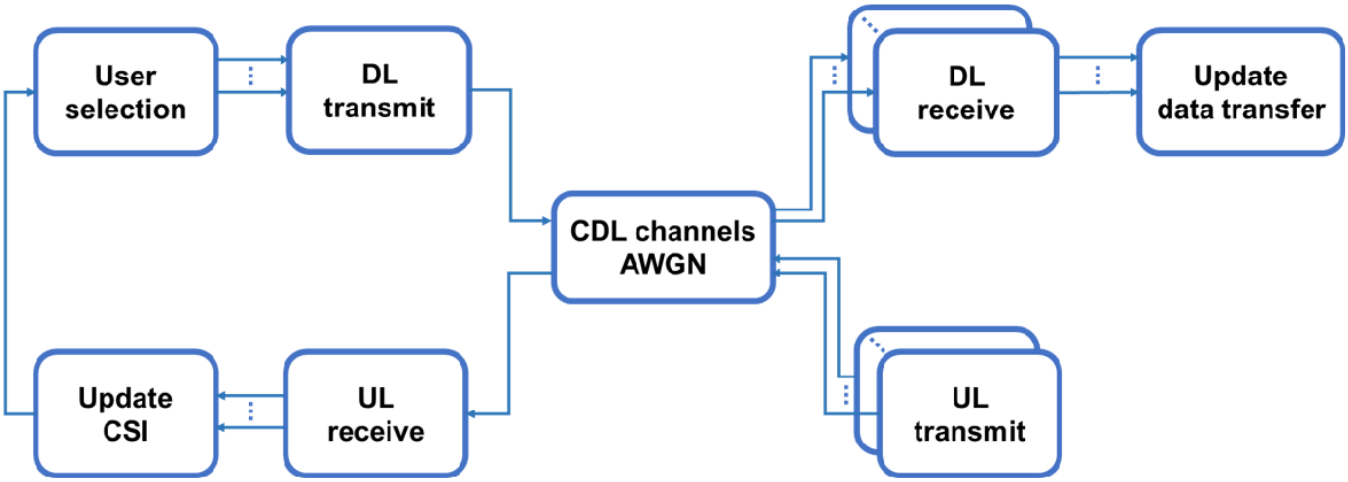
- Simulate the effect of different UEs in different locations in the environment around the base station. Each UE has an azimuth and zenith angle relative to the base station; the angles of departure for each cluster are offset by these angles. When `numGroups=1`, the UEs are configured to have randomly selected azimuth and zenith angles; otherwise, the UEs are configured to be clustered in spatial groups such that the azimuth and zenith angles for the grouped UEs are similar.
- Configure the UE antenna array according to the antenna array size chosen in Number of Layers and Antenna Array Sizes on page 5-35.
- Set a different value of the `Seed` property. This ensures that the channel for each UE has independent fast fading.

```
delayProfile = 'CDL-A';
delaySpread = 100e-9;
maximumDopplerShift = 5;
channels = hMultiUserChannels(delayProfile,delaySpread,maximumDopplerShift,bsAntSize,ueAntSizes,g
```

To visualize the configuration of a CDL channel, see the “Visualize CDL Channel Model Characteristics” on page 5-8 example. To create CDL channel model parameters by using the output of a ray tracing analysis, see the “CDL Channel Model Customization with Ray Tracing” on page 5-22 example.

MU-MIMO Throughput Simulation Using Reciprocity-Based Beamforming

This diagram shows the structure of the throughput simulation.



For slots scheduled for SRS:

- **UL transmit:** Generate the SRS waveform for each UE.
- **CDL channels, AWGN:** For each UE, send the waveform through the channel, combine the channel outputs at the base station, and add noise.
- **UL receive:** For each UE, perform SRS-based channel and noise estimation. For more information, see the “NR Uplink Channel State Information Estimation Using SRS” on page 2-78 example.
- **Update CSI:** Record the channel and noise estimates for use in subsequent slots.

For slots scheduled for PDSCH:

- **User selection:** Considering the most recent CSI for all UEs, determine which UEs to schedule for PDSCH and configure PDSCH parameters (allocated RBs, beamforming, DM-RS ports, scrambling identity).
- **DL transmit:** For each UE, perform DL-SCH encoding of data and generate and beamform the PDSCH. Sum the beamformed PDSCHs to form the transmitted signal.
- **CDL channels, AWGN:** Send the transmitted signal through the channel for each UE and add noise to create the received waveform for each UE.
- **DL receive:** For each UE, demodulate the received waveform, demodulate the PDSCH and decode the DL-SCH.
- **Update data transfer:** Record the transport block size (TBS) and cyclic redundancy check (CRC) for each UE, which the example uses to calculate final throughput results.

The `diagnosticsOn` flag controls the display of diagnostic information during the simulation. When set to `true`, the example displays this information:

- The current slot number and the type of the slot (D, S, or U)
- For slots containing SRS, a list of the UEs for which SRS is transmitted and received
- For slots containing PDSCH, a list of the UEs for which PDSCH is scheduled, the number of allocated resource blocks (NPRB), the EVM for each layer, and the CRC result from DL-SCH decoding

```
[numRF,channels,algParameters] = setupJSDM(algParameters,groups,numUEs,channels,bsAntSize);

% Set up record of data transfer and CSI
dataState = setupDataTransfer(carrier,numFrames,numLayers);
csi = setupCSI(carrier,bsAntSize,ueAntSizes);

diagnosticsOn = true;

% For each slot
for nSlot = 0:(carrier.SlotsPerFrame*numFrames)-1

    % Update the slot number
    carrier.NSlot = nSlot;

    % Display slot number and type (if diagnostics are enabled)
    if (diagnosticsOn)
        disp("Slot " + string(carrier.NSlot) + ": " + tddPattern(mod(carrier.NSlot,numel(tddPatte
    end

    % Schedule UEs for data transmission
    [schedule,PDSCHs,B] = hMultiUserSelection(csi,tddPattern,specialSlot,carrier,PDSCHs,bsAntSize

    % PDSCH transmissions for all UEs scheduled for data
    [txDL,txSymbols,singleLayerTBS] = hMultiDLTransmit(carrier,PDSCHs(schedule.PDSCH),numRF,B);

    % SRS transmissions for all UEs scheduled for SRS
    [txUL,schedule.SRS] = hMultiULTransmit(carrier,SRSs);

    % Apply fading channels
    [channels,rxDL,rxUL] = hApplyMultiUserChannels(tddPattern,specialSlot,carrier,schedule,channe

    % Apply AWGN
    rxDL = hApplyMultiUserAWGN(carrier,rxDL,SNRdBDL,CombineWaveforms=false);
```

```
        rxUL = hApplyMultiUserAWGN(carrier,rxUL,SNRdBUL,CombineWaveforms=true);

        % For all UEs scheduled for SRS, estimate CSI and record it
        [H,nVar] = hMultiULReceive(carrier,SRSs(schedule.SRS),rxUL,algParameters);
        csi = updateCSI(csi,carrier,schedule.SRS,H,nVar);

        % For all UEs scheduled for data, perform PDSCH reception and record the results
        [TBS,CRC,eqSymbols] = hMultiDLReceive(carrier,PDSCHs(schedule.PDSCH),rxDL,algParameters);
        dataState = updateDataTransfer(dataState,carrier,singleLayerTBS,schedule.PDSCH,TBS,CRC);

        % Display scheduled SRSs and PDSCHs, PDSCH EVM, and DL-SCH CRC (if diagnostics are enabled)
        if (diagnosticsOn)
            displayDiagnostics(schedule,PDSCHs,txSymbols,eqSymbols,CRC,groups);
        end

    end

    Slot 0: D
    Slot 1: D
    Slot 2: D
    Slot 3: D
    Slot 4: D
    Slot 5: D
    Slot 6: D
    Slot 7: S

    SRS transmission

    Slot 8: U
    Slot 9: U
    Slot 10: D

    Group:  1  1  1  1  1  1  1  1  1  1  1  1
    PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
     NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
      EVM: 18  5  5  4  2  6  6 14 11  8  7 11
           14  5     5  2        8  7  6     9
           12  5    13              7  6     6
           15  5     8              6  6     8
      CRC:  1  0  0  0  0  0  0  0  0  0  0  0

    Slot 11: D

    Group:  1  1  1  1  1  1  1  1  1  1  1  1
    PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
     NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
      EVM: 19  5  5  4  2  6  6 14 11  8  7 11
           15  5     5  2        8  7  5     8
           12  5    13              7  6     7
           15  6     8              6  6     8
      CRC:  1  0  0  0  0  0  0  0  0  0  0  0

    Slot 12: D

    Group:  1  1  1  1  1  1  1  1  1  1  1  1
    PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
     NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
      EVM: 18  5  5  4  2  6  6 14 10  8  7 11
           15  5     5  2        8  7  6     9
```

```
       11  5    13              7  6     7
       15  5     8              6  6     8
  CRC:  1  0  0  0  0  0  0  0  0  0  0  0

Slot 13: D

Group:  1  1  1  1  1  1  1  1  1  1  1  1
PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
 NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
  EVM: 17  5  5  4  2  6  6 13 10  7  7 11
       15  5     5  2        8  7  5     9
       12  5    13              7  6     6
       15  5     8              6  6     7
  CRC:  0  0  0  0  0  0  0  0  0  0  0  0

Slot 14: D

Group:  1  1  1  1  1  1  1  1  1  1  1  1
PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
 NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
  EVM: 17  5  5  4  2  6  6 14 11  8  8 11
       13  5     5  2        9  7  5     9
       11  5    13              7  6     7
       14  5     8              6  6     8
  CRC:  1  0  0  0  0  0  0  0  0  0  0  0

Slot 15: D

Group:  1  1  1  1  1  1  1  1  1  1  1  1
PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
 NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
  EVM: 16  6  5  4  2  6  6 13 11  7  8 12
       13  5     5  2        9  7  5     8
       11  5    13              7  6     7
       14  5     7              6  6     8
  CRC:  0  0  0  0  0  0  0  0  0  0  0  0

Slot 16: D

Group:  1  1  1  1  1  1  1  1  1  1  1  1
PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
 NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
  EVM: 18  5  6  4  2  6  6 13 10  8  9 12
       14  5     5  2        9  7  5     8
       12  5    13              7  6     7
       15  5     8              6  6     8
  CRC:  1  0  0  0  0  0  0  0  0  0  0  0

Slot 17: S

Group:  1  1  1  1  1  1  1  1  1  1  1  1
PDSCH:  1  2  3  4  5  6  7  8  9 10 11 12
 NPRB:  7  8 43  6 43 41 24 10  8  6 30  6
  EVM: 16  6  6  4  2  6  6 13 11  7  9 13
       14  5     5  2       10  7  5     9
       11  5    13              7  6     8
       14  6     8              6  6     9
  CRC:  0  0  0  0  0  0  0  0  0  0  0  0
SRS transmission
```

```
Slot 18: U
Slot 19: U
```

Throughput Results

Display a summary table of the throughput results, which shows the block error rate (BLER) and total bit throughput for each user.

```
results = summarizeResults(dataState);
disp(results);
```

```
    User    BLER     Throughput (bits)
    ____    _____    _________________

      1     0.625            42880
      2         0        1.5391e+05
      3         0         2.064e+05
      4         0        1.1462e+05
      5         0        4.1301e+05
      6         0        1.9539e+05
      7         0        1.1462e+05
      8         0             95800
      9         0        1.5391e+05
     10         0        1.1462e+05
     11         0        1.4278e+05
     12         0        1.1462e+05
```

Display the total throughput across all users.

```
totalThroughput = sum(results.("Throughput (bits)"));
dataRate = totalThroughput / (numFrames * 0.01) / 1e6;
disp(['Total throughput across all users: ' num2str(dataRate,'%0.3f') ' Mbps']);
```

```
Total throughput across all users: 186.254 Mbps
```

Display the average BLER across all users.

```
disp(['Average BLER across all users: ' num2str(mean(results.BLER,'omitnan'),'%0.3f')]);
```

```
Average BLER across all users: 0.052
```

Display the capacity relative to a single-layer user allocated to the entire carrier bandwidth. The maximum possible capacity equals the number of scheduled layers, given by `algParameters.ScheduledLayers`.

```
singleLayerThroughput = sum(dataState(1).SingleLayerTBS,'omitnan');
capacity = totalThroughput / singleLayerThroughput;
disp(['Capacity relative to a single-layer user: ' num2str(capacity,'%0.2f') 'x']);
```

```
Capacity relative to a single-layer user: 7.75x
```

The capacity exceeds the number of layers configured for the UEs (4 maximum, ~2.6 average), showing that MU-MIMO scheduling and precoding can successfully increase the capacity of the cell compared to making single-user transmissions.

References

1. 3GPP TS 38.213. "NR; Physical layer procedures for control." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

2. Spencer, Q.H., et al. "Zero-Forcing Methods for Downlink Spatial Multiplexing in Multiuser MIMO Channels." *IEEE Transactions on Signal Processing*, Vol. 52, No. 2, February 2004, pp. 461-471.

3. Peel, C.B., et al. "A Vector-Perturbation Technique for Near-Capacity Multiantenna Multiuser Communication—Part I: Channel Inversion and Regularization." *IEEE Transactions on Signal Processing*, Vol. 53, No. 1, February 2005, pp. 195-202.

4. 3GPP TS 38.101-4. "NR; User Equipment (UE) radio transmission and reception; Part 4: Performance requirements." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

5. 3GPP TS 38.211. "NR; Physical channels and modulation." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

6. 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

7. Adhikary, A., et al., "Joint Spatial Division and Multiplexing - The Large-Scale Array Regime." *IEEE Transactions on Information Theory*, Vol. 59, No. 10, October 2013, pp. 6441-6463.

Local Functions

```
function csi = setupCSI(carrier,bsAntSize,ueAntSizes)
% Set up record of CSI obtained via SRS

    NCRB = carrier.NSizeGrid + carrier.NStartGrid;
    numUEs = size(ueAntSizes,1);
    csi = repmat(struct('H',[],'nVar',[],'NSlot',[]),1,numUEs);
    P = prod(bsAntSize);

    for ue = 1:numUEs

        R = prod(ueAntSizes(ue,:));
        csi(ue).H = NaN([NCRB 1 P R]);
        csi(ue).nVar = NaN([NCRB 1 R]);
        csi(ue).NSlot = NaN([NCRB 1]);

    end

end

function dataState = setupDataTransfer(carrier,numFrames,numLayers)
% Set up record of PDSCH data transfer

    nSlots = carrier.SlotsPerFrame * numFrames;
    TBS = NaN(1,nSlots);
    CRC = NaN(1,nSlots);
```

```
        tput = zeros(1,nSlots);
        numUEs = numel(numLayers);
        dataState = repmat(struct('TBS',TBS,'SingleLayerTBS',TBS,'CRC',CRC,'Throughput',tput),1,numUE

    end

    function csi = updateCSI(csi,carrier,SRS,H,nVar)
    % Update record of CSI obtained via SRS

        for ue = SRS

            H_ue = H{ue==SRS};
            nVar_ue = nVar{ue==SRS};
            idx = find(all(~isnan(nVar_ue),3));
            csi(ue).H(idx,:,:,:) = H_ue(idx,:,:,:);
            csi(ue).nVar(idx,:,:) = nVar_ue(idx,:,:);
            csi(ue).NSlot(idx) = carrier.NSlot;

        end

    end

    function dataState = updateDataTransfer(dataState,carrier,singleLayerTBS,PDSCH,TBS,CRC)
    % Update record of PDSCH data transfer

        nSlot = carrier.NSlot;

        for ue = PDSCH

            TBS_ue = TBS{ue==PDSCH};
            CRC_ue = CRC{ue==PDSCH};
            dataState(ue).SingleLayerTBS(nSlot + 1) = singleLayerTBS;
            dataState(ue).TBS(nSlot + 1) = TBS_ue;
            dataState(ue).CRC(nSlot + 1) = CRC_ue;
            dataState(ue).Throughput(nSlot + 1) = TBS_ue .* (1 - CRC_ue);

        end

    end

    function results = summarizeResults(dataState)
    % Summarize simulation results

        numUEs = numel(dataState);
        BLER = zeros(numUEs,1);
        Throughput = zeros(numUEs,1);

        for ue = 1:numUEs

            CRC = dataState(ue).CRC;
            CRC = CRC(~isnan(CRC));
            BLER(ue) = sum(CRC) / numel(CRC);
            Throughput(ue) = sum(dataState(ue).Throughput);

        end

        User = (1:numUEs).';
        results = table(User,BLER,Throughput);
```

```
    results.Properties.VariableNames{3} = 'Throughput (bits)';

end

function displayDiagnostics(schedule,PDSCHs,txSymbols,eqSymbols,CRC,groups)
% Display diagnostic information

    dispfn = @(x,y)disp([sprintf('%5s: ',x) sprintf('%2d ',y)]);

    if (~isempty(schedule.PDSCH))

        numUEs = numel(schedule.PDSCH);
        maxLayers = 4;
        EVM = NaN(maxLayers,numUEs);
        NPRB = zeros(1,numUEs);

        for i = 1:numUEs

            ue = schedule.PDSCH(i);
            pdsch = PDSCHs(ue).Config;
            NPRB(i) = numel(pdsch.PRBSet);
            evm = comm.EVM;
            EVM(1:pdsch.NumLayers,i) = evm(txSymbols{i},eqSymbols{i});

        end

        dispfn('Group',groups(schedule.PDSCH));
        dispfn('PDSCH',schedule.PDSCH);
        dispfn('NPRB',NPRB);

        evmlabel = '  EVM: ';
        for i = 1:maxLayers
            if (i>1)
                evmlabel(:) = ' ';
            end
            if (~all(isnan(EVM(i,:))))
                disp([evmlabel strrep(sprintf('%2d ',round(EVM(i,:))),'NaN','  ')]);
            end
        end

        dispfn('CRC',[CRC{:}]);

    end

    if (~isempty(schedule.SRS))

        disp('SRS transmission');

    end

end

function [numRF,channels,algParameters] = setupJSDM(algParameters,groups,numUEs,channels,bsAntSi
% Modify parameters if JSDM is chosen as the precoding method

    if any(strcmp(algParameters.PrecodingMethod,{'JSDM-JGP','JSDM-PGP'}))
        algParameters.groups = groups;
        numGroups = max(groups);
```

```
            if numGroups < 2
                error('Specify more than one group for JSDM');
            end

            % The channel helper file allows for frequency-domain channel
            % estimation, but is not compatible for JSDM due to the effective
            % channel estimation. Set perfect channel estimation to false.
            algParameters.PerfectChannelEstimator = false;

            % Enable channel filtering for JSDM
            for u = 1:numUEs
                channels(u).channel.ChannelFiltering = true;
            end

            % One RF chain per layer
            numRF = algParameters.ScheduledLayers;
        else
            % One RF chain per antenna
            numRF = bsAntSize;
        end

    end
```

See Also

Functions
`nrPDSCHDMRS` | `nrPDSCHDMRSIndices`

Objects
`nrDLSCH` | `nrDLSCHDecoder` | `nrCarrierConfig`

More About

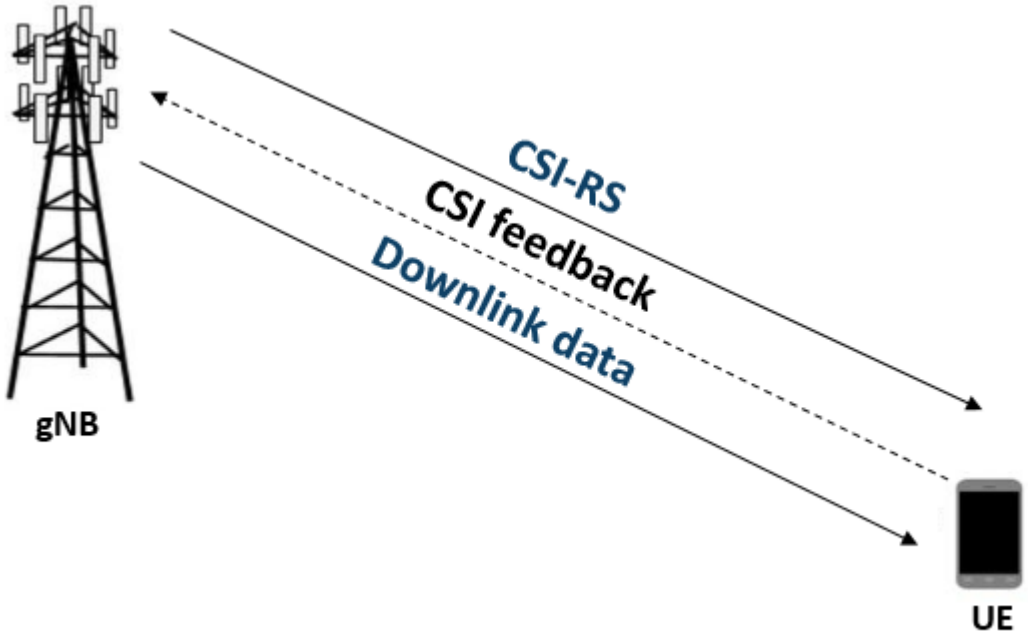
- “NR Uplink Channel State Information Estimation Using SRS” on page 2-78
- “NR SRS Configuration” on page 2-59
- “SNR Definition Used in Link Simulations” on page 5-86

5G NR Downlink CSI Reporting

This example shows how to compute downlink channel state information (CSI) parameters such as the channel quality indicator (CQI), precoding matrix indicator (PMI), and rank indicator (RI), for multiple input multiple output (MIMO) scenarios, as defined in TS 38.214 Section 5.2.2, over a tapped delay line (TDL) channel. The example supports CSI parameter computation for the type I single-panel, type I multi-panel, type II codebooks, and enhanced type II codebooks.

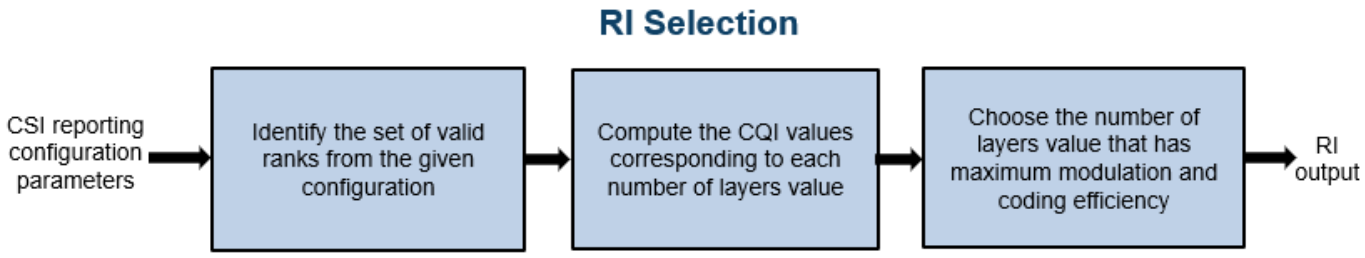
Introduction

CSI parameters are the quantities related to the state of a channel. The user equipment (UE) reports CSI parameters to the access network node (gNB) as feedback. The CSI feedback includes several parameters, such as the CQI, the PMI, and the RI. The UE uses the channel state information reference signal (CSI-RS) to measure the CSI feedback. Upon receiving the CSI parameters, the gNB schedules downlink data transmissions (such as modulation scheme, code rate, number of transmission layers, and MIMO precoding) accordingly. This figure shows an overview of CSI-RS transmission, CSI computation and feedback, and the transmission of downlink data that is scheduled based on the CSI parameters.



RI Selection

The RI defines the number of possible layers for the downlink transmission under specific channel conditions. The RI also corresponds to the maximum number of uncorrelated paths that the downlink transmission can use. Other CSI parameters like the PMI and CQI are computed based on the rank provided by the RI. For given channel conditions, the `hRISelect` function computes the CQI values for all valid values of number of transmission layers. The function returns the number of transmission layers that have maximum modulation and coding efficiency.



PMI Selection

The PMI consists of a set of indices corresponding to the precoding matrix. The gNB can apply this precoding matrix for the downlink data transmission. The `hDLPMISelect` function reports a precoding matrix by considering a codebook from these supported codebooks:

- Type I single-panel codebooks, as defined in TS 38.214 Tables 5.2.2.2.1-1 to 5.2.2.2.1-12
- Type I multi-panel codebooks, as defined in TS 38.214 Tables 5.2.2.2.2-1 to 5.2.2.2.2-6
- Type II codebooks, as defined in TS 38.214 Tables 5.2.2.2.2-1 to 5.2.2.2.3-5
- Enhanced type II codebooks, as defined in TS 38.214 Tables 5.2.2.2.5-1 to 5.2.2.2.5-6

The PMI selection is based on the codebook type, the number of transmission layers, and other CSI reporting configuration parameters such as antenna panel dimensions. Each codebook consists of a set of precoding matrices. NR considers a dual-stage precoding matrix based on the codebooks design from TS 38.214 Sections 5.2.2.2.1 to 5.2.2.2.3 and Section 5.2.2.2.5.

$$W = W_1 W_2$$

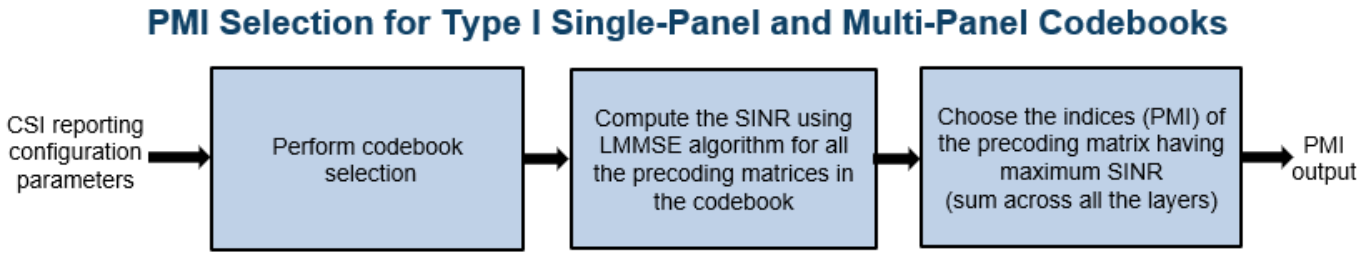
Where,

W_1 is a matrix representing a discrete Fourier transform (DFT) beam or beam group for both polarizations based on the codebook type

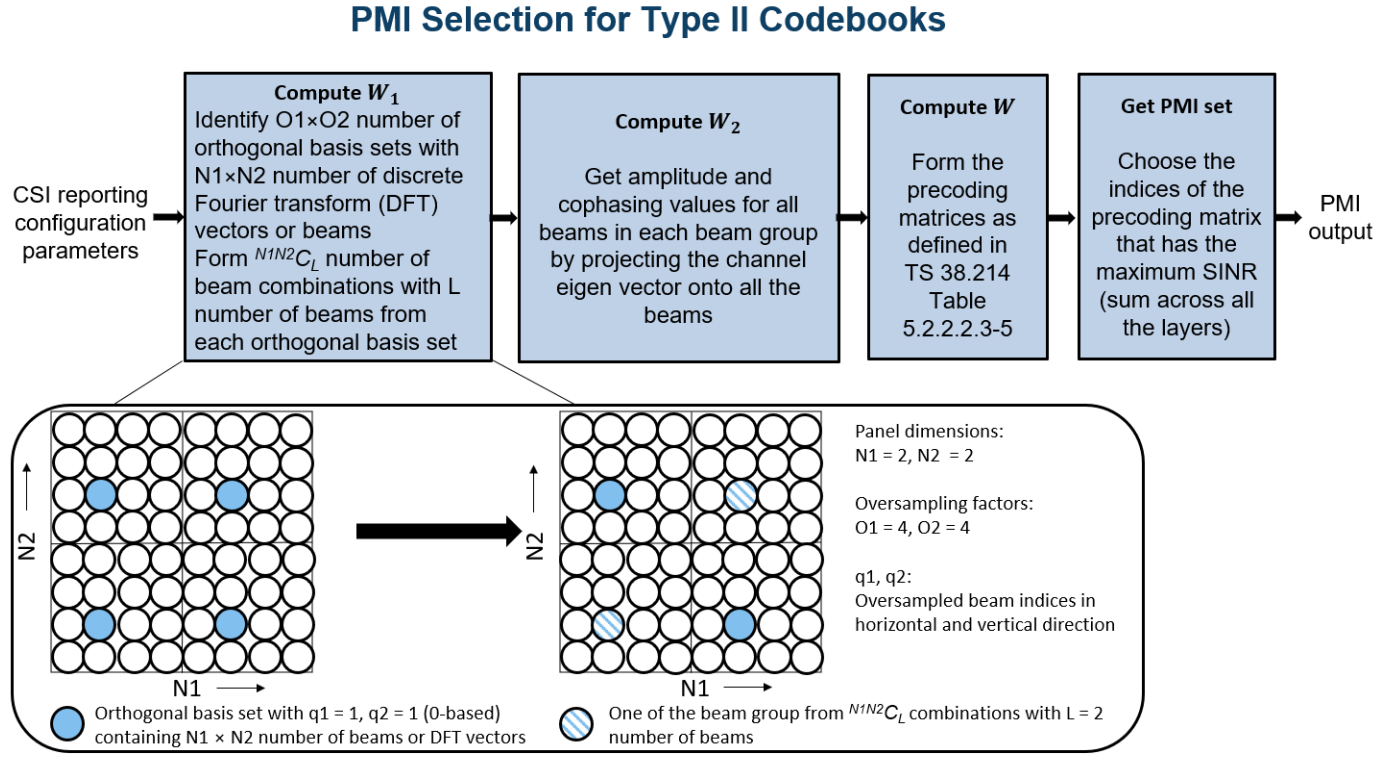
W_2 can be any of the following based on the codebook type:

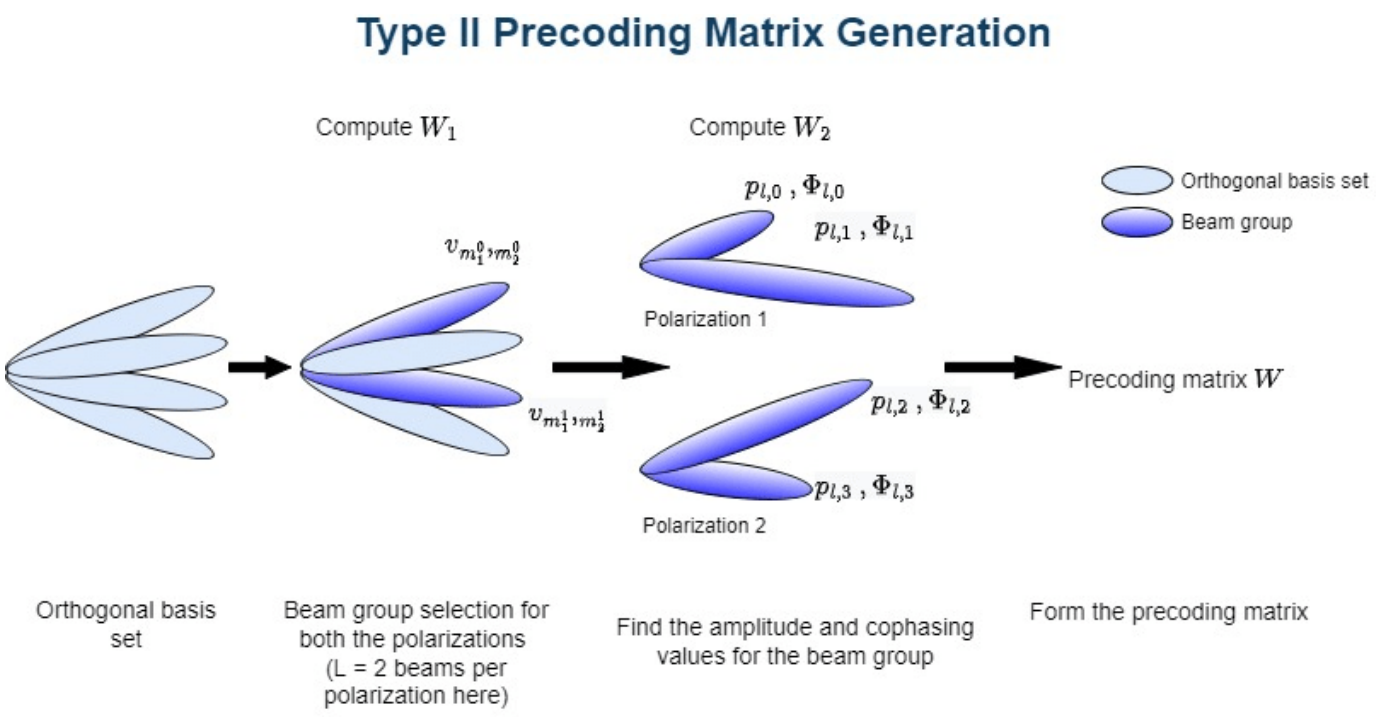
- Beam selection from W_1
- Weighting coefficients for the beams in W_1
- Cophasing values between two polarizations

For type I single-panel and multi-panel codebooks, the function computes signal to interference plus noise ratio (SINR) at the receiver side for all precoding matrices from the selected codebook and for the given channel conditions. The function reports the PMI as the set of indices i_1, i_2, as defined in TS 38.214 Section 5.2.2.2.1 for type I single-panel codebooks and TS 38.214 Section 5.2.2.2.2 for type I multi-panel codebooks. These indices correspond to a precoding matrix W, which gives the maximum SINR. Type I single and multi-panel codebooks are useful for single-user MIMO scenarios. This figure shows the procedure for PMI selection from type I single-panel and multi-panel codebooks.



Type II codebooks consider a set of orthogonal DFT beams to form the precoding matrix. As mentioned above, the W_1 matrix contains the information related to the DFT beams. For the given channel conditions and the number of DFT beams, the function computes the beam amplitude scaling and cophasing values (W_2) for all the DFT beams in each orthogonal beam group so that the linear combination of orthogonal beams approximates the eigenvector of the channel. The function reports the indices $i_1$and i_2, as defined in TS 38.214 Section 5.2.2.2.3. These indices correspond to the precoding matrix W, which gives the maximum SINR. Type II codebook based PMI is more accurate than type I codebook based PMI, as multiple beams are considered in type II codebooks for the approximation of channel eigenvector. Because type II codebooks consider multiple beams, multi-user MIMO scenarios use type II codebooks instead of type I codebooks. These figures show the PMI selection procedure from type II codebooks.

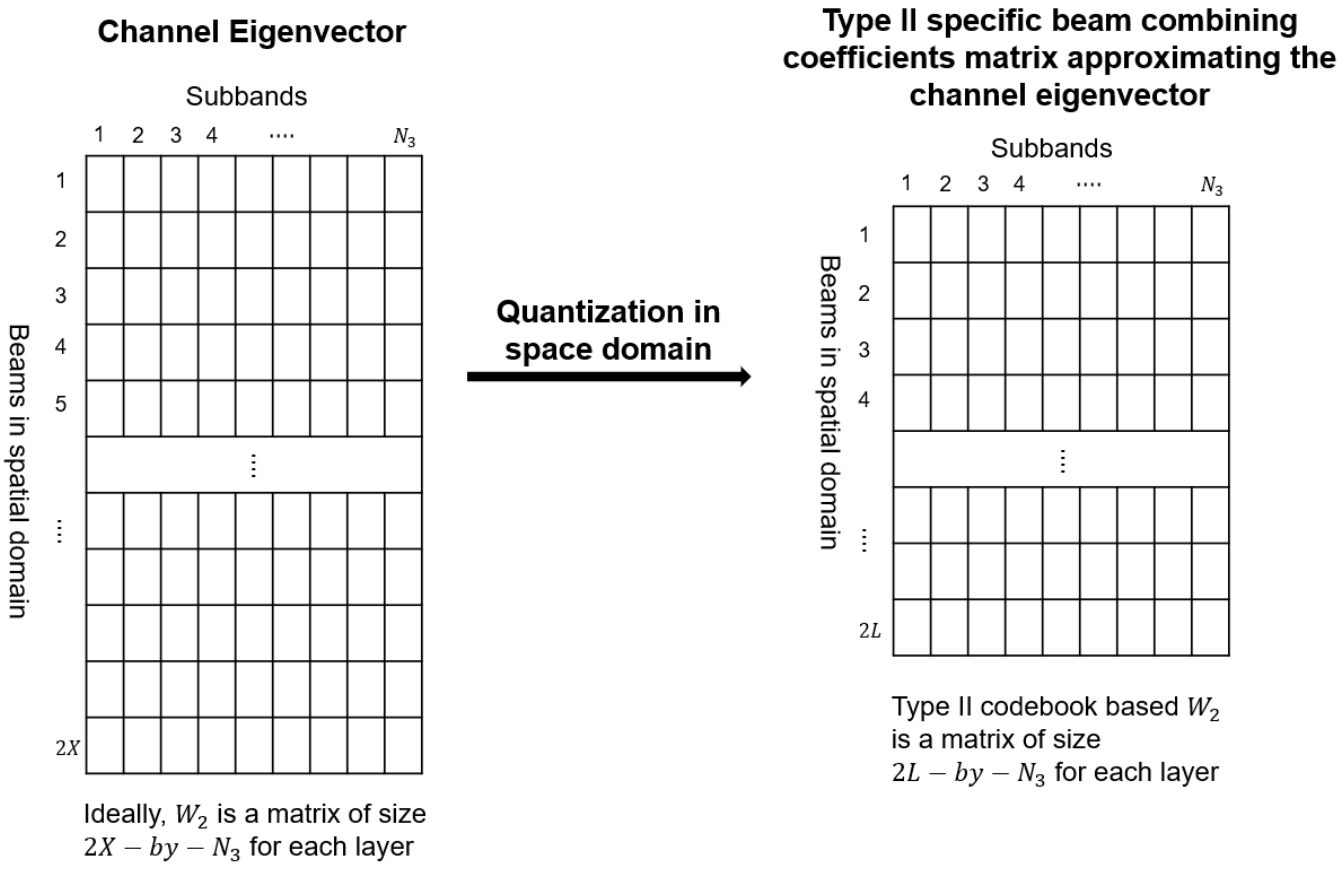




Where,

l - Layer index
i - Beam index
$v_{m_1^i,m_2^i}$ - DFT vector / beam
$p_{l,i}$ - Amplitude scaling for each beam and it is:
$p_{l,i}^{(1)}$ when the subband amplitude is false
$p_{l,i}^{(1)}$ $p_{l,i}^{(2)}$ when the subband amplitude is true
$\Phi_{l,i}$ - Beam cophasing value
W - Precoding matrix, as defined in TS 38.214 Section 5.2.2.2.3-5

This figure shows the quantization process in space domain (i.e., few beams in the approximation of channel eigenvector) in type II codebooks to form W_2 matrix for each layer.



N_3 : Number of frequency-domain units
X : Number of ideal spatial domain beams to represent channel eigen vector per polarization
L : Number of beams used for spatial domain compression per polarization

Because type II codebooks support only up to rank 2, the standard introduces enhanced type II codebooks. The PMI computation that is based on enhanced type II codebooks extends type II codebooks by supporting up to rank 4 with one overhead reduction technique. The beam amplitude scaling and cophasing values (also called beam combining coefficients) for all the beams are computed similar to type II codebooks approach. With the rank extension, the feedback overhead increases linearly and with the number of subbands. For this amount of feedback, the uplink resources may not be sufficient. By taking the advantage of the frequency-domain correlation of beam combining coefficients, a process called DFT or frequency-domain compression is employed for enhanced type II codebooks to reduce the feedback overhead.

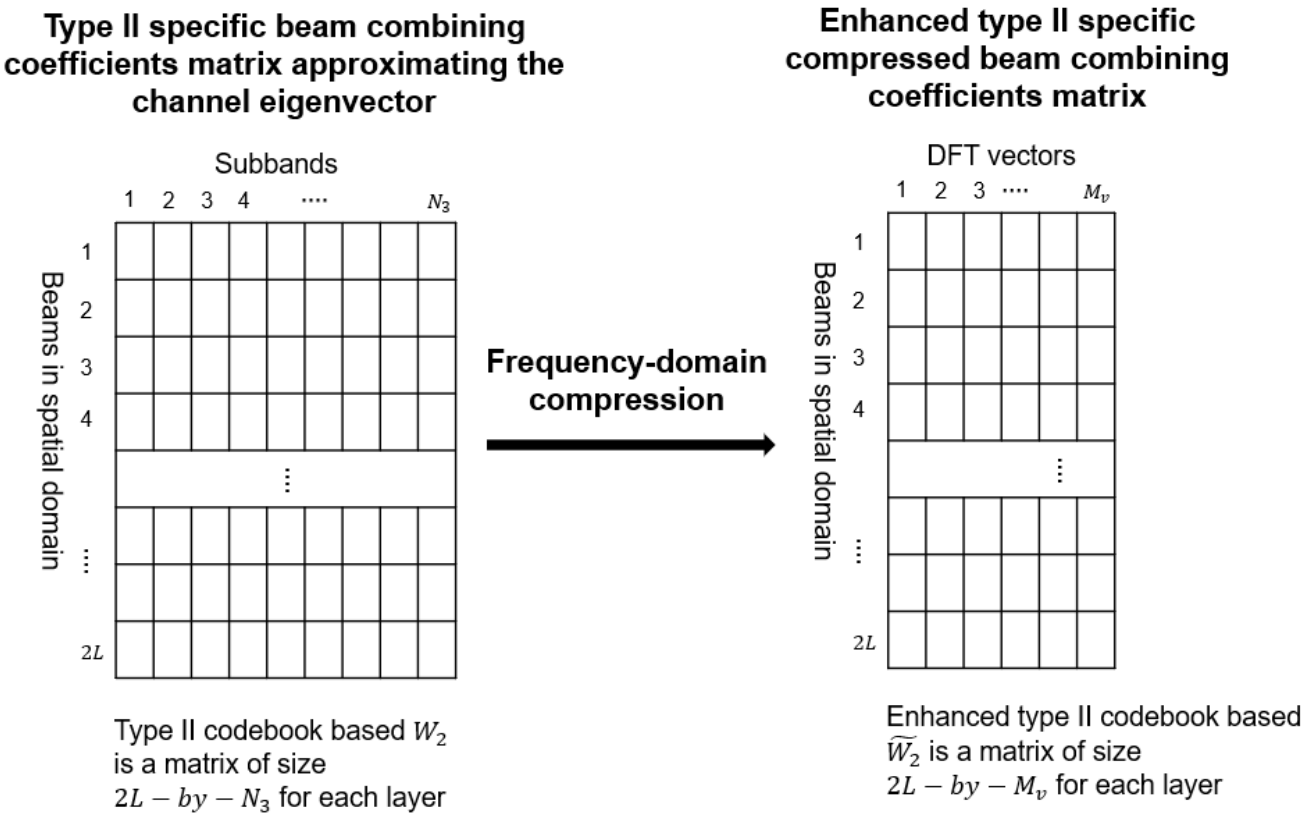
Dual-stage precoding matrix is defined as: $W = W_1 W_2$

Compressed coefficients are computed as: $\widetilde{W_2} = W_2 W_f$, W_f is the DFT matrix used for compression

Precoding matrix is computed as: $W = W_1 \widetilde{W_2} W_f^H$

Note: $\widetilde{W_2}$ related information is reported instead of W_2

The figure shows the frequency-domain compression process.



Along with frequency-domain compression, enhanced type II codebooks limit the number of nonzero coefficients in the feedback by ignoring the weaker beam combining coefficients.

CQI Selection

The CQI is an indicator of channel quality. The CQI value is a scalar in the range [0, 15]. The CQI value provides information about the highest modulation scheme and the code rate (MCS) suitable for the downlink transmission to achieve the required block error rate (BLER) for given channel conditions.

The CSI reference resource is a group of downlink frequency-domain and time-domain resources that are configured as defined in TS 38.214 Section 5.2.2.5. The gNB transmits a single physical downlink shared channel (PDSCH) transport block occupying the resource blocks called the CSI reference resource, with a combination of modulation scheme and target code rate that correspond to each CQI index. The UE selects the highest possible CQI, when the PDSCH transport block can be received with a transport block error probability not exceeding:

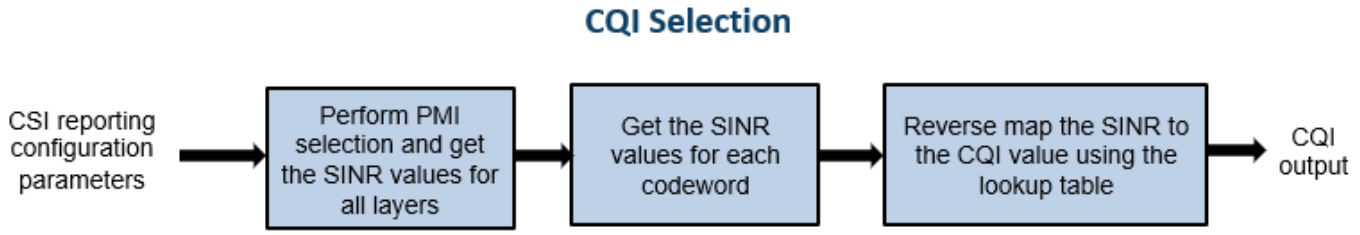
- 0.1 when '*cqi-Table*' is '*table1*' or '*table2*'
- 0.00001 when '*cqi-Table*' is '*table3*'

The '*cqi-Table*' is a higher layer parameter that corresponds to the CQI versus MCS table, and the SINR lookup table is computed for this table. This example uses '*cqi-Table*' as '*table1*' (TS 38.214 Table 5.2.2.1-2). The relationship between the CQI indices, the modulation scheme, and the code rate (from which the transport block size is derived) is described in TS 38.214 Tables 5.2.2.1-2 to 5.2.2.1-4.

The `hCQISelect` function computes the CQI value by considering the SINR values that correspond to the reported PMI. This function uses the precalculated lookup table of the CQI index versus $SINR_i$, as a reference, if you provide it.

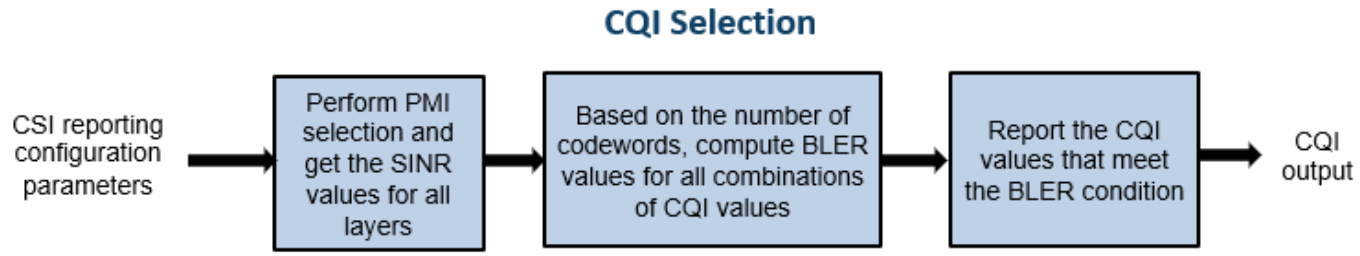
CQI Selection Using SINR Lookup Table

The function maps the SINR values across all layers (corresponding to the reported PMI) to each codeword. For each codeword, the function compares the respective SINR with the $SINR_i$ values from the table and then selects the CQI value against the maximum SINR, which is less than the codeword SINR. The function sets the CQI value so that the BLER is less than or equal to 0.1 when operated at $SINR_i$. If a CQI index of 1 does not satisfy the BLER condition, then the function sets the CQI index to 0. This figure shows the procedure for CQI selection.



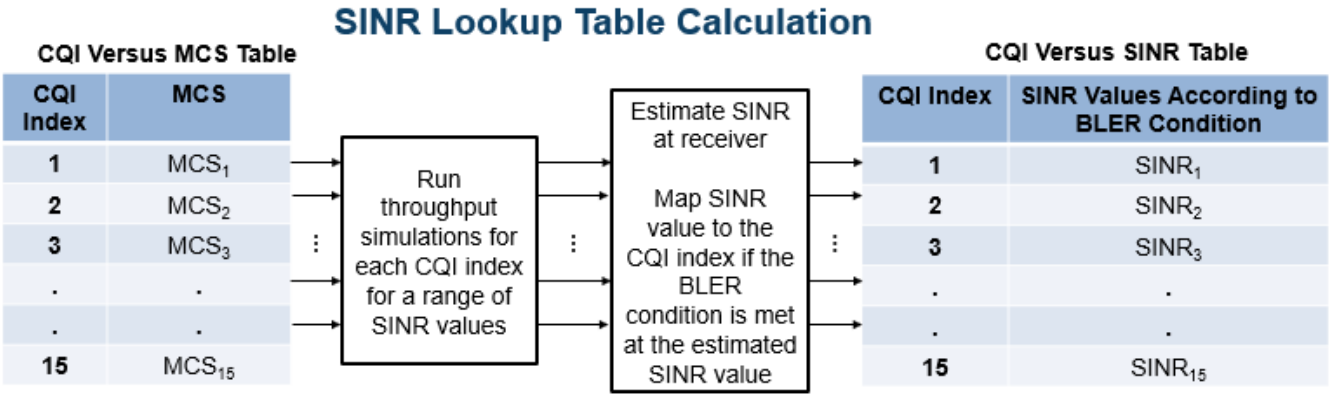
CQI Selection Without SINR Lookup Table

The function computes the CQI values by using the SINR values across all layers corresponding to the reported PMI. Based on the number of codewords, the function forms the combinations of CQI values and reports the one which satisfies the BLER condition.



CQI Versus SINR Table

To create the lookup tables, update the “NR PDSCH Throughput” on page 1-58 example by configuring the CSI reference resource, as defined in TS 38.214 Section 5.2.2.5. Perform simulations for PDSCH throughput by specifying the modulation scheme and code rate corresponding to each CQI, the channel conditions, and a finite range of SINR values with zero interference. Map the SINR value observed at the receiver against each CQI index when the target BLER is achieved. This figure shows the procedure for CQI versus SINR table creation.



Outline of Example

This example shows how to compute the CQI and PMI indices for a 4-by-4 MIMO scenario over a TDL channel with the TDL-C delay profile, a delay spread of 300 nanoseconds, and a maximum Doppler shift of 50 Hz. It compares the time-domain and frequency-domain variations of CQI values against the time-domain and frequency-domain variations of SINR values for practical and perfect channel estimation scenarios. The example also highlights the time-domain and frequency-domain variations of the PMI values corresponding to the reported rank for practical and perfect channel estimation scenarios.

Simulation Length and SNR Point

Set the length of the simulation in terms of the number of 10 ms frames. The SNR (SINR with zero interference) is defined per resource element (RE) and applies to each receive antenna. For an explanation of the SNR definition that this example uses, see “SNR Definition Used in Link Simulations” on page 5-86.

```
nFrames = 5; % Number of 10 ms frames
SNRdB = 10;  % SNR in dB
```

Carrier, Bandwidth Part, and CSI-RS Configuration

Create a carrier configuration object representing a 10 MHz carrier with a subcarrier spacing of 15 kHz.

```
carrier = nrCarrierConfig;
carrier.SubcarrierSpacing = 15;
carrier.NSizeGrid = 52;
```

Configure the size of the bandwidth part (BWP) and the start of the BWP relative to common resource block 0 (CRB 0).

```
NStartBWP = 0;
NSizeBWP = 52;
```

Create a CSI-RS configuration object representing a non-zero-power CSI-RS (NZP-CSI-RS) resource set with a set of NZP-CSI-RS resources. Make sure that the NZP-CSI-RS resources have the same code division multiplexing (CDM) types and the same number of CSI-RS ports, as defined in TS 38.214 Section 5.2.2.3.1.

```
csirs = nrCSIRSConfig;
csirs.CSIRSType = {'nzp','nzp','nzp'};
csirs.RowNumber = [4 4 4];
csirs.NumRB = 52;
csirs.RBOffset = 0;
csirs.CSIRSPeriod = [4 0];
csirs.SymbolLocations = {0, 0, 0};
csirs.SubcarrierLocations = {0, 4, 8};
csirs.Density = {'one','one','one'};
```

Configure the number of transmit and receive antennas. The number of transmit antennas must be equal to the number of CSI-RS ports because this example does not consider the mapping of antenna ports to physical antennas.

```
nTxAnts = csirs.NumCSIRSPorts(1);
nRxAnts = 4;
```

Validate the CSI-RS configuration object.

```
validateCSIRSConfig(carrier,csirs,nTxAnts);
```

CSI Reporting Configuration

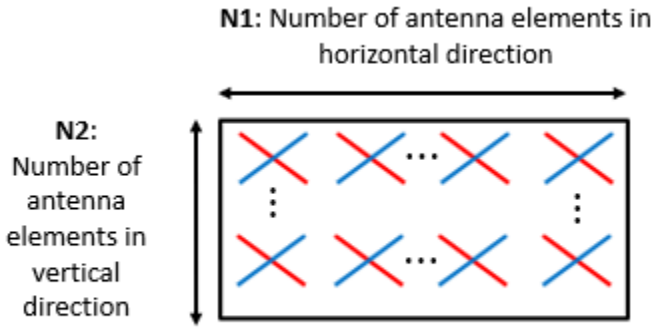
Specify the parameters for the CQI, PMI, and RI computation as a structure with these fields:

- `NSizeBWP` — Size of the BWP in terms of the number of physical resource blocks (PRBs)
- `NStartBWP` — Starting PRB index of the BWP relative to CRB 0
- `CQITable` — Channel quality indicator table (`'table1'`, `'table2'`, `'table3'`), as defined in TS 38.214 Tables 5.2.2.1-2 to 5.2.2.1-4
- `CodebookType` — Codebook type for CSI parameter computation (`'Type1SinglePanel'`, `'Type1MultiPanel'`, `'Type2'`, `'eType2'`)
- `PanelDimensions` — Antenna panel dimensions corresponding to the `CodebookType` field

If `CodebookType` is set to any one of (`'Type1SinglePanel'`, `'Type2'`, `'eType2'`), the panel dimensions are in the form of [N1 N2] according to TS 38.214 Table 5.2.2.2.1-2. This figure shows the supported panel dimensions.

Supported Configurations of (N1, N2)

Number of CSI-RS Ports	(N1,N2)	(O1,O2) DFT Oversampling Factors
4	(2,1)	(4,1)
8	(2,2)	(4,4)
	(4,1)	(4,1)
12	(3,2)	(4,4)
	(6,1)	(4,1)
16	(4,2)	(4,4)
	(8,1)	(4,1)
24	(4,3)	(4,4)
	(6,2)	(4,4)
	(12,1)	(4,1)
32	(4,4)	(4,4)
	(8,2)	(4,4)
	(16,1)	(4,1)

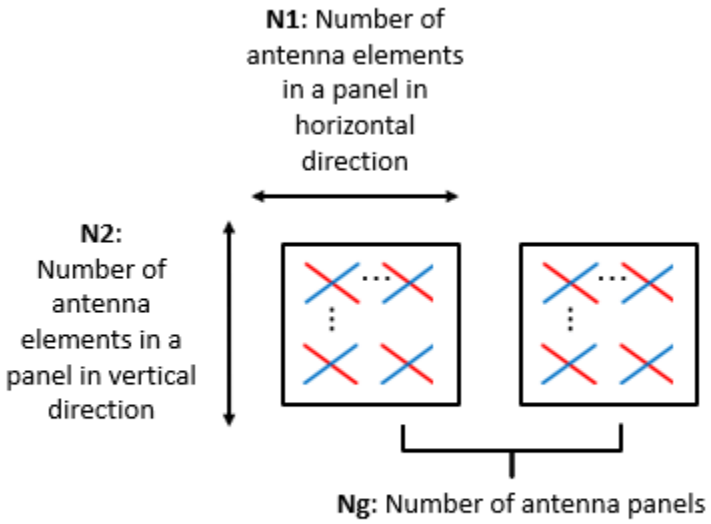


Note: Number of CSI-RS ports = 2×N1×N2

If `CodebookType` is set to '`Type1MultiPanel`', the panel dimensions are in the form of [Ng N1 N2] according to TS 38.214 Table 5.2.2.2.2-1. This figure shows the supported type I multi-panel dimensions.

Supported Configurations of (Ng, N1, N2)

Number of CSI-RS Ports	(Ng,N1,N2)	(O1,O2) DFT Oversampling Factors
8	(2,2,1)	(4,1)
16	(2,4,1)	(4,1)
	(4,2,1)	(4,1)
	(2,2,2)	(4,4)
32	(2,8,1)	(4,1)
	(4,4,1)	(4,1)
	(2,4,2)	(4,4)
	(4,2,2)	(4,4)



Note: Number of CSI-RS ports = 2×Ng×N1×N2

- `CQIMode` — Mode of CQI reporting (`'Subband'`, `'Wideband'`)
- `PMIMode` — Mode of PMI reporting (`'Subband'`, `'Wideband'`)
- `SubbandSize` — Size of the subband, as defined in TS 38.214 Table 5.2.1.4-2 (required only when `CQIMode` or `PMIMode` is `'Subband'`)
- `PRGSize` — Precoding resource block group (PRG) size for the CQI calculation (required only when the reporting must be done for the report quantity '*cri-RI-i1-CQI*' and `CodebookType` is configured as `'Type1SinglePanel'`, as defined in TS 38.214 Section 5.2.1.4.2)
- `CodebookMode` — Codebook mode according to which the codebooks are derived. The value must be 1 or 2. This field is applicable only when the `CodebookType` is specified as

`'Type1SinglePanel'` or `'Type1MultiplePanel'`. When the `CodebookType` is specified as `'Type1SinglePanel'`, this field is required only when the number of transmission layers is 1 or 2 and the number of CSI-RS ports is greater than 2. When the `CodebookType` is specified as `'Type1MultiplePanel'`, this field is required for all the number of transmission layers, but the `CodebookMode` value 2 is applicable only for the panel configurations with Ng value 2.

- `CodebookSubsetRestriction` — For type I single-panel and multi-panel codebooks, it is a restriction parameter pertaining to the codebook index set i_1, that is, v_{lm} or $\widetilde{v}_{lm}$ restriction parameter, as defined in TS 38.214 Section 5.2.2.2.1. This field denotes the set of i_1 indices ($[i_{11}, i_{12}]$) that are restricted from the consideration for the PMI computation. For type II codebooks, this field restricts the maximum amplitudes that are allowed for four vector groups indicated by $r_1^{(k)}$, $r_2^{(k)}$ for $k = 0, 1, 2, 3$, as defined in TS 38.214 Section 5.2.2.2.3. For enhanced type II codebooks, this field restricts the maximum average amplitude coefficients that are allowed for four vector groups indicated by $r_1^{(k)}$, $r_2^{(k)}$ for $k = 0, 1, 2, 3$, as defined in TS 38.214 Section 5.2.2.2.5. In any of the codebook types, this field is specified as a bitmap.
- `i2Restriction` — Restriction parameter pertaining to the codebook index i_2. This field denotes the set of i_2 indices that are restricted from the consideration for the PMI computation. This field is applicable only when `CodebookType` is `'Type1SinglePanel'`.
- `RIRestriction` — Restriction parameter pertaining to the rank indicator, which implies the set of ranks that are restricted from usage for the RI computation.
- `NumberOfBeams` — Number of beams in the beam group per polarization. This field is applicable only when the `CodebookType` is specified as `'Type2'`. The value must be one of {2, 3, 4}.
- `PhaseAlphabetSize` — Represents the range of the phases that are considered for the computation of i_2 indices. This field is applicable only when the `CodebookType` is specified as `'Type2'`. The value must be one of {4, 8}. The value 4 represents the phases corresponding to QPSK and the value 8 represents the phases corresponding to 8-PSK.
- `SubbandAmplitude` — Logical scalar that enables the reporting of differential amplitudes per subband when set to `true` and disables differential subband amplitude in PMI reporting when set to `false`. The value must be one of {`true`, `false`}. This field is applicable when `CodebookType` is specified as `'Type2'` and `PMIMode` is `'Subband'`.
- `ParameterCombination` — It is a positive scalar integer in the range 1...8. This is applicable when `CodebookType` is specified as `'eType2'`. This parameter defines the number of beams and two other parameters as defined in TS 38.214 Table 5.2.2.2.5-1.
- `NumberOfPMISubbandsPerCQISubband` — Represents the number of PMI subbands within one CQI subband, as defined in TS 38.214 Section 5.2.2.2.5. It is a positive scalar integer, and it must be either 1 or 2.

Configure CSI reporting configuration parameters.

```
reportConfig.NStartBWP = NStartBWP;
reportConfig.NSizeBWP = NSizeBWP;
reportConfig.CQITable = 'table1';
reportConfig.CodebookType = 'Type1SinglePanel';
reportConfig.PanelDimensions = [2 1];
reportConfig.CQIMode = 'Subband';
reportConfig.PMIMode = 'Subband';
reportConfig.SubbandSize = 4;
reportConfig.PRGSize = [];
reportConfig.CodebookMode = 1;
reportConfig.CodebookSubsetRestriction = [];
```

```
reportConfig.i2Restriction = [];
reportConfig.RIRestriction = [];
reportConfig.NumberOfBeams = 2;                       % Applicable only when CodebookType is 'Type2
reportConfig.SubbandAmplitude = false;                % Applicable only when CodebookType is 'Type2
reportConfig.PhaseAlphabetSize = 4;                   % Applicable only when CodebookType is 'Type2
reportConfig.ParameterCombination = 2;                % Applicable only when CodebookType is 'eType2
reportConfig.NumberOfPMISubbandsPerCQISubband = 2; % Applicable only when CodebookType is 'eType2
```

Propagation Channel Model Configuration

Create a TDL channel with the TDL-C delay profile, a delay spread of 300 nanoseconds, and a maximum Doppler shift of 50 Hz.

```
channel = nrTDLChannel;
channel.NumTransmitAntennas = nTxAnts;
channel.NumReceiveAntennas = nRxAnts;
channel.DelayProfile = 'TDL-C';
channel.MaximumDopplerShift = 50;
channel.DelaySpread = 300e-9;
```

Set the sampling rate for the channel model by using the value that the `nrOFDMInfo` function returns.

```
waveformInfo = nrOFDMInfo(carrier);
channel.SampleRate = waveformInfo.SampleRate;
```

Get the maximum number of delayed samples by a channel multipath component. The maximum number of delayed samples is calculated from the channel path with the largest delay and the implementation delay of the channel filter. The number of samples that correspond to the maximum channel delay is required later to flush the channel filter to obtain the received signal.

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

Processing Loop

For each slot, generate the CSI-RS, transmit it through the channel, and then process at the receiver side to compute the CQI, PMI, and RI values. Follow the steps for the transmit-to-receive processing for each slot.

1. **Generate the resource grid** — Generate the slot grid that contains the CSI-RS and map the slot grid from CSI-RS ports to transmit antennas.
2. **Generate the waveform** — Perform orthogonal frequency division multiplexing (OFDM) modulation on the generated grid by using the `nrOFDMModulate` function.
3. **Model and apply a noisy channel** — Pass the waveform through a TDL-C fading channel, and then add the additive white Gaussian noise (AWGN). The SNR is defined per RE and applies to each receive antenna.
4. **Perform timing synchronization and OFDM demodulation** — For practical synchronization, correlate the received waveform with the CSI-RS. For perfect synchronization, use the path gains and path filters of the channel. Perform OFDM demodulation on the synchronized signal using the `nrOFDMDemodulate` function.
5. **Perform channel estimation** — For practical channel estimation, use the CSI-RS. For perfect channel estimation, use the path gains, the path filters, and the sample times of channel snapshots.

6 **Compute CQI, PMI, and RI values** — Compute CQI, PMI, and RI values using practical and perfect channel estimations.

Calculate total number of slots.

```
totSlots = nFrames*carrier.SlotsPerFrame;
```

Initialize variables to store CQI, PMI, RI, and subband SINR values for practical and perfect channel estimation scenarios.

After the completion of processing loop:

- The variables `cqiPracticalPerSlot`, `cqiPerfectPerSlot`, `SINRPerSubbandPerCWPractical`, and `SINRPerSubbandPerCWPrerfect` are multidimensional arrays of size `numSBs`-by-2-by-`totSlots`. The `numSBs` value is equal to the number of subbands plus 1, that is, the corresponding wideband value followed by subband values. The second dimension value denotes the maximum possible number of codewords.
- The variables `pmiPracticalPerSlot` and `pmiPerfectPerSlot` are arrays of structures of size `totSlots`. The structure represents the PMI values with fields `i1` and `i2`.
- The variables `riPracticalPerSlot` and `riPerfectPerSlot` are arrays of size 1-by-`totSlots`.

```
cqiPracticalPerSlot = [];
subbandCQIPractical = [];
pmiPracticalPerSlot = struct('i1',[],'i2',[]);
SINRPerSubbandPerCWPractical = [];
cqiPerfectPerSlot = [];
subbandCQIPerfect = [];
pmiPerfectPerSlot = struct('i1',[],'i2',[]);
SINRPerSubbandPerCWPerfect = [];
riPracticalPerSlot = [];
riPerfectPerSlot = [];

% Get number of CSI-RS ports
csirsPorts = csirs.NumCSIRSPorts(1);

% Get CDM lengths corresponding to configured CSI-RS resources
cdmLengths = getCDMLengths(csirs);

% Initialize the practical timing offset as zero. It is updated in the
% slots when the correlation is strong for practical synchronization.
offsetPractical = 0;

% Initialize a variable to store the information of the slots
% where NZP-CSI-RS resources are present. It is of length totSlots.
totSlotsBinaryVec = zeros(1,totSlots);

% Set RNG state for repeatability
rng('default');

% Loop over all slots
for nslot = 0:totSlots - 1
    % Create carrier resource grid for one slot
    csirsSlotGrid = nrResourceGrid(carrier,csirsPorts);

    % Update slot number in carrier configuration object
    carrier.NSlot = nslot;
```

```
% Generate CSI-RS indices and symbols
csirsInd = nrCSIRSIndices(carrier,csirs);
csirsSym = nrCSIRS(carrier,csirs);

% Map CSI-RS to slot grid
csirsSlotGrid(csirsInd) = csirsSym;

% Map CSI-RS ports to transmit antennas
wtx = eye(csirsPorts,nTxAnts);
txGrid = reshape(reshape(csirsSlotGrid,[],csirsPorts)*wtx,size(csirsSlotGrid,1),size(csirsSlo

% Perform OFDM modulation to generate time-domain waveform
txWaveform = nrOFDMModulate(carrier,txGrid);

% Append zeros at the end of the transmitted waveform to flush channel
% content. These zeros take into account any delay introduced in the
% channel. The channel delay is a mix of multipath delay and
% implementation delay. This value may change depending on the sampling
% rate, delay profile, and delay spread.
txWaveform = [txWaveform; zeros(maxChDelay,size(txWaveform,2))]; %#ok<AGROW>

% Transmit waveform through channel
[rxWaveform,pathGains,sampleTimes] = channel(txWaveform);

% Generate and add AWGN to received waveform
SNR = 10^(SNRdB/10);                                                % Linear SNR value
sigma = 1/(sqrt(2.0*nRxAnts*double(waveformInfo.Nfft)*SNR));        % Noise standard deviatio
noise = sigma*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
rxWaveform = rxWaveform + noise;

% Perform practical timing estimation. Correlate the received waveform
% with the CSI-RS to obtain the timing offset estimate and the
% correlation magnitude. Use the hSkipWeakTimingOffset function to
% update the receiver timing offset. If the correlation peak is weak,
% the current timing estimate is ignored and the previous offset is
% used.
[t,mag] = nrTimingEstimate(carrier,rxWaveform,csirsInd,csirsSym);
offsetPractical = hSkipWeakTimingOffset(offsetPractical,t,mag);

% Get path filters
pathFilters = getPathFilters(channel);
% Perform perfect timing estimation
offsetPerfect = nrPerfectTimingEstimate(pathGains,pathFilters);

% Perform time-domain offset correction for practical and
% perfect timing estimation scenarios
rxWaveformPractical = rxWaveform(1+offsetPractical:end,:);
rxWaveformPerfect = rxWaveform(1+offsetPerfect:end,:);

% Perform OFDM demodulation on previously synchronized waveforms
rxGridPractical = nrOFDMDemodulate(carrier,rxWaveformPractical);
rxGridPerfect = nrOFDMDemodulate(carrier,rxWaveformPerfect);

% Append zeros when the timing synchronization results in an incomplete
% slot
symbPerSlot = carrier.SymbolsPerSlot;
K = size(rxGridPractical,1);
```

```
LPractical = size(rxGridPractical,2);
LPerfect = size(rxGridPerfect,2);
if LPractical < symbPerSlot
    rxGridPractical = cat(2,rxGridPractical,zeros(K,symbPerSlot-LPractical,nRxAnts));
end
if LPerfect < symbPerSlot
    rxGridPerfect = cat(2,rxGridPerfect,zeros(K,symbPerSlot-LPerfect,nRxAnts));
end
rxGridPractical = rxGridPractical(:,1:symbPerSlot,:);
rxGridPerfect = rxGridPerfect(:,1:symbPerSlot,:);

% Consider only the NZP-CSI-RS symbols and indices for channel estimation
nzpCSIRSSym = csirsSym(csirsSym ~= 0);
nzpCSIRSInd = csirsInd(csirsSym ~= 0);

% Calculate practical channel estimate. Use a time-averaging window
% that covers all the transmitted CSI-RS symbols.
[PracticalHest,nVarPractical] = nrChannelEstimate(carrier,rxGridPractical, ...
    nzpCSIRSInd,nzpCSIRSSym,'CDMLengths',cdmLengths,'AveragingWindow',[0 5]);

% Perform perfect channel estimation
PerfectHest = nrPerfectChannelEstimate(carrier,pathGains,pathFilters,offsetPerfect,sampleTime

% Get perfect noise estimate value from noise realization
noiseGrid = nrOFDMDemodulate(carrier,noise(1+offsetPerfect:end,:));
nVarPerfect = var(noiseGrid(:));
if ~isempty(nzpCSIRSInd)
    % Set the totSlotsBinaryVec value corresponding to the slot
    % index where NZP-CSI-RS is present to 1
    totSlotsBinaryVec(nslot+1) = 1;

    % Calculate the RI value using practical channel estimate
    numLayersPractical = hRISelect(carrier,csirs,reportConfig,PracticalHest,nVarPractical,'Ma

    % Calculate CQI and PMI values using practical channel estimate
    [cqiPractical,pmiPractical,cqiInfoPractical,pmiInfoPractical] = hCQISelect(carrier,csirs
        reportConfig,numLayersPractical,PracticalHest,nVarPractical);
    numCodeWordsPr = size(cqiPractical,2);
    numSBs = size(cqiPractical,1);

    % Store CQI, PMI, RI, and subband SINR values of each slot for the
    % practical channel estimation scenario. Because the number of
    % codewords can vary based on the rank, append NaNs to the
    % CQI-related variables to account for the second codeword
    % information in the slots where only one codeword is present.
    riPracticalPerSlot(1,nslot+1) = numLayersPractical; %#ok<SAGROW>
    cqiPracticalPerSlot(:,:,nslot+1) = [cqiPractical NaN(numSBs,2-numCodeWordsPr)]; %#ok<SAGR
    pmiPracticalPerSlot(nslot+1) = pmiPractical;
    subbandCQIPractical(:,:,nslot+1) = [cqiInfoPractical.SubbandCQI NaN(numSBs,2-numCodeWords
    SINRPerSubbandPerCWPractical(:,:,nslot+1) = [cqiInfoPractical.SINRPerSubbandPerCW NaN(num

    % Calculate the RI value using perfect channel estimate
    numLayersPerfect = hRISelect(carrier,csirs,reportConfig,PerfectHest,nVarPerfect,'MaxSE')

    % Calculate CQI and PMI values using perfect channel estimate
    [cqiPerfect,pmiPerfect,cqiInfoPerfect,pmiInfoPerfect] = hCQISelect(carrier,csirs, ...
        reportConfig,numLayersPerfect,PerfectHest,nVarPerfect);
    numCodeWordsPe = size(cqiPerfect,2);
```

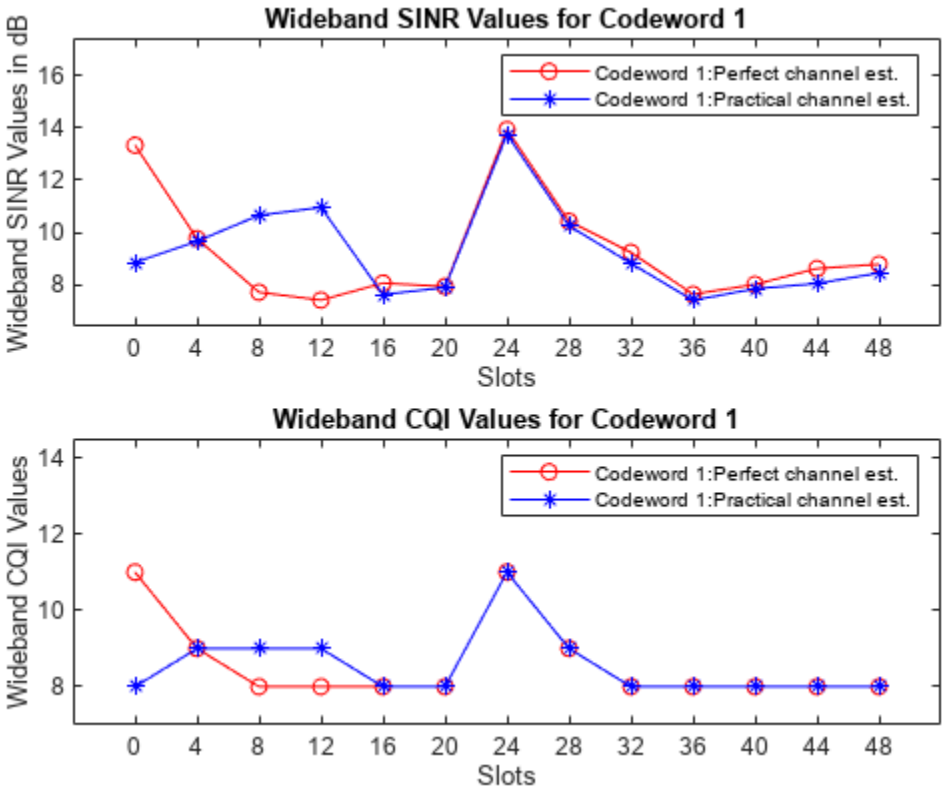
```
        % Store CQI, PMI, RI, and subband SINR values of each slot for the
        % perfect channel estimation scenario. Because the number of
        % codewords can vary based on the rank, append NaNs to the
        % CQI-related variables to account for the second codeword
        % information in the slots where only one codeword is present.
        riPerfectPerSlot(1,nslot+1) = numLayersPerfect; %#ok<SAGROW>
        cqiPerfectPerSlot(:,:,nslot+1) = [cqiPerfect NaN(numSBs,2-numCodeWordsPe)]; %#ok<SAGROW>
        subbandCQIPerfect(:,:,nslot+1) = [cqiInfoPerfect.SubbandCQI NaN(numSBs,2-numCodeWordsPe)]
        pmiPerfectPerSlot(nslot+1) = pmiPerfect;
        SINRPerSubbandPerCWPerfect(:,:,nslot+1) = [cqiInfoPerfect.SINRPerSubbandPerCW NaN(numSBs
    end
end

% Get the active slot numbers (1-based) in which NZP-CSI-RS is present
activeSlotNum = find(totSlotsBinaryVec);

% Fill the CQI, PMI and RI variables with NaNs in the slots where NZP-CSI-RS is
% absent according to codebook type
[cqiPracticalPerSlot,subbandCQIPractical,pmiPracticalPerSlot,SINRPerSubbandPerCWPractical, ...
    cqiPerfectPerSlot,subbandCQIPerfect,pmiPerfectPerSlot,SINRPerSubbandPerCWPerfect,riPracticalP
    riPerfectPerSlot] = fillInactiveSlots(cqiPracticalPerSlot,subbandCQIPractical, ...
    pmiPracticalPerSlot,SINRPerSubbandPerCWPractical,cqiPerfectPerSlot,subbandCQIPerfect,pmiPerfe
    SINRPerSubbandPerCWPerfect,riPracticalPerSlot,riPerfectPerSlot,reportConfig,totSlotsBinaryVec
```

Compare CQI, PMI, and RI Values Reported in Practical and Perfect Channel Estimation Scenarios

Plot CQI Indices

Plot the wideband SINR values and wideband CQI values for each codeword for practical and perfect channel estimation scenarios. The plot shows only the slots in which the CSI-RS is transmitted or in which the CQI is calculated. The figure shows how the SINR and the corresponding reported CQI vary across the slots due to channel fading.

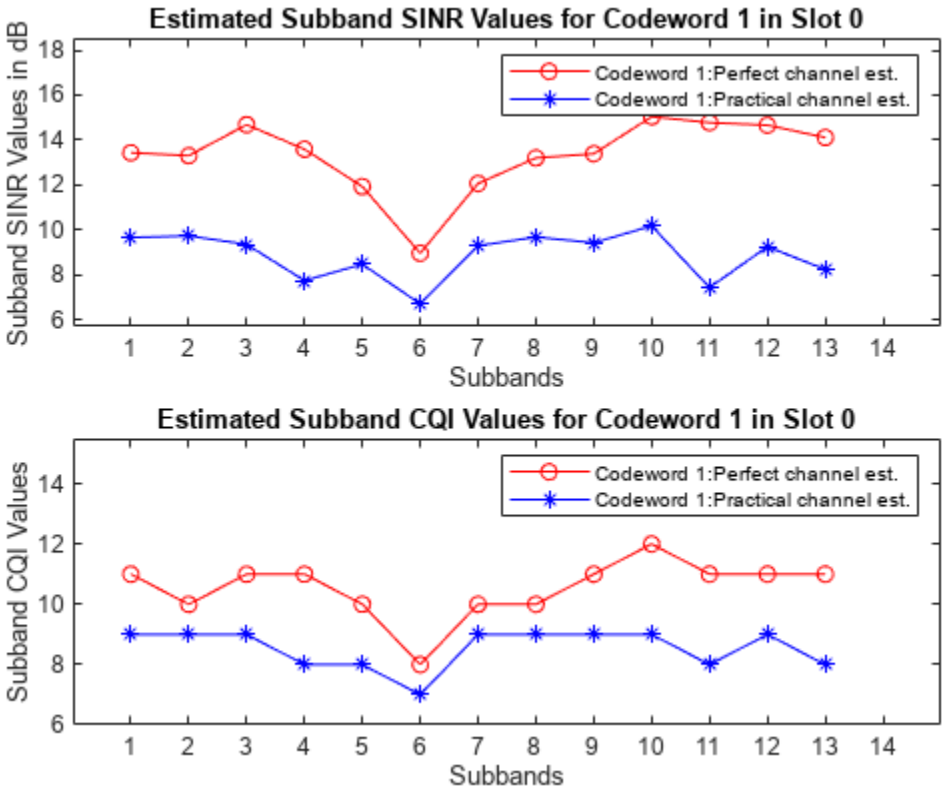
```
plotWidebandCQIAndSINR(cqiPracticalPerSlot,cqiPerfectPerSlot, ...
    SINRPerSubbandPerCWPractical,SINRPerSubbandPerCWPerfect,activeSlotNum);
```



Plot the subband SINR values and subband CQI values for each codeword for practical and perfect channel estimation scenarios. The plot shows the variations of SINR and CQI values across the subbands for a specified slot number (0-based) if the CQI is reported in that slot. Otherwise, plot is not generated because the CQI is not reported.

The `plotSubbandCQIAndSINR` function plots the CQI values across all subbands only when `CQIMode` is configured as `'Subband'`.

```
% Provide slot number for which subband CQI values and PMI i2 indices must be plotted
slotNumForSBType1 = 0; % Consider slot number as 0 (0-based) here, because CSI is
                       % reported in that slot for the configured CSI-RS resource(s)
plotSubbandCQIAndSINR(subbandCQIPractical,subbandCQIPerfect, ...
    SINRPerSubbandPerCWPractical,SINRPerSubbandPerCWPerfect,activeSlotNum,slotNumForSBType1);
```



Plot PMI and RI Indices

First plot in this section shows the variations of RI indices due to channel fading conditions, for practical and perfect channel estimation scenarios. The plot shows only the slots in which the CSI-RS is transmitted or in which RI is calculated.

Codebook Type `'Type1SinglePanel'`

The below code shows the variations of PMI due to channel fading conditions, for practical and perfect channel estimation scenarios.

The PMI indices are $i_1 : [i_{1,1}, i_{1,2}, i_{1,3}]$ and i_2, as defined in TS 38.214 Section 5.2.2.2.1.

The first plot shows the rank and the corresponding PMI i_1 indices variation across multiple slots.

The second plot shows the variation in i_2 indices across the:

- Slots when the `PMIMode` is specified as `'Wideband'`
- Subbands in the specified slot when the `PMIMode` is specified as `'Subband'`

Codebook Type `'Type1MultiPanel'`

The below code shows the variations of PMI due to channel fading conditions, for practical and perfect channel estimation scenarios.

The PMI indices are i_1:$[i_{1,1}, i_{1,2}, i_{1,3}, i_{1,4,1}, i_{1,4,2}, i_{1,4,3}]$ and i_2:$[i_{2,0}, i_{2,1}, i_{2,2}]$, as defined in TS 38.214 Section 5.2.2.2.2.

In this case, the first two plots show the rank and the corresponding PMI i_1 indices ($i_{1,1}$ to $i_{1,4,3}$) variation across the slots.

The third plot shows the variation in each of the i_2 indices ($i_{2,0}$ to $i_{2,2}$) across the:

- Slots when the `PMIMode` is specified as `'Wideband'`
- Subbands in the specified slot when the `PMIMode` is specified as `'Subband'`

Codebook Type `'Type2'` or `'eType2'`

The below code shows the grid of beams by highlighting the DFT vectors or beams that are used to construct the precoding matrix, for practical and perfect channel estimation scenarios. The plot shows the type II or enhanced type II PMI related information for a specified slot number (0-based), if PMI is reported in that slot. Otherwise, plot is not generated because the PMI is not reported.

For type II codebooks, the PMI indices are defined in TS 38.214 Section 5.2.2.2.3. These indices are:

- i_1:$[i_{1,1}, i_{1,2}, i_{1,3,1}, i_{1,4,1}]$ and i_2:$[i_{2,1,1}]$ for single layer and `SubbandAmplitude` as `'false'`
- i_1:$[i_{1,1}, i_{1,2}, i_{1,3,1}, i_{1,4,1}]$ and i_2:$[i_{2,1,1}, i_{2,2,1}]$ for single layer and `SubbandAmplitude` as `'true'`
- i_1:$[i_{1,1}, i_{1,2}, i_{1,3,1}, i_{1,4,1}, i_{1,3,2}, i_{1,4,2}]$ and i_2:$[i_{2,1,1}, i_{2,1,2}]$ for two layers and `SubbandAmplitude` as `'false'`
- i_1:$[i_{1,1}, i_{1,2}, i_{1,3,1}, i_{1,4,1}, i_{1,3,2}, i_{1,4,2}]$ and i_2:$[i_{2,1,1}, i_{2,2,1}, i_{2,1,2}, i_{2,2,2}]$ for two layers and `SubbandAmplitude` as `'true'`

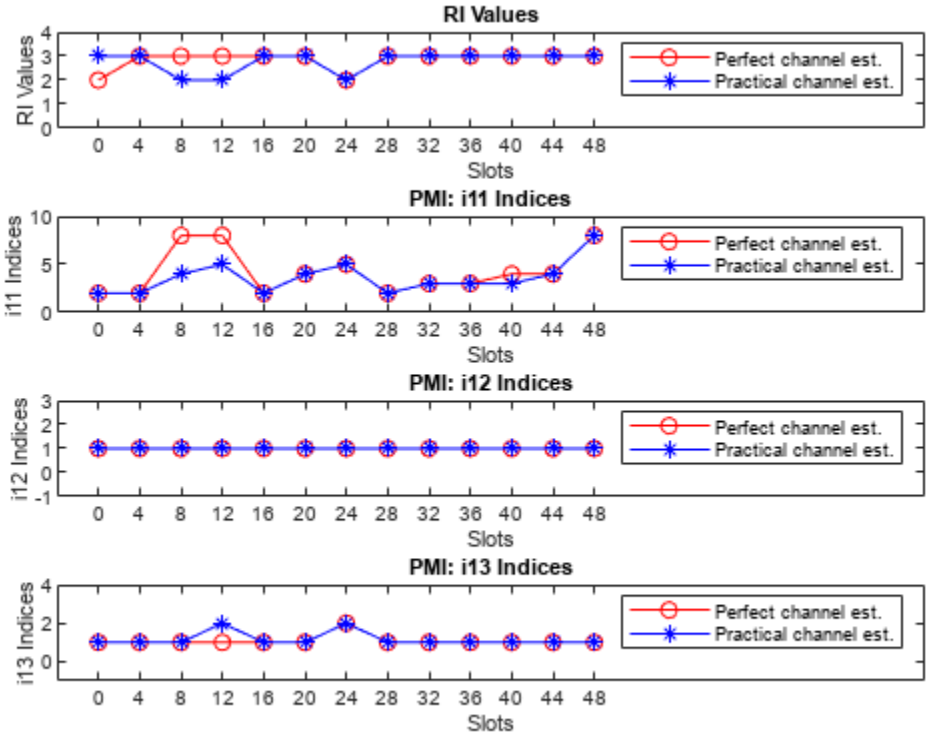
For enhanced type II codebooks, the PMI indices are defined in TS 38.214 Section 5.2.2.2.5. These indices are:

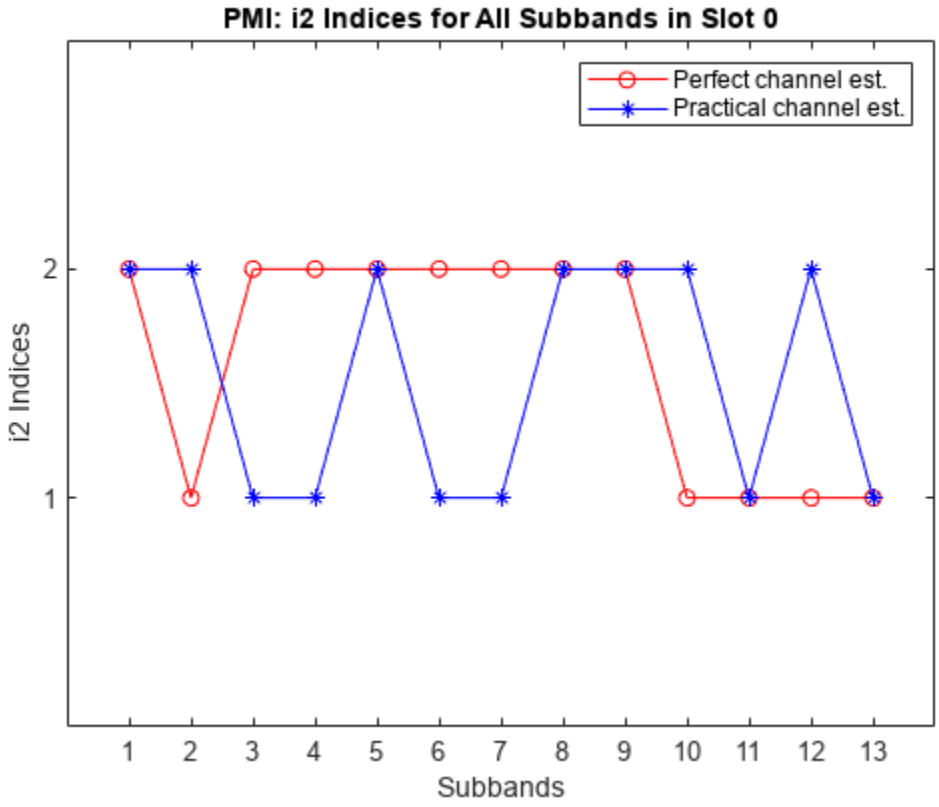
- i_1:$[i_{1,1}, i_{1,2}, i_{1,5}, i_{1,6,1}, i_{1,7,1}, i_{1,8,1}]$ and i_2:$[i_{2,3,1}, i_{2,4,1}, i_{2,5,1}]$ for single layer
- i_1:$[i_{1,1}, i_{1,2}, i_{1,5}, i_{1,6,1}, i_{1,7,1}, i_{1,8,1}, i_{1,6,2}, i_{1,7,2}, i_{1,8,2}]$ and i_2:$[i_{2,3,1}, i_{2,4,1}, i_{2,5,1}, i_{2,3,2}, i_{2,4,2}, i_{2,5,2}]$ for two layers
- i_1:$[i_{1,1}, i_{1,2}, i_{1,5}, i_{1,6,1}, i_{1,7,1}, i_{1,8,1}, i_{1,6,2}, i_{1,7,2}, i_{1,8,2}, i_{1,6,3}, i_{1,7,3}, i_{1,8,3}]$ and i_2:$[i_{2,3,1}, i_{2,4,1}, i_{2,5,1}, i_{2,3,2}, i_{2,4,2}, i_{2,5,2}, i_{2,3,3}, i_{2,4,3}, i_{2,5,3}]$ for three layers
- i_1:$[i_{1,1}, i_{1,2}, i_{1,5}, i_{1,6,1}, i_{1,7,1}, i_{1,8,1}, i_{1,6,2}, i_{1,7,2}, i_{1,8,2}, i_{1,6,3}, i_{1,7,3}, i_{1,8,3}, i_{1,6,4}, i_{1,7,4}, i_{1,8,4}]$ and i_2:$[i_{2,3,1}, i_{2,4,1}, i_{2,5,1}, i_{2,3,2}, i_{2,4,2}, i_{2,5,2}, i_{2,3,3}, i_{2,4,3}, i_{2,5,3}, i_{2,3,4}, i_{2,4,4}, i_{2,5,4}]$ for four layers

The first plot shows the rank variations, and second plot shows the grid of beams for practical and perfect channel estimation scenarios.

```
% To see the PMI and RI plots for the type I single or multi-panel
% codebooks, set the showType1PMIandRIPlots flag to true
showType1PMIandRIPlots = true;
% To see the PMI and RI plots for the type II codebooks, set the
% showType2PMI flag to true
showType2PMIRI = true;
if (strcmpi(reportConfig.CodebookType,'Type1SinglePanel') || strcmpi(reportConfig.CodebookType,'
    plotType1PMIAndRI(pmiPracticalPerSlot,pmiPerfectPerSlot,riPracticalPerSlot, ...
        riPerfectPerSlot,activeSlotNum,slotNumForSBType1);
elseif (strcmpi(reportConfig.CodebookType,'Type2') || strcmpi(reportConfig.CodebookType,'eType2'
```

```
    slotNumForType2PMI = 0; % 0-based
    plotType2PMIAndRI(pmiPracticalPerSlot,pmiPerfectPerSlot,riPracticalPerSlot, ...
        riPerfectPerSlot,reportConfig.PanelDimensions,reportConfig.NumberOfBeams,activeSlotNum,s
end
```





Summary and Further Exploration

This example shows how to compute downlink CSI parameters such as the CQI and PMI from type I single-panel codebooks and RI for a MIMO scenario with a TDL channel. The example also supports the computation of CSI parameters using type I multi-panel, type II, and enhanced type II codebooks.

You can modify carrier, channel, CSI-RS resource configurations, and CSI reporting configuration parameters (such as the codebook type, the mode of CQI and PMI reporting, and the subband size) and observe the variations in the computed CQI, PMI, and RI values across time (slots) and frequency (subbands).

References

[1] 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local Functions

This example uses these local functions to validate the CSI-RS configuration object and to plot the computed CQI, PMI, and RI values.

```
function validateCSIRSConfig(carrier,csirs,nTxAnts)
%   Validates the CSI-RS configuration, given the carrier specific
%   configuration object, CSI-RS configuration object, and the number of
%   transmit antennas.

    % Validate the number of CSI-RS ports
```

```
    if ~isscalar(unique(csirs.NumCSIRSPorts))
        error('nr5g:InvalidCSIRSPorts', ...
            'All the CSI-RS resources must be configured to have the same number of CSI-RS ports
    end

    % Validate the CDM lengths
    if ~iscell(csirs.CDMType)
        cdmType = {csirs.CDMType};
    else
        cdmType = csirs.CDMType;
    end
    if (~all(strcmpi(cdmType,cdmType{1})))
        error('nr5g:InvalidCSIRSCDMTypes', ...
            'All the CSI-RS resources must be configured to have the same CDM lengths.');
    end
    if nTxAnts ~= csirs.NumCSIRSPorts(1)
        error('nr5g:InvalidNumTxAnts',['Number of transmit antennas (' num2str(nTxAnts) ...
            ') must be equal to the number of CSI-RS ports (' num2str(csirs.NumCSIRSPorts(1)) ')
    end

    % Check for the overlap between the CSI-RS indices
    csirsInd = nrCSIRSIndices(carrier,csirs,"OutputResourceFormat",'cell');
    numRes = numel(csirsInd);
    csirsIndAll = cell(1,numRes);
    ratioVal = csirs.NumCSIRSPorts(1)/prod(getCDMLengths(csirs));
    for resIdx = 1:numRes
        if ~isempty(csirsInd{resIdx})
            grid = nrResourceGrid(carrier,csirs.NumCSIRSPorts(1));
            [~,tempInd] = nrExtractResources(csirsInd{resIdx},grid);
            if numel(tempInd)/numel(csirsInd{resIdx}) ~= ratioVal
                error('nr5g:OverlappedCSIRSREsSingleResource',['CSI-RS indices of resource ' ...
                    num2str(resIdx) ' must be unique. Try changing the symbol or subcarrier loca
            end
            csirsIndAll{resIdx} = tempInd(:);
            for idx = 1:resIdx-1
                overlappedInd = ismember(csirsIndAll{idx},csirsIndAll{resIdx});
                if any(overlappedInd)
                    error('nr5g:OverlappedCSIRSREsMultipleResources',['The resource elements of
                        'configured CSI-RS resources must not overlap. Try changing the symbol o
                        'subcarrier locations of CSI-RS resource ' num2str(idx) ' and resource '
                end
            end
        end
    end
end

function cdmLengths = getCDMLengths(csirs)
%   Returns the CDM lengths, given the CSI-RS configuration object.

    CDMType = csirs.CDMType;
    if ~iscell(csirs.CDMType)
        CDMType = {csirs.CDMType};
    end
    CDMTypeOpts = {'noCDM','fd-CDM2','CDM4','CDM8'};
    CDMLengthOpts = {[1 1],[2 1],[2 2],[2 4]};
    cdmLengths = CDMLengthOpts{strcmpi(CDMTypeOpts,CDMType{1})};
end
```

```
function [cqiPracticalPerSlot,subbandCQIPractical,pmiPracticalPerSlot,SINRPerSubbandPerCWPractica
    subbandCQIPerfect,pmiPerfectPerSlot,SINRPerSubbandPerCWPerfect,riPracticalPerSlot,riPerfectPe
    subbandCQIPractical,pmiPracticalPerSlot,SINRPerSubbandPerCWPractical,cqiPerfectPerSlot,subban
    SINRPerSubbandPerCWPerfect,riPracticalPerSlot,riPerfectPerSlot,reportConfig,totSlotsBinaryVec
%   Returns the CQI, PMI, and RI related variables filled with NaNs in the
%   slots where NZP-CSI-RS is not present according to the codebook type from
%   the report configuration structure. Note that the CQI, PMI, and RI
%   variables are returned as empty if there are no NZP-CSI-RS resources,
%   that is, no active slots in the entire simulation duration.

    % Compute the indices of the slots and the number of slots in which
    % NZP-CSI-RS is not present
    inactiveSlotIdx = ~totSlotsBinaryVec;
    numInactiveSlots = nnz(inactiveSlotIdx);

    if ~isempty(activeSlots)
        numCQISBs = size(cqiPracticalPerSlot,1);

        % Get the codebook type
        codebookType = 'Type1SinglePanel';
        if isfield(reportConfig,'CodebookType')
            codebookType = validatestring(reportConfig.CodebookType,{'Type1SinglePanel','Type1Mu
        end

        % Fill the CQI, PMI, and RI variables with NaNs in the slots where NZP-CSI-RS is
        % not present
        cqiPracticalPerSlot(:,:,inactiveSlotIdx) = NaN(numCQISBs,2,numInactiveSlots);
        subbandCQIPractical(:,:,inactiveSlotIdx) = NaN(numCQISBs,2,numInactiveSlots);
        SINRPerSubbandPerCWPractical(:,:,inactiveSlotIdx) = NaN(numCQISBs,2,numInactiveSlots);
        cqiPerfectPerSlot(:,:,inactiveSlotIdx) = NaN(numCQISBs,2,numInactiveSlots);
        subbandCQIPerfect(:,:,inactiveSlotIdx) = NaN(numCQISBs,2,numInactiveSlots);
        SINRPerSubbandPerCWPerfect(:,:,inactiveSlotIdx) = NaN(numCQISBs,2,numInactiveSlots);
        riPracticalPerSlot(inactiveSlotIdx) = NaN;
        riPerfectPerSlot(inactiveSlotIdx) = NaN;

        numi1Indices = 3;
        numi2Indices = 1;
        if strcmpi(codebookType,'Type1MultiPanel')
            numi1Indices = 6;
            numi2Indices = 3;
        end
        numPMISBs = size(pmiPerfectPerSlot(activeSlots(1)).i2,2);
        [pmiPerfectPerSlot(inactiveSlotIdx),pmiPracticalPerSlot(inactiveSlotIdx)] = deal(struct(
    end
end

function plotWidebandCQIAndSINR(cqiPracticalPerSlot,cqiPerfectPerSlot,SINRPerSubbandPerCWPractica
%   Plots the wideband SINR and wideband CQI values for each codeword
%   across all specified active slots (1-based) (in which the CQI is
%   reported as other than NaN) for practical and perfect channel
%   estimation cases.

    % Check if there are no slots in which NZP-CSI-RS is present
    if isempty(activeSlotNum)
        disp('No CQI data to plot, because there are no slots in which NZP-CSI-RS is present.');
        return;
    end
    cqiPracticalPerCW = permute(cqiPracticalPerSlot(1,:,:),[1 3 2]);
```

```
    cqiPerfectPerCW = permute(cqiPerfectPerSlot(1,:,:),[1 3 2]);
    SINRPerCWPractical = permute(SINRPerSubbandPerCWPractical(1,:,:),[1 3 2]);
    SINRPerCWPerfect = permute(SINRPerSubbandPerCWPerfect(1,:,:),[1 3 2]);

    % Extract wideband CQI indices for slots where NZP-CSI-RS is present
    cqiPracticalPerCWActiveSlots = cqiPracticalPerCW(1,activeSlotNum,:);
    cqiPerfectPerCWActiveSlots = cqiPerfectPerCW(1,activeSlotNum,:);
    widebandSINRPractical = 10*log10(SINRPerCWPractical(1,activeSlotNum,:));
    widebandSINRPerfect = 10*log10(SINRPerCWPerfect(1,activeSlotNum,:));

    if isempty(reshape(cqiPracticalPerCWActiveSlots(:,:,1),1,[]))
        disp('No CQI data to plot, because all CQI values are NaNs.');
        return;
    end

    figure();
    plotWBCQISINR(widebandSINRPerfect,widebandSINRPractical,211,activeSlotNum,'SINR');
    plotWBCQISINR(cqiPerfectPerCWActiveSlots,cqiPracticalPerCWActiveSlots,212,activeSlotNum,'CQI
end

function plotWBCQISINR(perfectVals,practicalVals,subplotIdx,activeSlotNum,inpText)
%   Plots the wideband SINR and wideband CQI values for each codeword
%   across all specified active slots (1-based) (in which the CQI is
%   reported as other than NaN) for practical and perfect channel
%   estimation cases.

    subplot(subplotIdx)
    plot(perfectVals(:,:,1),'r-o');
    hold on;
    plot(practicalVals(:,:,1),'b-*');
    if ~all(isnan(perfectVals(:,:,2))) % Two codewords
        hold on;
        plot(perfectVals(:,:,2),'r:s');
        hold on;
        plot(practicalVals(:,:,2),'b:d');
        title(['Wideband ' inpText ' Values for Codeword 1&2']);
        legend({'Codeword 1:Perfect channel est.','Codeword 1:Practical channel est.','Codeword 2
    else
        title(['Wideband ' inpText ' Values for Codeword 1']);
        legend({'Codeword 1:Perfect channel est.','Codeword 1:Practical channel est.'});
    end
    xlabel('Slots');
    if strcmpi(inpText,'SINR')
        units = ' in dB';
    else
        units = '';
    end
    ylabel(['Wideband ' inpText ' Values' units]);
    xticks(1:size(perfectVals,2));
    xTickLables = num2cell(activeSlotNum(:)-1);
    xticklabels(xTickLables);
    [lowerBound,upperBound] = bounds([practicalVals(:);perfectVals(:)]);
    ylim([lowerBound-1 upperBound+3.5]);
end

function plotSubbandCQIAndSINR(subbandCQIPractical,subbandCQIPerfect,SINRPerCWPractical,SINRPerCW
%   Plots the SINR and CQI values for each codeword across all the subbands
%   for practical and perfect channel estimation cases for the given slot
```

```
%   number (0-based) among all specified active slots (1-based). The
%   function does not plot the values if CQIMode is 'Wideband' or if the
%   CQI and SINR values are all NaNs in the given slot.

    % Check if there are no slots in which NZP-CSI-RS is present
    if isempty(activeSlotNum)
        disp('No CQI data to plot, because there are no slots in which NZP-CSI-RS is present.');
        return;
    end
    numSubbands = size(subbandCQIPractical,1);
    if numSubbands > 1 && ~any(nslot+1 == activeSlotNum) % Check if the CQI values are reported i
        disp(['For the specified slot (' num2str(nslot) '), CQI values are not reported. Please c
        return;
    end

    % Plot subband CQI values
    if numSubbands > 1 % Subband mode
        subbandCQIPerCWPractical = subbandCQIPractical(2:end,:,nslot+1);
        subbandCQIPerCWPerfect = subbandCQIPerfect(2:end,:,nslot+1);
        subbandSINRPerCWPractical = 10*log10(SINRPerCWPractical(2:end,:,nslot+1));
        subbandSINRPerCWPerfect = 10*log10(SINRPerCWPerfect(2:end,:,nslot+1));
        figure();
        plotSBCQISINR(subbandSINRPerCWPerfect,subbandSINRPerCWPractical,numSubbands,211,nslot,'SI
        plotSBCQISINR(subbandCQIPerCWPerfect,subbandCQIPerCWPractical,numSubbands,212,nslot,'CQI
    end
end

function plotSBCQISINR(perfectVals,practicalVals,numSubbands,subplotIdx,nslot,inpText)
%   Plots the SINR and CQI values for each codeword across all the subbands
%   for practical and perfect channel estimation cases for the given slot
%   number (0-based). The function does not plot the values if CQIMode is
%   'Wideband' or if the CQI and SINR values are all NaNs in the given
%   slot.

    subplot(subplotIdx)
    plot(perfectVals(:,1),'ro-');
    hold on;
    plot(practicalVals(:,1),'b*-');
    if ~all(isnan(perfectVals(:,2))) % Two codewords
        hold on;
        plot(perfectVals(:,2),'rs:');
        hold on;
        plot(practicalVals(:,2),'bd:');
        legend({'Codeword 1:Perfect channel est.','Codeword 1:Practical channel est.','Codeword 2
        title(['Estimated Subband ' inpText ' Values for Codeword 1&2 in Slot ' num2str(nslot)])
    else % Single codeword
        legend({'Codeword 1:Perfect channel est.','Codeword 1:Practical channel est.'});
        title(['Estimated Subband ' inpText ' Values for Codeword 1 in Slot ' num2str(nslot)]);
    end

    if strcmpi(inpText,'SINR')
        units = ' in dB';
    else
        units = '';
    end
    xlabel('Subbands');
    ylabel(['Subband ' inpText ' Values' units]);
    xticks(1:numSubbands);
```

```
    xTickLables = num2cell(1:numSubbands);
    xticklabels(xTickLables);
    xlim([0 numSubbands+1]);
    [lowerBound,upperBound] = bounds([perfectVals(:);practicalVals(:)]);
    ylim([lowerBound-1 upperBound+3.5]);
end

function plotType1PMIAndRI(pmiPracticalPerSlot,pmiPerfectPerSlot,riPracticalPerSlot,riPerfectPerS
%   Plots the RI and PMI i1 indices across all specified active slots
%   (1-based), for practical and perfect channel estimation scenarios. The
%   function also plots the i2 indices of practical and perfect channel
%   estimation scenarios across all specified active slots when the PMI
%   mode is 'Wideband' or plots i2 indices across all the subbands for the
%   specified slot number (0-based) when the PMI mode is 'Subband'.

    % Check if there are no slots in which NZP-CSI-RS is present
    if isempty(activeSlotNum)
        disp('No PMI and RI data to plot, because there are no slots in which NZP-CSI-RS is prese
        return;
    end

    numi1Indices = numel(pmiPracticalPerSlot(activeSlotNum(1)).i1);
    if numi1Indices == 6
        codebookType = 'Type1MultiPanel';
    else
        codebookType = 'Type1SinglePanel';
    end

    % Extract wideband PMI indices (i1 values) for slots where NZP-CSI-RS
    % is present
    i1PerfectValsActiveSlots = reshape([pmiPerfectPerSlot(activeSlotNum).i1],numi1Indices,[])';
    i1PracticalValsActiveSlots = reshape([pmiPracticalPerSlot(activeSlotNum).i1],numi1Indices,[]

    if isempty(i1PerfectValsActiveSlots)
        disp('No PMI and RI data to plot, because all PMI and RI values are NaNs.');
        return;
    end

    figure;
    % Plot RI
    plotRI(riPracticalPerSlot,riPerfectPerSlot,activeSlotNum,411);

    % Extract and plot i11 indices
    i11PerfectVals = i1PerfectValsActiveSlots(:,1);
    i11PracticalVals = i1PracticalValsActiveSlots(:,1);
    plotIxxIndices(i11PerfectVals,i11PracticalVals,activeSlotNum,412,'i11');

    % Extract and plot i12 indices
    i12PerfectVals = i1PerfectValsActiveSlots(:,2);
    i12PracticalVals = i1PracticalValsActiveSlots(:,2);
    plotIxxIndices(i12PerfectVals,i12PracticalVals,activeSlotNum,413,'i12');

    % Extract and plot i13 indices
    i13PerfectVals = i1PerfectValsActiveSlots(:,3);
    i13PracticalVals = i1PracticalValsActiveSlots(:,3);
    plotIxxIndices(i13PerfectVals,i13PracticalVals,activeSlotNum,414,'i13');

    % Plot the i141, i142 and i143 indices in type I multi-panel case
```

```
if strcmpi(codebookType,'Type1MultiPanel')
    figure()
    % Extract and plot i141 indices
    i141PerfectVals = i1PerfectValsActiveSlots(:,4);
    i141PracticalVals = i1PracticalValsActiveSlots(:,4);
    plotIxxIndices(i141PerfectVals,i141PracticalVals,activeSlotNum,311,'i141');

    % Extract and plot i142 indices
    i142PerfectVals = i1PerfectValsActiveSlots(:,5);
    i142PracticalVals = i1PracticalValsActiveSlots(:,5);
    plotIxxIndices(i142PerfectVals,i142PracticalVals,activeSlotNum,312,'i142');

    % Extract and plot i143 indices
    i143PerfectVals = i1PerfectValsActiveSlots(:,6);
    i143PracticalVals = i1PracticalValsActiveSlots(:,6);
    plotIxxIndices(i143PerfectVals,i143PracticalVals,activeSlotNum,313,'i143');
end

% Get the number of subbands
numSubbands = size(pmiPracticalPerSlot(activeSlotNum(1)).i2,2);
% Get the number of i2 indices according to codebook type
numi2Indices = 1;
if strcmpi(codebookType,'Type1MultiPanel')
    numi2Indices = 3;
end

% Get number of active slots
numActiveSlots = numel(activeSlotNum);
% Extract i2 values
i2PerfectVals = reshape([pmiPerfectPerSlot(activeSlotNum).i2],[numSubbands,numi2Indices,numA
i2PracticalVals = reshape([pmiPracticalPerSlot(activeSlotNum).i2],[numSubbands,numi2Indices,n

% Plot i2 values
if numSubbands == 1 % Wideband mode
    figure;

    % In type I single-panel case, there is only one i2 index. The
    % first column of i2PerfectVals and i2PracticalVals corresponds to
    % i2 index. In type I multi-panel case, the i2 values are a set of
    % three indices i20, i21, and i22. Each column of i2PerfectVals and
    % i2PracticalVals correspond to i20, i21, and i22 indices. Extract
    % and plot the respective index values
    if strcmpi(codebookType,'Type1SinglePanel')
        % Extract and plot i2 values in each slot
        i2PerfectVals = reshape(i2PerfectVals(:,1,:),[],numActiveSlots).';
        i2PracticalVals = reshape(i2PracticalVals(:,1,:),[],numActiveSlots).';
        plotIxxIndices(i2PerfectVals,i2PracticalVals,activeSlotNum,111,'i2');
    else
        % Extract and plot i20 values in each slot
        i20PerfectVals = reshape(i2PerfectVals(:,1,:),[],numActiveSlots).';
        i20PracticalVals = reshape(i2PracticalVals(:,1,:),[],numActiveSlots).';
        plotIxxIndices(i20PerfectVals,i20PracticalVals,activeSlotNum,311,'i20');

        % Extract and plot i21 values in each slot
        i21PerfectVals = reshape(i2PerfectVals(:,2,:),[],numActiveSlots).';
        i21PracticalVals = reshape(i2PracticalVals(:,2,:),[],numActiveSlots).';
        plotIxxIndices(i21PerfectVals,i21PracticalVals,activeSlotNum,312,'i21');
```

```
            % Extract and plot i22 values in each slot
            i22PerfectVals = reshape(i2PerfectVals(:,3,:),[],numActiveSlots).';
            i22PracticalVals = reshape(i2PracticalVals(:,3,:),[],numActiveSlots).';
            plotIxxIndices(i22PerfectVals,i22PracticalVals,activeSlotNum,313,'i22');
        end
    else % Subband mode
        if any(nslot+1 == activeSlotNum)

            % In subband mode, plot the PMI i2 indices corresponding to the
            % specified slot number
            figure;

            if strcmpi(codebookType,'Type1SinglePanel')
                % Extract and plot i2 values
                pmiSBi2Perfect = pmiPerfectPerSlot(nslot+1).i2(1,:);
                pmiSBi2Practical = pmiPracticalPerSlot(nslot+1).i2(1,:);
                plotI2xIndices_SB(pmiSBi2Perfect,pmiSBi2Practical,numSubbands,nslot,111,'i2');
            else
                % Extract and plot i20 values
                pmiSBi20Perfect = pmiPerfectPerSlot(nslot+1).i2(1,:);
                pmiSBi20Practical = pmiPracticalPerSlot(nslot+1).i2(1,:);
                plotI2xIndices_SB(pmiSBi20Perfect,pmiSBi20Practical,numSubbands,nslot,311,'i20')

                % Extract and plot i21 values
                pmiSBi21Perfect = pmiPerfectPerSlot(nslot+1).i2(2,:);
                pmiSBi21Practical = pmiPracticalPerSlot(nslot+1).i2(2,:);
                plotI2xIndices_SB(pmiSBi21Perfect,pmiSBi21Practical,numSubbands,nslot,312,'i21')

                % Extract and plot i22 values
                pmiSBi22Perfect = pmiPerfectPerSlot(nslot+1).i2(3,:);
                pmiSBi22Practical = pmiPracticalPerSlot(nslot+1).i2(3,:);
                plotI2xIndices_SB(pmiSBi22Perfect,pmiSBi22Practical,numSubbands,nslot,313,'i22')
            end
        else
            disp(['For the specified slot (' num2str(nslot) '), PMI i2 indices are not reported.
        end
    end
end

function plotType2PMIAndRI(pmiPracticalPerSlot,pmiPerfectPerSlot,riPracticalPerSlot,riPerfectPerS
%   Plots the grid of beams by highlighting the beams that are used for the
%   precoding matrix generation for the specified slot number (0-based),
%   for practical and perfect channel estimation scenarios.

    % Check if there are no slots in which NZP-CSI-RS is present
    if isempty(activeSlotNum)
        disp('No PMI and RI data to plot, because there are no slots in which NZP-CSI-RS is prese
        return;
    end
    plotRI(riPracticalPerSlot,riPerfectPerSlot,activeSlotNum,111);
    if ~any(nslot+1 == activeSlotNum)
        disp(['For the specified slot (' num2str(nslot) '), PMI values are not reported. Please c
    else
        pmiPractical = pmiPracticalPerSlot(nslot+1);
        pmiPerfect = pmiPerfectPerSlot(nslot+1);
        figure();
        plotType2GridOfBeams(pmiPractical,panelDims,numBeams,'Practical Channel Estimation Scenar
        hold on;
```

```
            plotType2GridOfBeams(pmiPerfect,panelDims,numBeams,'Perfect Channel Estimation Scenario'
        end
    end

    function plotRI(riPracticalPerSlot,riPerfectPerSlot,activeSlotNum,subplotIndex)
    %   Plots the RI values across all specified active slots (1-based), for
    %   practical and perfect channel estimation scenarios.

        % Get number of active slots
        numActiveSlots = numel(activeSlotNum);

        % Extract RI values for slots where NZP-CSI-RS is present
        RIPerfectValsActiveSlots = riPerfectPerSlot(activeSlotNum)';
        RIPracticalValsActiveSlots = riPracticalPerSlot(activeSlotNum)';

        if isempty(RIPerfectValsActiveSlots)
            disp('No RI data to plot, because all RI values are NaNs.');
            return;
        end

        figure;
        subplot(subplotIndex);
        plot(RIPerfectValsActiveSlots,'r-o');
        hold on;
        plot(RIPracticalValsActiveSlots,'b-*');
        xlabel('Slots')
        ylabel('RI Values');
        xticks(1:numActiveSlots);
        xTickLables = num2cell(activeSlotNum(:)-1);
        xticklabels(xTickLables);
        [~,upperBound] = bounds([RIPerfectValsActiveSlots; RIPracticalValsActiveSlots]);
        xlim([0 numActiveSlots+8]);
        ylim([0 upperBound+1]);
        yticks(0:upperBound+1);
        title('RI Values')
        legend({'Perfect channel est.','Practical channel est.'});
    end

    function plotType2GridOfBeams(PMISet,panelDims,numBeams,chEstType,subplotNum)
    %   Plots the grid of beams by highlighting the beams that are used for the
    %   type II codebook based precoding matrix generation.

        N1 = panelDims(1);
        N2 = panelDims(2);
        % Get the oversampling factors
        O1 = 4;
        O2 = 1 + 3*(N2 ~= 1);

        % Extract q1, q2 values
        qSet = PMISet.i1(1:2);
        q1 = qSet(1)-1;
        q2 = qSet(2)-1;

        % Extract i12 value
        i12 = PMISet.i1(3);
        s = 0;
        % Find the n1, n2 values for all the beams, as defined in TS 38.214
        % Section 5.2.2.2.3
```

```
n1_i12 = zeros(1,numBeams);
n2_i12 = zeros(1,numBeams);
for beamIdxI = 0:numBeams-1
    i12minussVal = i12 - s;
    xValues = numBeams-1-beamIdxI:N1*N2-1-beamIdxI;
    CValues = zeros(numel(xValues),1);
    for xIdx = 1:numel(xValues)
        if xValues(xIdx) >= numBeams-beamIdxI
            CValues(xIdx) = nchoosek(xValues(xIdx),numBeams-beamIdxI);
        end
    end
    indices = i12minussVal >= CValues;
    maxIdx = find(indices,1,'last');
    xValue = xValues(maxIdx);
    ei = CValues(maxIdx);
    s = s+ei;
    ni = N1*N2 - 1 - xValue;
    n1_i12(beamIdxI+1) = mod(ni,N1);
    n2_i12(beamIdxI+1) = (ni-n1_i12(beamIdxI+1))/N1;
end
m1 = O1*(0:N1-1) + q1;
m2 = O2*(0:N2-1) + q2;

% Calculate the indices of orthogonal basis set which corresponds to
% the reported i12 value
m1_LBeams = O1*(n1_i12) + q1;
m2_LBeams = O2*(n2_i12) + q2;
OrthogonalBeams = [repmat(m1,1,length(m2));reshape(repmat(m2,length(m1),1),1,[])]';

% Plot the grid of beams
numCirlcesInRow = N1*O1;
numCirlcesInCol = N2*O2;
subplot(2,1,subplotNum);
circleRadius = 1;
for colIdx = 0:numCirlcesInCol-1
    for rowIdx = 0:numCirlcesInRow-1
        p = nsidedpoly(1000, 'Center', [2*rowIdx 2*colIdx], 'Radius', circleRadius);
        if any(prod(OrthogonalBeams == [rowIdx colIdx],2))
            h2 = plot(p, 'FaceColor', 'w','EdgeColor','r','LineWidth',2.5);
            hold on;
            if any(prod([m1_LBeams' m2_LBeams'] == [rowIdx colIdx],2))
                h3 = plot(p, 'FaceColor', 'g','LineStyle','-.');
            end
        else
            h1 = plot(p, 'FaceColor', 'w');
        end
        hold on;
    end
end
rowLength = 2*circleRadius*O1;
colLength = 2*circleRadius*O2;
for n2 = 0:N2-1
    for n1 = 0:N1-1
        x1 = -1*circleRadius + rowLength*n1;
        x2 = x1 + rowLength;
        y1 = -1*circleRadius + colLength*n2;
        y2 = y1 + colLength;
        x = [x1, x2, x2, x1, x1];
```

```
            y = [y1, y1, y2, y2, y1];
            plot(x, y, 'b-', 'LineWidth', 2);
            hold on;
        end
    end

    xlabel('N1O1 beams');
    ylabel('N2O2 beams');
    axis equal;
    set(gca,'xtick',[],'ytick',[]);
    legend([h1 h2 h3],{'Oversampled DFT beams',['Orthogonal basis set with [q1 q2] = [' num2str(
    title(['Grid of Beams or DFT Vectors for ' chEstType]);
end

function plotIxxIndices(ixxPerfectVals,ixxPracticalVals,activeSlotNum,subplotInp,pmiIdxType)
%   Plots i11, i12, i13 indices in case of type I single-panel codebooks
%   and plots i141, i142, and i143 in case of type I multi-panel codebooks.

    % Plot ixx values
    subplot(subplotInp)
    plot(ixxPerfectVals,'r-o');
    hold on;
    plot(ixxPracticalVals,'b-*');
    xlabel('Slots')
    ylabel([pmiIdxType ' Indices']);
    % Get number of active slots
    numActiveSlots = numel(activeSlotNum);
    xticks(1:numActiveSlots);
    xTickLables = num2cell(activeSlotNum(:)-1);
    xticklabels(xTickLables);
    [lowerBound,upperBound] = bounds([ixxPerfectVals; ixxPracticalVals]);
    xlim([0 numActiveSlots+8]);
    ylim([lowerBound-2 upperBound+2]);
    title(['PMI: ' pmiIdxType ' Indices']);
    legend({'Perfect channel est.','Practical channel est.'});
end

function plotI2xIndices_SB(pmiSBi2Perfect,pmiSBi2Practical,numSubbands,nslot,subplotInp,pmiIdxTy
%   Plots i2 indices in case of type I single-panel codebooks and plots
%   i20, i21, and i22 in case of type I multi-panel codebooks.

    subplot(subplotInp)
    plot(pmiSBi2Perfect,'r-o');
    hold on;
    plot(pmiSBi2Practical,'b-*');
    title(['PMI: ' pmiIdxType ' Indices for All Subbands in Slot ' num2str(nslot)]);
    xlabel('Subbands')
    ylabel([pmiIdxType ' Indices']);
    xticks(1:numSubbands);
    xticklabels(num2cell(1:numSubbands));
    [lowerBound,upperBound] = bounds([pmiSBi2Perfect pmiSBi2Practical]);
    yticks(lowerBound:upperBound);
    yticklabels(num2cell(lowerBound:upperBound));
    xlim([0 numSubbands+1])
    ylim([lowerBound-1 upperBound+1]);
```

```
    legend({'Perfect channel est.','Practical channel est.'});
end
```

See Also

Functions
`nrCSIRS` | `nrCSIRSIndices` | `nrResourceGrid` | `nrOFDMModulate` | `nrOFDMDemodulate` | `nrOFDMInfo`

Objects
`nrCarrierConfig` | `nrCSIRSConfig` | `nrTDLChannel`

Related Examples

- “NR PDSCH Throughput” on page 1-58
- “SNR Definition Used in Link Simulations” on page 5-86

Include Path Loss in NR Link-Level Simulations

This example shows how to include path loss, transmit power, and receive noise in 5G NR link-level simulations to study the impact of these parameters in the performance of a 5G link.

Introduction

Link-level simulations measure block error rate and throughput across a range of average signal-to-noise ratio (SNR) values at the receive side. This example shows how to calculate the SNR from parameters such as the path loss, transmit power, and receiver input noise and noise figure. You can use this SNR in the "NR PDSCH Throughput" on page 1-58 and "NR PUSCH Throughput" on page 2-43 examples to study the effect of those parameters in a 5G link. In this example, you can:

1. Configure a transmitter, receiver, carrier, and propagation channel.
2. Calculate the path loss and noise power.
3. Calculate the SNR at the receiver.
4. Verify the SNR calculation.

Configure Transmitter, Receiver, and Carrier

This section configures a carrier, transmitter, and receiver with these characteristics:

- The transmitter or base station (BS) model includes the average power delivered to all antennas and the antenna height. You can specify the average transmit power of a fully allocated resource grid, but this power does not include antenna element gains. Therefore, the actual average power transmitted scales with the resource grid allocation and antenna element gains, but this power does not scale with the number of transmit antennas.
- The receiver or user equipment (UE) model includes its noise figure and the antenna temperature and height. The noise figure models the receiver internal noise, and the antenna temperature models the input noise. This receiver specifies the noise per antenna element.
- You can specify the distance between the BS and UE as a vector to calculate the SNR values at the specified distances.

```
% Configure the carrier.
simParameters.Carrier = nrCarrierConfig;
simParameters.Carrier.NSizeGrid = 51;            % Bandwidth in number of resource blocks (51 RBs
simParameters.Carrier.SubcarrierSpacing = 30;    % 15, 30, 60, 120, 240 (kHz)
simParameters.Carrier.CyclicPrefix = 'Normal';   % 'Normal' or 'Extended' (Extended CP is relevar

% Configure the carrier frequency, transmitter (BS), receiver (UE), and
% distance between the BS and UE. Specify this distance as a vector for
% multiple SNR points.
simParameters.CarrierFrequency = 3.5e9;   % Carrier frequency (Hz)
simParameters.TxHeight = 25;              % Height of the BS antenna (m)
simParameters.TxPower = 40;               % Power delivered to all antennas of the BS on a fully
simParameters.RxHeight = 1.5;             % Height of UE antenna (m)
simParameters.RxNoiseFigure = 6;          % Noise figure of the UE (dB)
simParameters.RxAntTemperature = 290;     % Antenna temperature of the UE (K)
simParameters.TxRxDistance = [5e2 9e2];   % Distance between the BS and UE (m)
```

Configure Propagation Channel

The propagation channel model includes path loss and small-scale fading. The path loss depends on the distance between the BS and UE, the carrier frequency, and other parameters that are specific to

each scenario. The fading channels can be clustered delay line (CDL) or tapped delay line (TDL) channels. In this example, the fading channel configuration determines the line-of-sight (LOS) existence between the transmitter (BS) and receiver (UE), which is required for path loss calculation. Delay profiles A, B, and C configure non-LOS channels and delay profiles D and E configure LOS channels.

Configure Path Loss

Configure the scenario and other path loss parameters of the 5G path loss model as defined in TR 38.901 Section 7.4. Alternatively, specify the free-space path loss model. The path loss model determines the average attenuation of the transmitted signal.

```
simParameters.PathLossModel = '5G-NR';        % '5G-NR' or 'fspl'
simParameters.PathLoss = nrPathLossConfig;
simParameters.PathLoss.Scenario = 'UMa';      % Urban macrocell
simParameters.PathLoss.EnvironmentHeight = 1; % Average height of the environment in UMa/UMi
```

Configure Fading Channel and LOS

Configure a CDL or TDL fading channel as defined in TR 38.901. The fading channel models short-term variations of the channel response over time and frequency.

```
simParameters.DelayProfile = 'TDL-A'; % A, B, and C profiles are NLOS channels. D and E profiles

if contains(simParameters.DelayProfile,'CDL','IgnoreCase',true)
    channel = nrCDLChannel;
    channel.DelayProfile = simParameters.DelayProfile;
    chInfo = info(channel);
    kFactor = chInfo.KFactorFirstCluster; % dB
else % TDL
    channel = nrTDLChannel;
    channel.DelayProfile = simParameters.DelayProfile;
    chInfo = info(channel);
    kFactor = chInfo.KFactorFirstTap; % dB
end

% Determine LOS between Tx and Rx based on Rician factor K.
simParameters.LOS = kFactor>-Inf;

% Determine the sample rate and FFT size that are required for this carrier.
waveformInfo = nrOFDMInfo(simParameters.Carrier);

% Calculate the maximum delay of the fading channel.
channel.SampleRate = waveformInfo.SampleRate;
chInfo = info(channel);
maxChDelay = ceil(max(chInfo.PathDelays*channel.SampleRate)) + chInfo.ChannelFilterDelay;
```

Calculate Propagation Path Loss

This section calculates the path loss between the transmitter (BS) and receiver (UE).

```
% Calculate the path loss.
if contains(simParameters.PathLossModel,'5G','IgnoreCase',true)
    txPosition = [0;0; simParameters.TxHeight];
    dtr = simParameters.TxRxDistance;
    rxPosition = [dtr; zeros(size(dtr)); simParameters.RxHeight*ones(size(dtr))];
    pathLoss = nrPathLoss(simParameters.PathLoss,simParameters.CarrierFrequency,simParameters.LOS
else % Free-space path loss
```

```
    lambda = physconst('LightSpeed')/simParameters.CarrierFrequency;
    pathLoss = fspl(simParameters.TxRxDistance,lambda);
end
```

Calculate Antenna and Receiver Noise

This section calculates the equivalent noise temperature and amplitude per receive antenna element from the input noise temperature and receiver internal noise. The noise amplitude per receive antenna is

$$N_0 = \sqrt{\frac{1}{2} k B T_e}$$

where k is the Boltzmann constant (1.3807×10^{-23} JK^{-1}). B is the bandwidth, which is equal the waveform sample rate. The equivalent noise temperature T_e is

$T_e = T_{Ant} + 290(NF - 1)$,

T_{Ant} is the input noise temperature and NF is the receiver noise figure in linear units.

```
kBoltz = physconst('Boltzmann');
NF = 10^(simParameters.RxNoiseFigure/10);
Teq = simParameters.RxAntTemperature + 290*(NF-1); % K
N0 = sqrt(kBoltz*waveformInfo.SampleRate*Teq/2.0);
```

Calculate Signal-to-Noise Ratio (SNR)

This section uses the configuration parameters to calculate the SNR per resource element (RE) resulting from the transmit power, receive noise, bandwidth, and path loss. The average receive signal per RE and the receive antenna and noise power per RE are

$$S \equiv P_{RE}^{S} = \frac{P_{Tx}}{L} \times \frac{N_{FFT}^2}{12 N_{grid}^{size}} \text{ and}$$

$$N \equiv P_{RE}^{N} = 2N_0^2 N_{FFT}.$$

P_{Tx} is the total transmitted power at the transmit antenna array input. N_{FFT} is the number of fast Fourier transform (FFT) points used for OFDM modulation. N_{grid}^{size} is the size of the OFDM grid in resource blocks. L is the path loss. $\sqrt{2}N_0$ is root-mean-square (RMS) value of the noise per receive antenna. The average SNR per RE and receive antenna is

$$\left(\frac{S}{N}\right)_{dB} = 10\log_{10}(P_{Tx}) - 10\log_{10}(L) + 10\log_{10}\left(\frac{N_{FFT}}{12 N_{grid}^{size}}\right) - 10\log_{10}\left(2N_0^2\right).$$

These are the SNR values you must use in the “NR PDSCH Throughput” on page 1-58 and “NR PUSCH Throughput” on page 2-43 examples.

```
fftOccupancy = 12*simParameters.Carrier.NSizeGrid/waveformInfo.Nfft;
simParameters.SNRIn = (simParameters.TxPower-30) - pathLoss - 10*log10(fftOccupancy) - 10*log10(2
disp(simParameters.SNRIn)
```

```
    5.4206   -4.5425
```

Create a table to display the SNR at each Tx-Rx distance

```
SNRInc = mat2cell(simParameters.SNRIn(:),length(pathLoss),1);
tSNRIn = table(simParameters.TxRxDistance(:),SNRInc{:},'VariableNames',{'Distance Tx-Rx (m)','SNR
disp(tSNRIn)
```

```
    Distance Tx-Rx (m)    SNR (dB)
    __________________    ________

           500             5.4206
           900            -4.5425
```

For more information on how 5G Toolbox™ link-level simulations define the signal-to-noise ratio (SNR), see “SNR Definition Used in Link Simulations” on page 5-86.

Appendix: Verify SNR Values

This section helps you verify the SNR values obtained and review additional details of the relationship between the path loss, transmit power, and receive noise, and the resulting SNR. For that purpose, this section simulates the transmission of a CP-OFDM signal with the specified transmit power through the propagation channel, and measure the resulting SNR at the receiver. You do not need to implement this transmission in 5G Toolbox examples. Instead, you can use the SNR expression or values obtained in previous section of this example.

For the SNR measurement, the receiver has knowledge of the channel and noise. This SNR measurement uses independent channel realizations (block fading) to obtain meaningful results with the minimum number of transmission slots required.

Set the length of the simulation in terms of the number of 10ms frames.

```
NFrames = 20;
NSlots = NFrames*simParameters.Carrier.SlotsPerFrame;
nSNRPoints = length(pathLoss);                          % Number of SNR points

% Initialize measurements and create auxiliary variables.
nTxAnt = chInfo.NumTransmitAntennas;
nRxAnt = chInfo.NumReceiveAntennas;
[powSignalRE,powSignal,powNoiseRE,powNoise] = deal(zeros(nSNRPoints,nRxAnt,NSlots));
pgains = zeros(length(chInfo.PathDelays),nTxAnt,nRxAnt,nSNRPoints,NSlots);
scs = simParameters.Carrier.SubcarrierSpacing;
nSizeGrid = simParameters.Carrier.NSizeGrid;
nfft = waveformInfo.Nfft;

% Reset the random generator for reproducibility.
rng('default');

% Transmit a CP-OFDM waveform through the channel and measure the SNR for
% each distance between Tx and Rx (path loss values).
for pl = 1:length(pathLoss)
    carrier = simParameters.Carrier;
    for slot = 0:NSlots-1
        slotIdx = slot+1;
        carrier.NSlot = slot;

        % Change random seed to generate an independent realization of the
        % channel in every time slot (block fading).
        release(channel);
```

```
            channel.Seed = slot;

            % Create the OFDM resource grid and allocate random QPSK symbols.
            txgrid = nrResourceGrid(carrier,nTxAnt);
            txgrid(:) = nrSymbolModulate(randi([0 1],numel(txgrid)*2,1),'QPSK');

            % Perform CP-OFDM modulation.
            txWaveform = nrOFDMModulate(txgrid,scs,slot);

            % Calculate the amplitude of the transmitted signal. The FFT and
            % resource grid size scaling normalize the signal power. The
            % transmitter distributes the power across all antennas equally,
            % simulating the effect of unit norm beamformer.
            signalAmp = db2mag(simParameters.TxPower-30)*sqrt(nfft^2/(nSizeGrid*12*nTxAnt));
            txWaveform = signalAmp*txWaveform;

            % Pad the signal with zeros to ensure that a full slot is available
            % to the receiver after synchronization.
            txWaveform = [txWaveform; zeros(maxChDelay, size(txWaveform,2))]; %#ok<AGROW>

            % Pass the signal through fading channel.
            [rxWaveform,pathGains,sampleTimes] = channel(txWaveform);
            pgains(:,:,:,pl,slotIdx) = pathGains(1,:,:,:);

            % Apply path loss to the signal.
            rxWaveform = rxWaveform*db2mag(-pathLoss(pl));

            % Generate AWGN.
            noise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));

            % Perform perfect synchronization.
            pathFilters = getPathFilters(channel); % Path filters for perfect timing estimation
            offset = nrPerfectTimingEstimate(pathGains,pathFilters);
            rxWaveform = rxWaveform(1+offset:end,:);
            noise = noise(1+offset:end,:);

            % Perform CP-OFDM demodulation of the received signal and noise.
            ngrid = nrOFDMDemodulate(carrier,noise);
            rxgrid = nrOFDMDemodulate(carrier,rxWaveform);

            % Measure the RE and overall power of the received signal and noise.
            powSignalRE(pl,:,slotIdx) = rms(rxgrid,[1 2]).^2/nfft^2;
            powSignal(pl,:,slotIdx) = powSignalRE(pl,:,slotIdx)*nSizeGrid*12;

            powNoiseRE(pl,:,slotIdx) = rms(ngrid,[1 2]).^2/nfft^2;
            powNoise(pl,:,slotIdx) = powNoiseRE(pl,:,slotIdx)*nfft;
        end
    end
```

You can calculate the overall SNR and SNR per RE by using the obtained received signal and noise power. The difference between the overall SNR and SNR per RE is equal to the FFT spectrum occupancy ratio $12N_{\mathrm{grid}}^{\mathrm{size}}/N_{\mathrm{FFT}}$, which results from the spreading of noise energy across all FFT bins, including those bins beyond the frequency span of the resource grid.

```
fprintf('The resource grid uses %.1f %% of the FFT size, introducing a %.1f dB SNR gain.\n', fftO
```

```
The resource grid uses 59.8 % of the FFT size, introducing a 2.2 dB SNR gain.
```

```
% Correct CDL/TDL average gain.
Gf = permute(mean(sum(rms(pgains,5).^2,1),2),[4 3 1 2]); % Correction factor

% Calculate overall SNR and SNR per RE.
SNRo  = 10*log10(mean(powSignal,3)./mean(powNoise,3)) - 10*log10(Gf);
SNRre = 10*log10(mean(powSignalRE,3)./mean(powNoiseRE,3)) - 10*log10(Gf);

% Create a table to display the results.
SNRrec = mat2cell(SNRre,nSNRPoints,ones(nRxAnt,1));
tSNRre = table(simParameters.TxRxDistance(:),SNRrec{:},'VariableNames',["Distance (m)", "SNR RxA
disp(tSNRre)
```

```
    Distance (m)    SNR RxAnt1    SNR RxAnt2
    ____________    __________    __________

        500           5.4398        5.4816
        900          -4.5261       -4.4795
```

In addition to small-scale fading, CDL and TDL channels model other aspects such as antenna polarization, antenna gain, antenna correlation, and normalization by the number of received antennas, that affect the average propagation loss. The correction factor Gf applied to the SNR compensates for this average propagation loss introduced by the CDL or TDL channel and matches the measurement with the SNR definition in 5G Toolbox link-level simulations.

See Also

Functions
`nrPathLoss`

Objects
`nrPathLossConfig`

More About

- “SNR Definition Used in Link Simulations” on page 5-86

SNR Definition Used in Link Simulations

This example shows how 5G Toolbox™ link-level simulations define the signal-to-noise ratio (SNR).

SNR Definition

5G Toolbox™ link examples (“NR PDSCH Throughput” on page 1-58 and “NR PUSCH Throughput” on page 2-43) introduce AWGN to the received signal in the time domain, after the fading channel and before OFDM demodulation.

The examples define the SNR as the average SNR per resource element (RE) per receive antenna. REs are defined in the resource grid (that is, in the frequency domain). To achieve the desired SNR, the examples introduce an equivalent noise level in the time domain.

The SNR is defined as

$$SNR = \frac{S_{\mathrm{RE}}}{N_{\mathrm{RE}}}.$$

S_{RE} and N_{RE} are the average signal power per RE per receive antenna and the average noise power per RE per receive antenna, respectively. N_{RE} models the AWGN that is added to the signal.

For a signal x with discrete Fourier transform (DFT) X, Parseval's theorem states

$$\sum_{n=1}^{N_{\mathrm{FFT}}} |x_n|^2 = \frac{1}{N_{\mathrm{FFT}}} \sum_{k=1}^{N_{\mathrm{FFT}}} |X_k|^2.$$

N_{FFT} is the FFT length. Divide the equation by N_{FFT} to get the average signal power

$$S = \frac{1}{N_{\mathrm{FFT}}} \sum_{n=1}^{N_{\mathrm{FFT}}} |x_n|^2 = \frac{1}{N_{\mathrm{FFT}}^2} \sum_{k=1}^{N_{\mathrm{FFT}}} |X_k|^2 = \frac{1}{N_{\mathrm{FFT}}} X_{\mathrm{RMS}}^2.$$

In 5G, the signal of interest does not use all FFT bins (or REs) because of guard bands or zero padding. Additionally, the signal allocation can occupy only a part of the available grid. If the signal uses only K_S bins (or REs) of the FFT, the signal power is

$$S = \frac{1}{N_{\mathrm{FFT}}} \sum_{n=1}^{N_{\mathrm{FFT}}} |x_n|^2 = \frac{1}{N_{\mathrm{FFT}}^2} \sum_{k=1}^{K_S} |X_k|^2 = \frac{K_S}{{N_{\mathrm{FFT}}}^2} X_{\mathrm{RMS}}^2.$$

K_S is the number of nonzero power REs per OFDM symbol.

The signal power per RE is

$$S_{\mathrm{RE}} = \frac{S}{K_S}.$$

The noise power per RE is

$$N_{\mathrm{RE}} = \frac{N}{N_{\mathrm{FFT}}}.$$

Because the noise is added in the time domain, the noise occupies all bins, not just the allocated REs. Therefore, the noise power, N, is divided by N_{FFT} and not K_S.

Considering these definitions, the SNR becomes

$$SNR = \frac{S_{\mathrm{RE}}}{N_{\mathrm{RE}}} = \frac{\frac{K_S}{K_S N_{\mathrm{FFT}}{}^2} X_{\mathrm{RMS}}^2}{\frac{N}{N_{\mathrm{FFT}}}} = \frac{X_{\mathrm{RMS}}^2}{N_{\mathrm{FFT}} N}.$$

The 5G Toolbox link examples assume that $X_{\mathrm{RMS}}^2 = 1/N_{\mathrm{Rx}}$, where N_{Rx} is the number of receive antennas. This assumption means that the overall received power over all antennas is one.

$$SNR = \frac{1/N_{\mathrm{Rx}}}{N_{\mathrm{FFT}} N} = \frac{1}{N_{\mathrm{Rx}} N_{\mathrm{FFT}} N}$$

The noise power at the input of the OFDM demodulator is

$$N = \frac{1}{N_{\mathrm{Rx}} N_{\mathrm{FFT}} SNR}.$$

To generate noise with power N, scale the complex random samples by N_0.

$$N_0 = \sqrt{\frac{N}{2}} = \frac{1}{\sqrt{2 N_{\mathrm{Rx}} N_{\mathrm{FFT}} SNR}}$$

The factor of 2 in this equation accounts for the complex nature of the noise samples.

Most 5G Toolbox examples that model a link use this scaling factor. This scaling factor assumes that the root mean squared of the signal RE values is $X_{\mathrm{RMS}}^2 = 1/N_{\mathrm{Rx}}$. This assumption does not always apply when using a propagation channel.

The propagation channel introduces a number of effects, such as correlation, antenna polarization, and antenna element gain. These channel effects can impact the signal power at the receiver. These effects make estimating the SNR at the receiver and setting up a simulation to model a specific SNR difficult. To overcome this difficulty, the SNR definition in the link examples does not consider any of the channel effects. The SNR definition matches the SNR that is measured without a fading channel and when $X_{\mathrm{RMS}}^2 = 1/N_{\mathrm{Rx}}$.

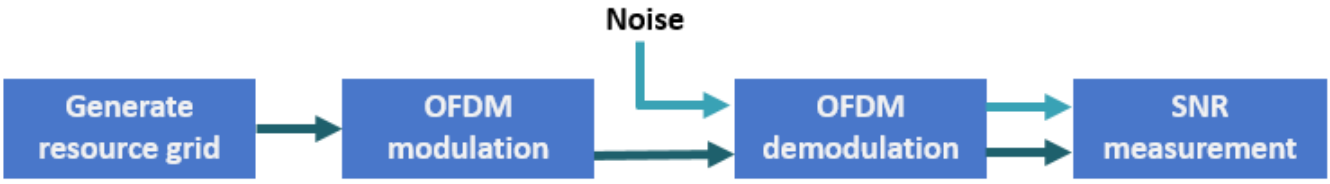
When you use a propagation channel (`nrTDLChannel` or `nrCDLChannel` object), take into account these considerations.

- Setting the `NormalizeChannelOutputs` property of the channel objects to `true` normalizes the channel outputs by the number of receive antennas such that $X_{\mathrm{RMS}}^2 = 1/N_{\mathrm{Rx}}$, as assumed in the derivation.
- Setting the `NormalizePathGains` property of the channel objects to `true` sets the total power of the average path gains to 0 dB.
- The SNR definition does not consider any of the channel effects.

SNR Verification

This section verifies the equation that is derived in the previous section. Because the introduced SNR definition does not take into account any of the channel effects, this verification does not include a propagation channel.

This figure shows the setup to measure the SNR per RE per antenna.



This setup implements these steps.

- Generate a resource grid with physical downlink shared channel (PDSCH) symbols.
- OFDM-modulate the grid.
- Generate AWGN.
- OFDM-demodulate the received signal and the noise separately.
- Measure the power of the signal and the noise per RE per antenna.
- Calculate and display the SNR.

Specify the desired SNR in dB.

```
SNRdB = 0;
rng("default") % Set default random number generator for repeatability
```

Set the number of transmit and receive antennas. Because no channel exists, assume that the number of transmit and receive antennas is the same.

```
nTxAnts = 2;
nRxAnts = nTxAnts;
```

Specify the carrier parameters.

```
carrier = nrCarrierConfig;
carrier.NSizeGrid = 52;          % Grid size in resource blocks
carrier.SubcarrierSpacing = 15;  % Subcarrier spacing

waveformInfo = nrOFDMInfo(carrier); % Waveform information

pdsch = nrPDSCHConfig;
```

```
pdsch.Modulation = "16QAM";
pdsch.PRBSet = 0:(carrier.NSizeGrid-1); % PDSCH allocation
```

Create a norm-one precoding vector that is normalized by the number of layers.

```
w = (1/sqrt(pdsch.NumLayers))*ones(pdsch.NumLayers,nTxAnts);
```

To achieve the desired SNR, calculate the noise scaling factor: $N_0 = \frac{1}{\sqrt{2N_{\mathrm{Rx}}N_{\mathrm{FFT}}SNR}}$.

```
SNR = 10^(SNRdB/10);
N0 = 1/sqrt(2.0*nRxAnts*double(waveformInfo.Nfft)*SNR);
```

Generate precoded PDSCH symbols.

```
[pdschIndices,pdschInfo] = nrPDSCHIndices(carrier,pdsch);
pdschBits = randi([0 1],pdschInfo.G,1);
pdschSymbols = nrPDSCH(carrier,pdsch,pdschBits);
pdschSymbolsPrecoded = pdschSymbols*w;
```

Create a resource grid and map the precoded PDSCH symbols to the resource grid.

```
pdschGrid = nrResourceGrid(carrier,nTxAnts);
[~,pdschAntIndices] = nrExtractResources(pdschIndices,pdschGrid);
pdschGrid(pdschAntIndices) = pdschSymbolsPrecoded;
```

OFDM-modulate.

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

Assume no channel exists. Because the SNR definition assumes that $X_{\mathrm{RMS}}^2 = 1/N_{\mathrm{Rx}}$, normalize the received signal by the number of receive antennas.

```
rxWaveform = txWaveform/sqrt(nRxAnts);
```

Generate AWGN.

```
rxNoise = N0*complex(randn(size(rxWaveform)),randn(size(rxWaveform)));
```

OFDM-demodulate the received signal (without noise), and extract the PDSCH symbols from the received grid to calculate S_{RE}.

```
% OFDM demodulation
rxSignalGrid = nrOFDMDemodulate(carrier,rxWaveform);

% PDSCH symbols extraction
rxPDSCHSymbols = rxSignalGrid(pdschAntIndices);
```

Measure the received signal power per RE, S_{RE}, and the noise power per RE, N_{RE}.

$$S_{\mathrm{RE}} = \frac{S}{K_S} = \frac{K_S}{K_S{N_{\mathrm{FFT}}}^2}X_{\mathrm{RMS}}^2 = \frac{1}{{N_{\mathrm{FFT}}}^2}X_{\mathrm{RMS}}^2$$

$$N_{\mathrm{RE}} = \frac{N}{N_{\mathrm{FFT}}}$$

Verify that the measured SNR values approximate the specified SNR parameter.

```
Sre = (1/waveformInfo.Nfft.^2)*rms(rxPDSCHSymbols).^2;
Nre = (1/waveformInfo.Nfft)*rms(rxNoise).^2;
for n=1:nRxAnts
    disp("Received signal power per RE antenna " + string(n) + " = " + string(pow2db(Sre(n))+30)
    disp("Received noise power per RE antenna " + string(n) + " = " + string(pow2db(Nre(n))+30) -
    disp("SNR (antenna " + string(n) + ") = " + string(pow2db(Sre(n)/Nre(n))) + " dB");
end
```

```
Received signal power per RE antenna 1 = -33.186 dBm

Received noise power per RE antenna 1 = -33.2432 dBm

SNR (antenna 1) = 0.057195 dB

Received signal power per RE antenna 2 = -33.186 dBm

Received noise power per RE antenna 2 = -33.2371 dBm

SNR (antenna 2) = 0.051146 dB
```

See Also

Related Examples

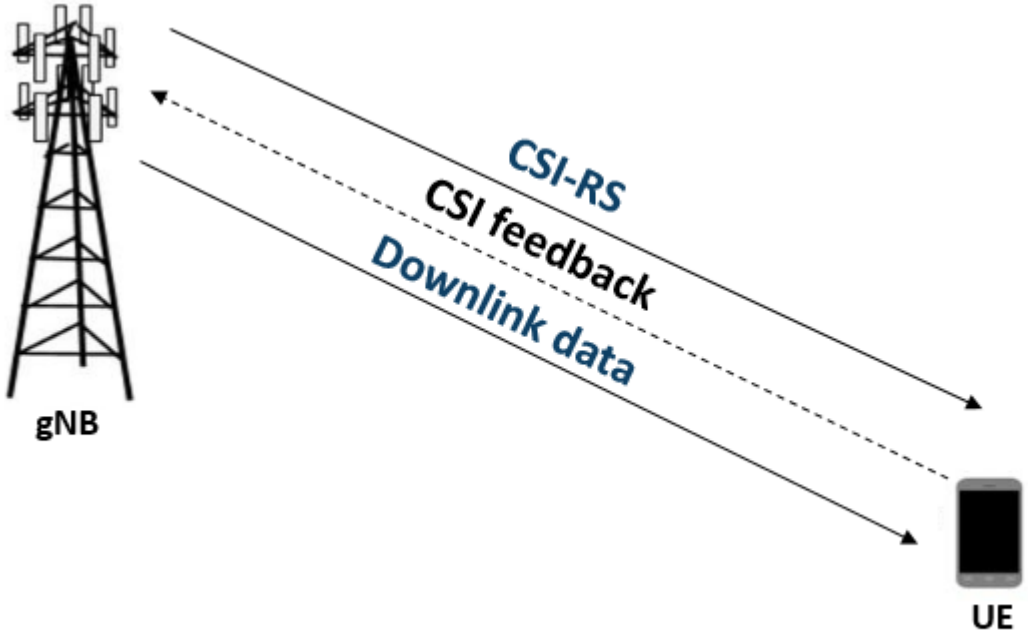
- “NR PDSCH Throughput” on page 1-58
- “NR PUSCH Throughput” on page 2-43

CSI Feedback with Autoencoders

This example shows how to use an autoencoder neural network to compress downlink channel state information (CSI) over a clustered delay line (CDL) channel. CSI feedback is in the form of a raw channel estimate array.

Introduction

In conventional 5G radio networks, CSI parameters are quantities related to the state of a channel that are extracted from the channel estimate array. The CSI feedback includes several parameters, such as the Channel Quality Indication (CQI), the precoding matrix indices (PMI) with different codebook sets, and the rank indicator (RI). The UE uses the CSI reference signal (CSI-RS) to measure and compute the CSI parameters. The user equipment (UE) reports CSI parameters to the access network node (gNB) as feedback. Upon receiving the CSI parameters, the gNB schedules downlink data transmissions with attributes such as modulation scheme, code rate, number of transmission layers, and MIMO precoding. This figure shows an overview of a CSI-RS transmission, CSI feedback, and the transmission of downlink data that is scheduled based on the CSI parameters.



The UE processes the channel estimate to reduce the amount of CSI feedback data. As an alternative approach, the UE compresses and feeds back the channel estimate array. After receipt, the gNB decompresses and processes the channel estimate to determine downlink data link parameters. The compression and decompression can be achieved using an autoencoder neural network [1 on page 5-118, 2 on page 5-118]. This approach eliminates the use of existing quantized codebook and can improve overall system performance.

This example uses a 5G downlink channel with these system parameters.

```
txAntennaSize = [2 2 2 1 1]; % rows, columns, polarizations, panels
rxAntennaSize = [2 1 1 1 1]; % rows, columns, polarizations, panels
```

```
rmsDelaySpread = 300e-9;     % s
maxDoppler = 5;              % Hz
nSizeGrid = 52;              % Number resource blocks (RB)
                             % 12 subcarriers per RB
subcarrierSpacing = 15;      % 15, 30, 60, 120 kHz
numTrainingChEst = 15000;

% Carrier definition
carrier = nrCarrierConfig;
carrier.NSizeGrid = nSizeGrid;
carrier.SubcarrierSpacing = subcarrierSpacing
```

```
carrier =
  nrCarrierConfig with properties:

             NCellID: 1
    SubcarrierSpacing: 15
         CyclicPrefix: 'normal'
            NSizeGrid: 52
           NStartGrid: 0
                NSlot: 0
               NFrame: 0

   Read-only properties:
       SymbolsPerSlot: 14
     SlotsPerSubframe: 1
        SlotsPerFrame: 10
```

```
autoEncOpt.NumSubcarriers = carrier.NSizeGrid*12;
autoEncOpt.NumSymbols = carrier.SymbolsPerSlot;
autoEncOpt.NumTxAntennas = prod(txAntennaSize);
autoEncOpt.NumRxAntennas = prod(rxAntennaSize);
```

Generate and Preprocess Data

The first step of designing an AI-based system is to prepare training and testing data. For this example, generate simulated channel estimates and preprocess the data. Use 5G Toolbox™ functions to configure a CDL-C channel.

```
waveInfo = nrOFDMInfo(carrier);
samplesPerSlot = ...
  sum(waveInfo.SymbolLengths(1:waveInfo.SymbolsPerSlot));

channel = nrCDLChannel;
channel.DelayProfile = 'CDL-C';
channel.DelaySpread = rmsDelaySpread;       % s
channel.MaximumDopplerShift = maxDoppler;   % Hz
channel.RandomStream = "Global stream";
channel.TransmitAntennaArray.Size = txAntennaSize;
channel.ReceiveAntennaArray.Size = rxAntennaSize;
channel.ChannelFiltering = false;           % No filtering for
                                            % perfect estimate
channel.NumTimeSamples = samplesPerSlot;    % 1 slot worth of samples
channel.SampleRate = waveInfo.SampleRate;
```

Simulate Channel

Run the channel and get the perfect channel estimate, `Hest`.

```
[pathGains,sampleTimes] = channel();
pathFilters = getPathFilters(channel);
offset = nrPerfectTimingEstimate(pathGains,pathFilters);
Hest = nrPerfectChannelEstimate(carrier,pathGains,pathFilters, ...
  offset,sampleTimes);
```

The channel estimate matrix is an $\left[N_{\mathrm{subcarriers}}\, N_{\mathrm{symbols}}\ N_{\mathrm{rx}}\, N_{\mathrm{tx}}\right]$ array for each slot.

```
[nSub,nSym,nRx,nTx] = size(Hest)
```

```
nSub = 624
```

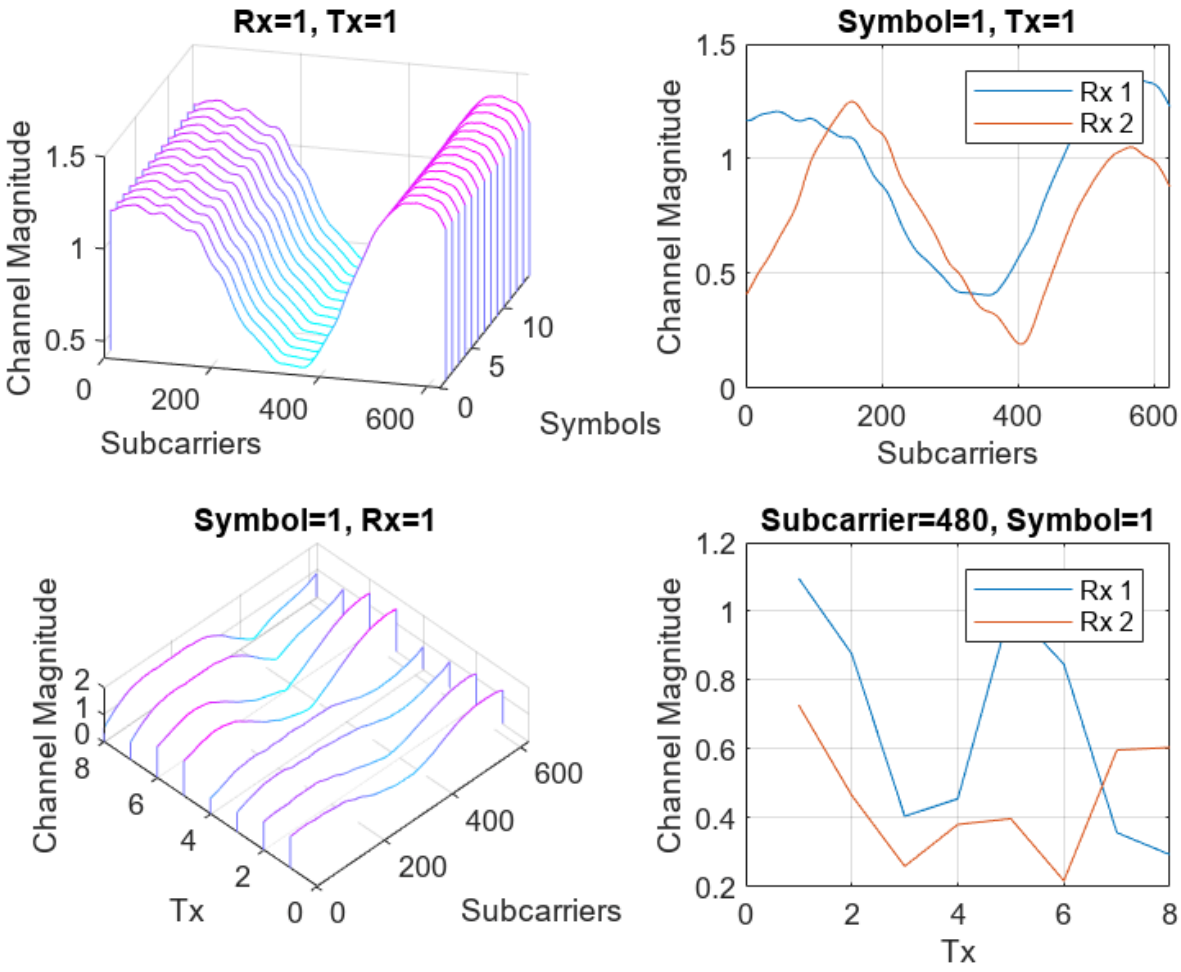
```
nSym = 14
```

```
nRx = 2
```

```
nTx = 8
```

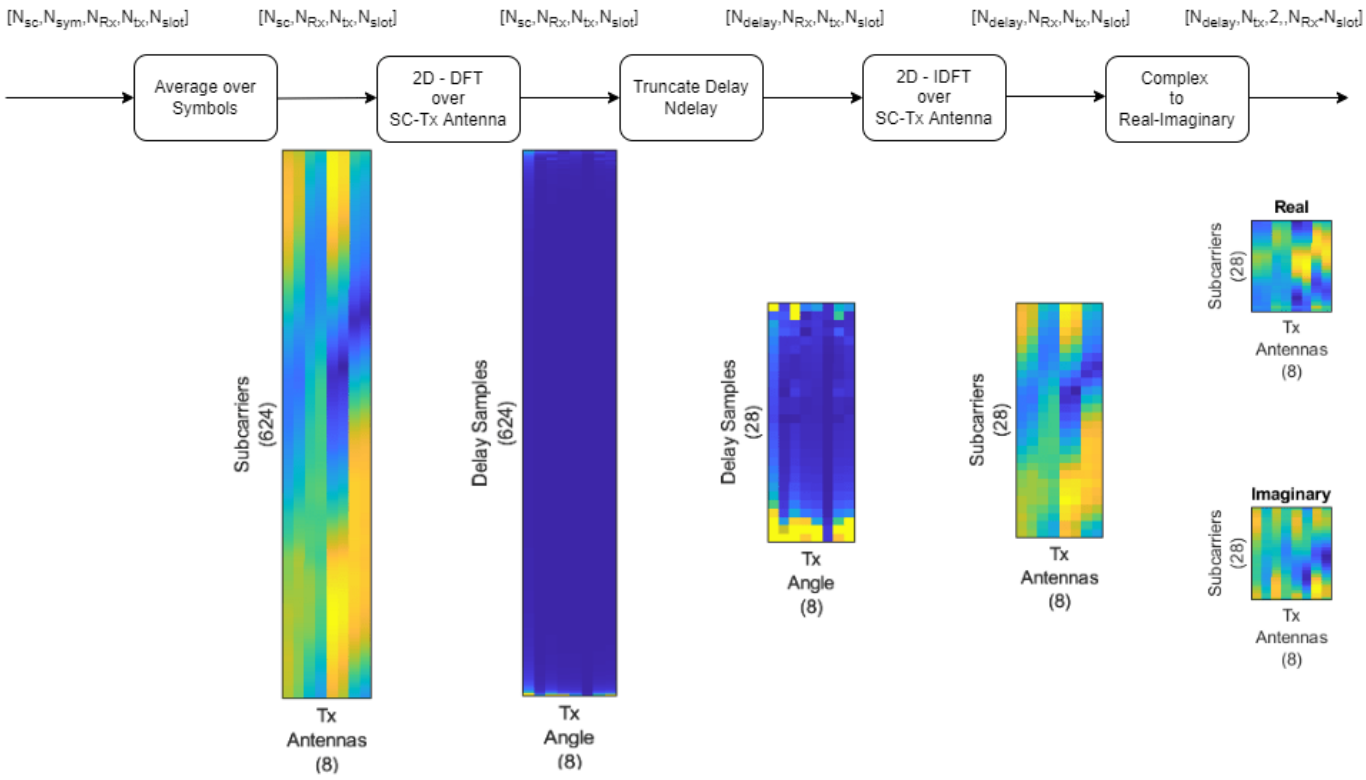
Plot the channel response. The upper left plot shows the channel frequency response as a function of time (symbols) for receive antenna 1 and transmit antenna 1. The lower left plot shows the channel frequency response as a function of transmit antennas for symbol 1 and receive antenna 1. The upper right plot shows the channel frequency response for all receive antennas for symbol 1 and transmit antenna 1. The lower right plot shows the change in channel magnitude response as a function of transmit antennas for all receive antennas for subcarrier 400 and symbol 1.

```
plotChannelResponse(Hest)
```



Preprocess Channel Estimate

Preprocess the channel estimate to reduce the size and convert it to a real-valued array. This figure shows the channel estimate reduction preprocess.



Assume that the channel coherence time is much larger than the slot time. Average the channel estimate over a slot and obtain a $[N_{\text{subcarriers}}\ 1\ N_{\text{rx}}\ N_{\text{tx}}]$array.

```
Hmean = mean(Hest,2);
```

To enable operation on subcarriers and Tx antennas, move the Tx and Rx antenna dimensions to the second and third dimensions, respectively.

```
Hmean = permute(Hmean,[1 4 3 2]);
```

To obtain the delay-angle representation of the channel, apply a 2-D discrete Fourier transform (DFT) over subcarriers and Tx antennas for each Rx antenna and slot. To demonstrate the workflow and reduce runtime, this subsection processes Rx channel 1 only.

```
Hdft2 = fft2(Hmean(:,:,1));
```

Since the multipath delay in the channel is limited, truncate the delay dimension to remove values that do not carry information. The sampling period on the delay dimension is $T_{delay} = 1/(N_{subcarriers} * F_{ss})$, where F_{ss} is subcarrier spacing. The expected RMS delay spread in delay samples is τ_{RMS}/T_{delay}, where τ_{RMS} is the RMS delay spread of the channel in seconds.

```
Tdelay = 1/(autoEncOpt.NumSubcarriers*carrier.SubcarrierSpacing*1e3);
rmsTauSamples = channel.DelaySpread / Tdelay;
maxTruncationFactor = floor(autoEncOpt.NumSubcarriers / rmsTauSamples);
```

Truncate the channel estimate to an even number of samples that is 10 times the expected RMS delay spread. Increasing the `truncationFactor` value can decrease the performance loss due to preprocessing. But, doing so increases the neural network complexity, number of required training

data points, and training time. A neural network with more learnable parameters might not converge to a better solution.

```
truncationFactor = 10;
maxDelay = round((channel.DelaySpread/Tdelay)*truncationFactor/2)*2
```

```
maxDelay = 28
```

```
autoEncOpt.MaxDelay = maxDelay;
```

Calculate the truncation indices and truncate the channel estimate.

```
midPoint = floor(nSub/2);
lowerEdge = midPoint - (nSub-maxDelay)/2 + 1;
upperEdge = midPoint + (nSub-maxDelay)/2;
Htemp = Hdft2([1:lowerEdge-1 upperEdge+1:end],:);
```

To get back to the subcarriers-Tx antennas domain, apply a 2-D inverse discrete Fourier transform (IDFT) to the truncated array [2 on page 5-118]. This process effectively decimates the channel estimate in the subcarrier axis.

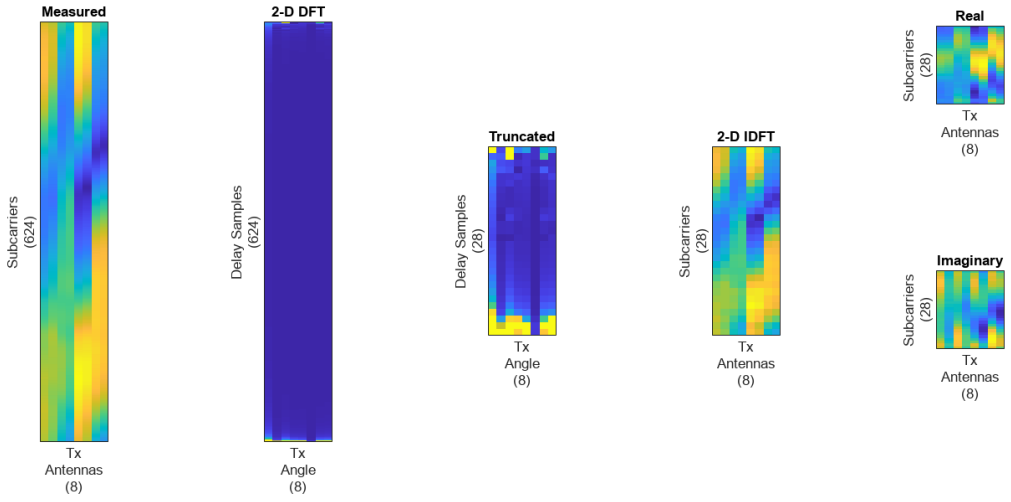
```
Htrunc = ifft2(Htemp);
```

Separate the real and imaginary parts of the channel estimate to obtain a $\left[N_{\mathrm{delay}}\ N_{\mathrm{tx}}\ 2\right]$ array.

```
HtruncReal = zeros(maxDelay,nTx,2);
HtruncReal(:,:,1) = real(Htrunc);
HtruncReal(:,:,2) = imag(Htrunc); %#ok<NASGU>
```

Plot the channel estimate signal through the preprocessing steps. Images are scaled to help visualization.

```
plotPreprocessingSteps(Hmean(:,:,1),Hdft2,Htemp,Htrunc,nSub,nTx, ...
  maxDelay)
```



Prepare Data in Bulk

The `helperCSINetTrainingData` helper function generates `numTrainingChEst` of preprocessed $\left[N_{\mathrm{delay}}\ N_{\mathrm{tx}}\ 2\right]$ channel estimates by using the process described in this section. The function saves

each $\left[N_{\mathrm{delay}}\ N_{\mathrm{tx}}\ 2\right]$ channel estimate as an individual file in the `dataDir` with the prefix of `trainingDataFilePrefix`. If Parallel Computing Toolbox™ is available, `helperCSINetTrainingData` function uses `parfor` to parallelize data generation. Data generation takes less than three minutes on a PC with Intel® Xeon® W-2133 CPU @ 3.60GHz and running in parallel on six workers.

```
dataDir = fullfile(exRoot(),"Data");
trainingDataFilePrefix = "nr_channel_est";
if validateTrainingFiles(dataDir,trainingDataFilePrefix, ...
    numTrainingChEst,autoEncOpt,channel,carrier) == false
  disp("Starting training data generation")
  tic
  autoEncOpt.Normalization = false;  % Do not normalize data yet

  helperCSINetTrainingData(dataDir,trainingDataFilePrefix, ...
    numTrainingChEst,carrier,channel,autoEncOpt);
  t = seconds(toc);
  t.Format = "hh:mm:ss";
  disp(string(t) + " - Finished training data generation")
end
```

```
Starting training data generation

6 workers running
00:00:12 -  8% Completed
00:00:23 - 16% Completed
00:00:35 - 24% Completed
00:00:46 - 32% Completed
00:00:58 - 40% Completed
00:01:09 - 48% Completed
00:01:21 - 56% Completed
00:01:32 - 64% Completed
00:01:44 - 72% Completed
00:01:56 - 80% Completed
00:02:07 - 88% Completed
00:02:19 - 96% Completed

00:02:26 - Finished training data generation
```

Create a `signalDatastore` object to access the data. The signal datastore uses individual files for each data point.

```
sds = signalDatastore( ...
  fullfile(dataDir,"processed",trainingDataFilePrefix+"_*"));
```

Load data into memory, calculate the mean value and standard deviation, and then use the mean and standard deviation values to normalize the data.

```
HtruncRealCell = readall(sds);
HtruncReal = cat(4,HtruncRealCell{:});
meanVal = mean(HtruncReal,'all')
```

```
meanVal = single
    -0.0236
```

```
stdVal = std(HtruncReal,[],'all')
```

```
stdVal = single
    16.0657
```

Separate the data into training, validation, and test sets. Also, normalize the data to achieve zero mean and a target standard deviation of 0.0212, which restricts most of the data to the range of [-0.5 0.5].

```
N = size(HtruncReal, 4);
numTrain = floor(N*10/15)
```

```
numTrain = 10000
```

```
numVal = floor(N*3/15)
```

```
numVal = 3000
```

```
numTest = floor(N*2/15)
```

```
numTest = 2000
```

```
targetStd = 0.0212;
HTReal = (HtruncReal(:,:,:,1:numTrain)-meanVal) ...
  /stdVal*targetStd+0.5;
HVReal = (HtruncReal(:,:,:,numTrain+(1:numVal))-meanVal) ...
  /stdVal*targetStd+0.5;
HTestReal = (HtruncReal(:,:,:,numTrain+numVal+(1:numTest))-meanVal) ...
  /stdVal*targetStd+0.5;
autoEncOpt.MeanVal = meanVal;
autoEncOpt.StdValue = stdVal;
autoEncOpt.TargetSTDValue = targetStd; %#ok<STRNU>
```

Define and Train Neural Network Model

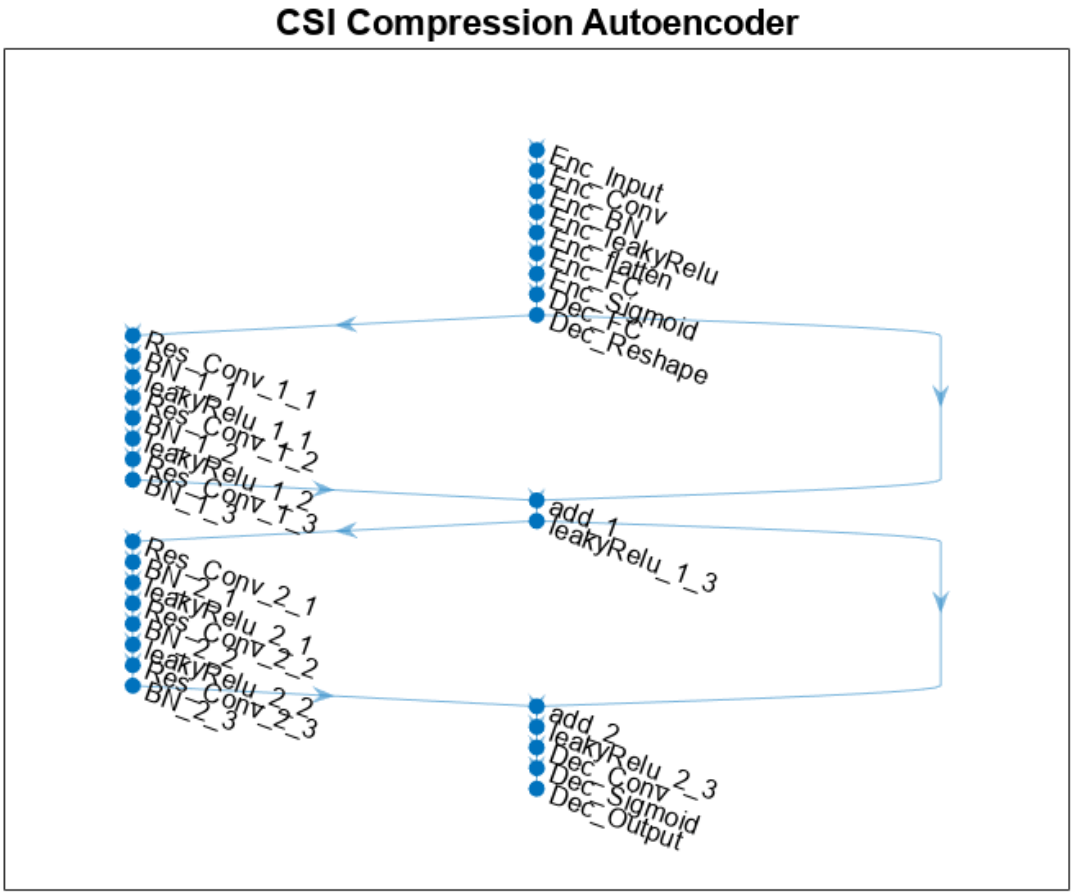
The second step of designing an AI-based system is to define and train the neural network model.

Define Neural Network

This example uses a modified version of the autoencoder neural network proposed in [1 on page 5-118].

```
inputSize = [maxDelay nTx 2];  % Third dimension is real and imaginary parts
nLinear = prod(inputSize);
nEncoded = 64;

autoencoderLGraph = layerGraph([ ...
    % Encoder
    imageInputLayer(inputSize,"Name","Htrunc", ...
      "Normalization","none","Name","Enc_Input")

    convolution2dLayer([3 3],2,"Padding","same","Name","Enc_Conv")
    batchNormalizationLayer("Epsilon",0.001,"MeanDecay",0.99, ...
      "VarianceDecay",0.99,"Name","Enc_BN")
    leakyReluLayer(0.3,"Name","Enc_leakyRelu")

    flattenLayer("Name","Enc_flatten")

    fullyConnectedLayer(nEncoded,"Name","Enc_FC")

    sigmoidLayer("Name","Enc_Sigmoid")

    % Decoder
    fullyConnectedLayer(nLinear,"Name","Dec_FC")
```

```
    functionLayer(@(x)dlarray(reshape(x,maxDelay,nTx,2,[]),'SSCB'), ...
      "Formattable",true,"Acceleratable",true,"Name","Dec_Reshape")
    ]);

autoencoderLGraph = ...
  helperCSINetAddResidualLayers(autoencoderLGraph, "Dec_Reshape");

autoencoderLGraph = addLayers(autoencoderLGraph, ...
    [convolution2dLayer([3 3],2,"Padding","same","Name","Dec_Conv") ...
    sigmoidLayer("Name","Dec_Sigmoid") ...
    regressionLayer("Name","Dec_Output")]);
autoencoderLGraph = ...
  connectLayers(autoencoderLGraph,"leakyRelu_2_3","Dec_Conv");

figure
plot(autoencoderLGraph)
title('CSI Compression Autoencoder')
```



Train Neural Network

Set the training options for the autoencoder neural network and train the network using the `trainNetwork` (Deep Learning Toolbox) function. Training takes less than 15 minutes on an AMD EPYC 7262 3.2 GHz 8C/16T with 8 NVIDIA RTX A5000 GPUs with `ExecutionEnvironment` set to `'multi-gpu'`. Set `trainNow` to `false` to load the pretrained network. Note that the saved network works for the following settings. If you change any of these settings, set `trainNow` to `true`.

```
txAntennaSize = [2 2 2 1 1]; % rows, columns, polarizations, panels
rxAntennaSize = [2 1 1 1 1]; % rows, columns, polarizations, panels
rmsDelaySpread = 300e-9;     % s
maxDoppler = 5;              % Hz
nSizeGrid = 52;              % Number resource blocks (RB)
                             % 12 subcarriers per RB
subcarrierSpacing = 15;

trainNow = false;

miniBatchSize = 1000;
options = trainingOptions("adam", ...
    InitialLearnRate=0.0074, ...
    LearnRateSchedule="piecewise", ...
    LearnRateDropPeriod=112, ...
    LearnRateDropFactor=0.6085, ...
    Epsilon=1e-7, ...
    MaxEpochs=1000, ...
    MiniBatchSize=miniBatchSize, ...
    Shuffle="every-epoch", ...
    ValidationData={HVReal,HVReal}, ...
    ValidationFrequency=20, ...
    Verbose=false, ...
    ValidationPatience=20, ...
    OutputNetwork="best-validation-loss", ...
    ExecutionEnvironment="auto", ...
    Plots='training-progress') %#ok<NASGU>

options =
  TrainingOptionsADAM with properties:

             GradientDecayFactor: 0.9000
         SquaredGradientDecayFactor: 0.9990
                            Epsilon: 1.0000e-07
                  InitialLearnRate: 0.0074
                 LearnRateSchedule: 'piecewise'
                LearnRateDropFactor: 0.6085
                LearnRateDropPeriod: 112
                   L2Regularization: 1.0000e-04
           GradientThresholdMethod: 'l2norm'
                 GradientThreshold: Inf
                         MaxEpochs: 1000
                     MiniBatchSize: 1000
                           Verbose: 0
                   VerboseFrequency: 50
                     ValidationData: {[28×8×2×3000 single]  [28×8×2×3000 single]}
                ValidationFrequency: 20
                 ValidationPatience: 20
                            Shuffle: 'every-epoch'
```

```
                CheckpointPath: ''
            CheckpointFrequency: 1
       CheckpointFrequencyUnit: 'epoch'
          ExecutionEnvironment: 'auto'
                    WorkerLoad: []
                     OutputFcn: []
                         Plots: 'training-progress'
                SequenceLength: 'longest'
          SequencePaddingValue: 0
      SequencePaddingDirection: 'right'
          DispatchInBackground: 0
       ResetInputNormalization: 1
  BatchNormalizationStatistics: 'population'
                OutputNetwork: 'best-validation-loss'
```

```
if trainNow
  [net,trainInfo] = ...
    trainNetwork(HTReal,HTReal,autoencoderLGraph,options); %#ok<UNRCH>
  save("csiTrainedNetwork_" ...
    + string(datetime("now","Format","dd_MM_HH_mm")), ...
    'net','trainInfo','options','autoEncOpt')
else
  helperCSINetDownloadData()
  autoEncOptCached = autoEncOpt;
  load("csiTrainedNetwork",'net','trainInfo','options','autoEncOpt')
  if autoEncOpt.NumSubcarriers ~= autoEncOptCached.NumSubcarriers ...
      || autoEncOpt.NumSymbols ~= autoEncOptCached.NumSymbols ...
      || autoEncOpt.NumTxAntennas ~= autoEncOptCached.NumTxAntennas ...
      || autoEncOpt.NumRxAntennas ~= autoEncOptCached.NumRxAntennas ...
      || autoEncOpt.MaxDelay ~= autoEncOptCached.MaxDelay
    error("CSIExample:Missmatch", ...
      "Saved network does not match settings. Set trainNow to true.")
  end
end
```

```
Files already exist. Skipping download and extract.
```

Test Trained Network

Use the `predict` (Deep Learning Toolbox) function to process the test data.

```
HTestRealHat = predict(net,HTestReal);
```

Calculate the correlation and normalized mean squared error (NMSE) between the input and output of the autoencoder network. The correlation is defined as

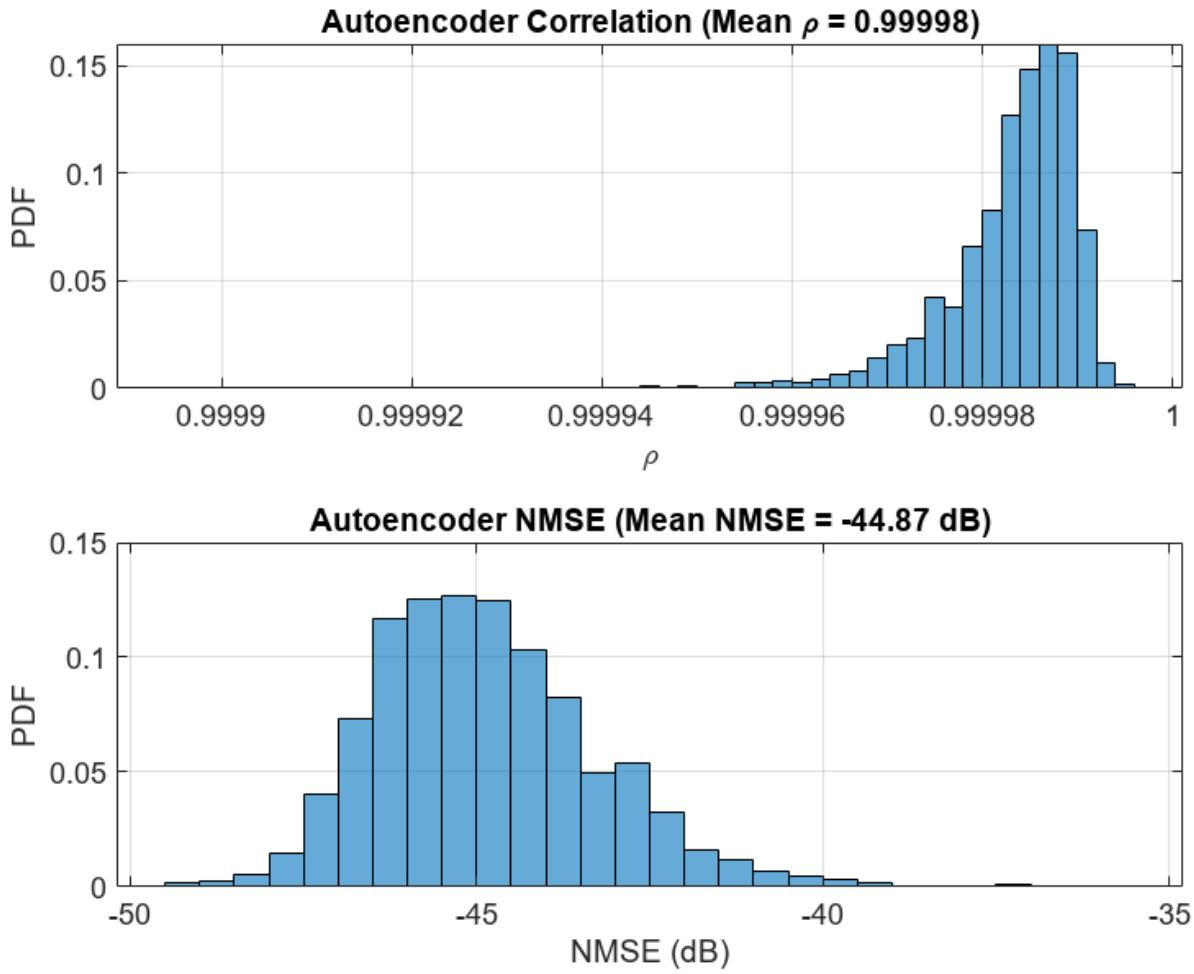
$$\rho = \mathbb{E}\left\{\frac{1}{N}\sum_{n=1}^{N}\frac{|\hat{h}_n^H h_n|}{\|\hat{h}_n\|_2\|h_n\|_2}\right\}$$

where h_n is the channel estimate at the input of the autoencoder and $\hat{h}_n$ is the channel estimate at the output of the autoencoder. NMSE is defined as

$$NMSE = \mathbb{E}\left\{\frac{\|H-\hat{H}\|_2^2}{\|H\|_2^2}\right\}$$

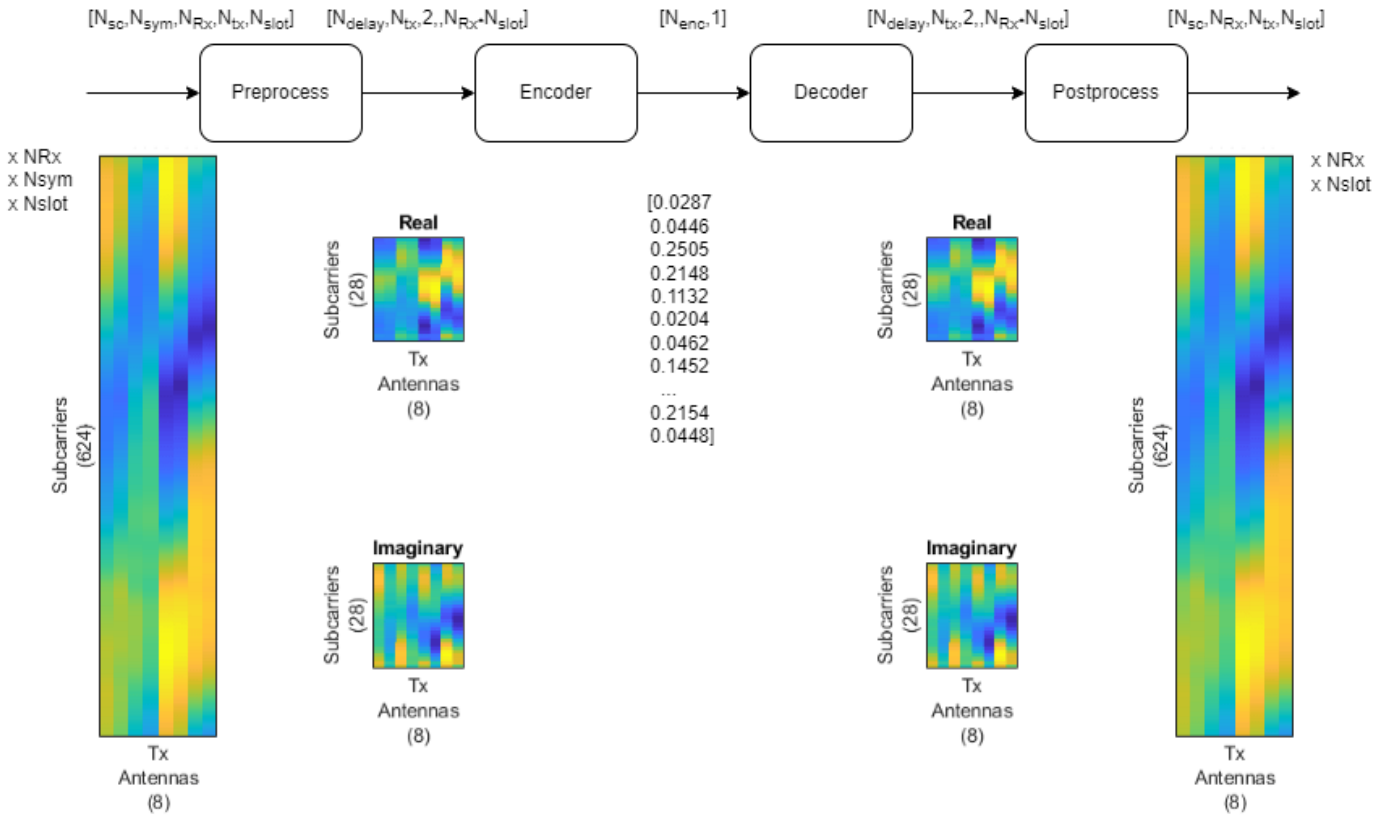
where H is the channel estimate at the input of the autoencoder and $\widehat{H}$ is the channel estimate at the output of the autoencoder.

```
rho = zeros(numTest,1);
nmse = zeros(numTest,1);
for n=1:numTest
    in = HTestReal(:,:,1,n) + 1i*(HTestReal(:,:,2,n));
    out = HTestRealHat(:,:,1,n) + 1i*(HTestRealHat(:,:,2,n));

    % Calculate correlation
    n1 = sqrt(sum(conj(in).*in,'all'));
    n2 = sqrt(sum(conj(out).*out,'all'));
    aa = abs(sum(conj(in).*out,'all'));
    rho(n) = aa / (n1*n2);

    % Calculate NMSE
    mse = mean(abs(in-out).^2,'all');
    nmse(n) = 10*log10(mse / mean(abs(in).^2,'all'));
end

figure
tiledlayout(2,1)
nexttile
histogram(rho,"Normalization","probability")
grid on
title(sprintf("Autoencoder Correlation (Mean \\rho = %1.5f)", ...
  mean(rho)))
xlabel("\rho"); ylabel("PDF")
nexttile
histogram(nmse,"Normalization","probability")
grid on
title(sprintf("Autoencoder NMSE (Mean NMSE = %1.2f dB)",mean(nmse)))
xlabel("NMSE (dB)"); ylabel("PDF")
```



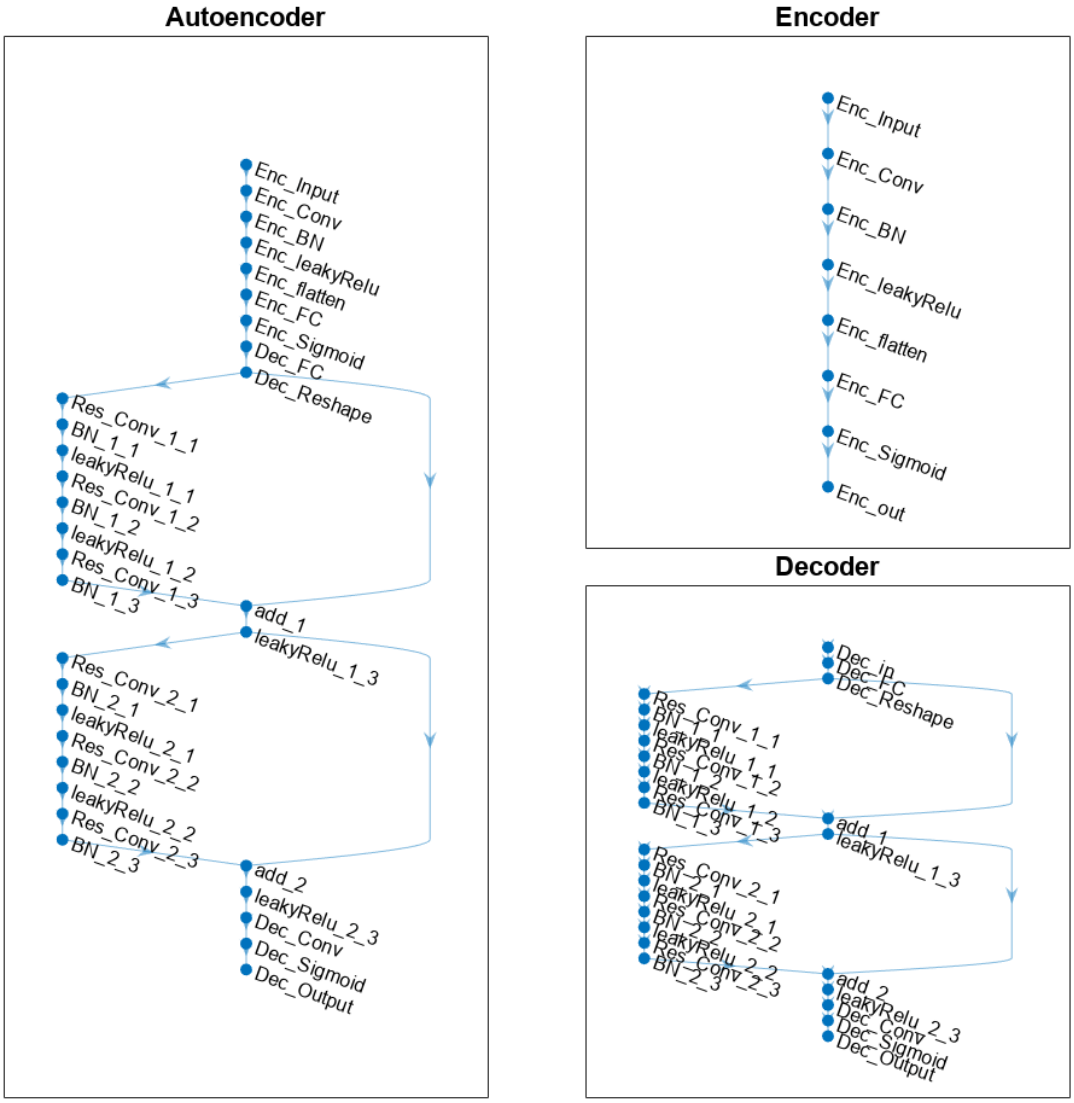
End-to-End CSI Feedback System

This figure shows the end-to-end processing of channel estimates for CSI feedback. The UE uses the CSI-RS signal to estimate the channel response for one slot, H_{est}. The preprocessed channel estimate, H_{tr}, is encoded by using the encoder portion of the autoencoder to produce a 1-by-N_{enc} compressed array. The compressed array is decompressed by the decoder portion of the autoencoder to obtain $\widehat{H_{tr}}$. Postprocessing $\widehat{H_{tr}}$ produces $\widehat{H_{est}}$.



To obtain the encoded array, split the autoencoder into two parts: the encoder network and the decoder network.

```
[encNet,decNet] = helperCSINetSplitEncoderDecoder(net,"Enc_Sigmoid");
plotNetwork(net,encNet,decNet)
```



Generate channel estimates.

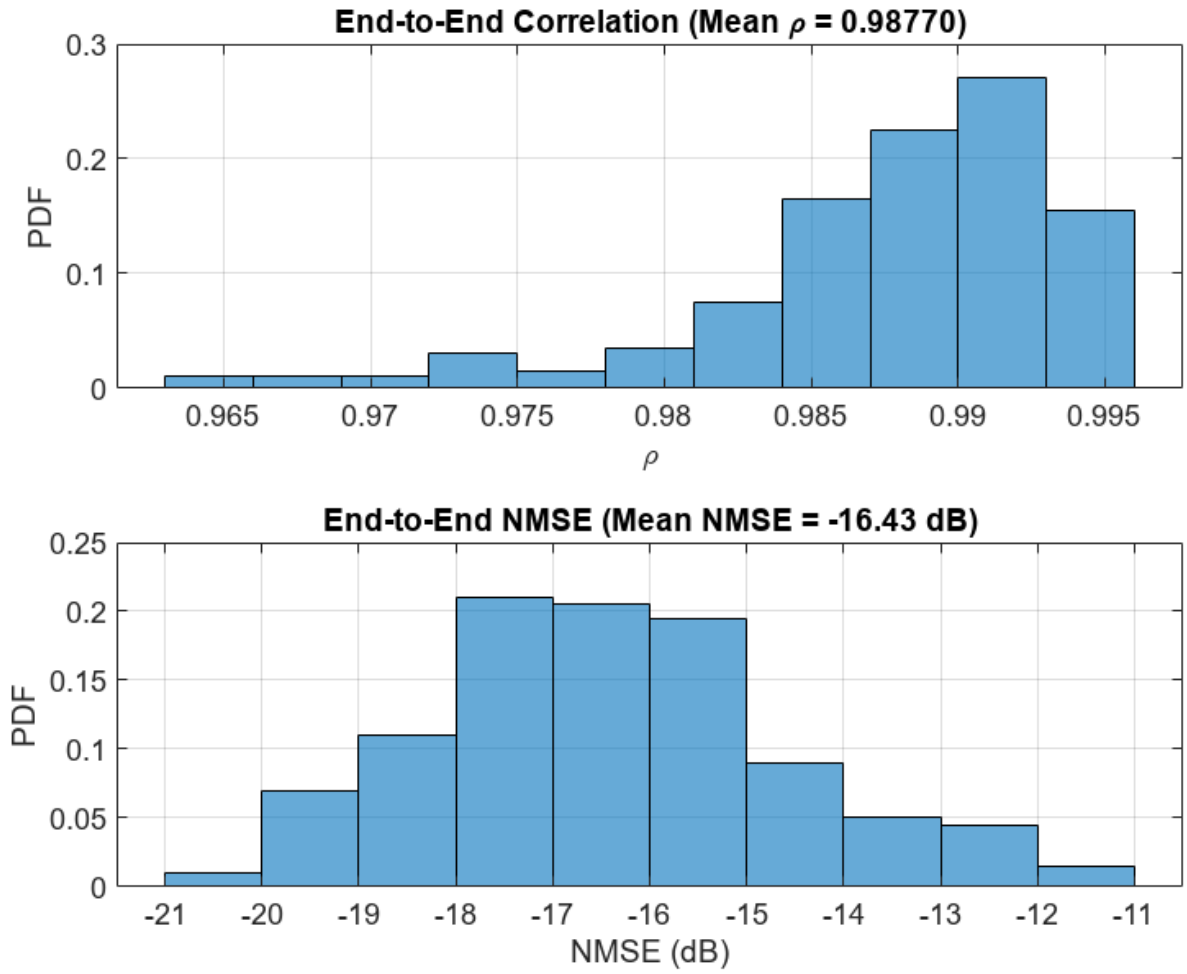
```
nSlots = 100;
Hest = helperCSINetChannelEstimate(nSlots,carrier,channel);
```

Encode and decode the channel estimates with `Normalization` set to `true`.

```
autoEncOpt.Normalization = true;
codeword = helperCSINetEncode(encNet, Hest, autoEncOpt);
Hhat = helperCSINetDecode(decNet, codeword, autoEncOpt);
```

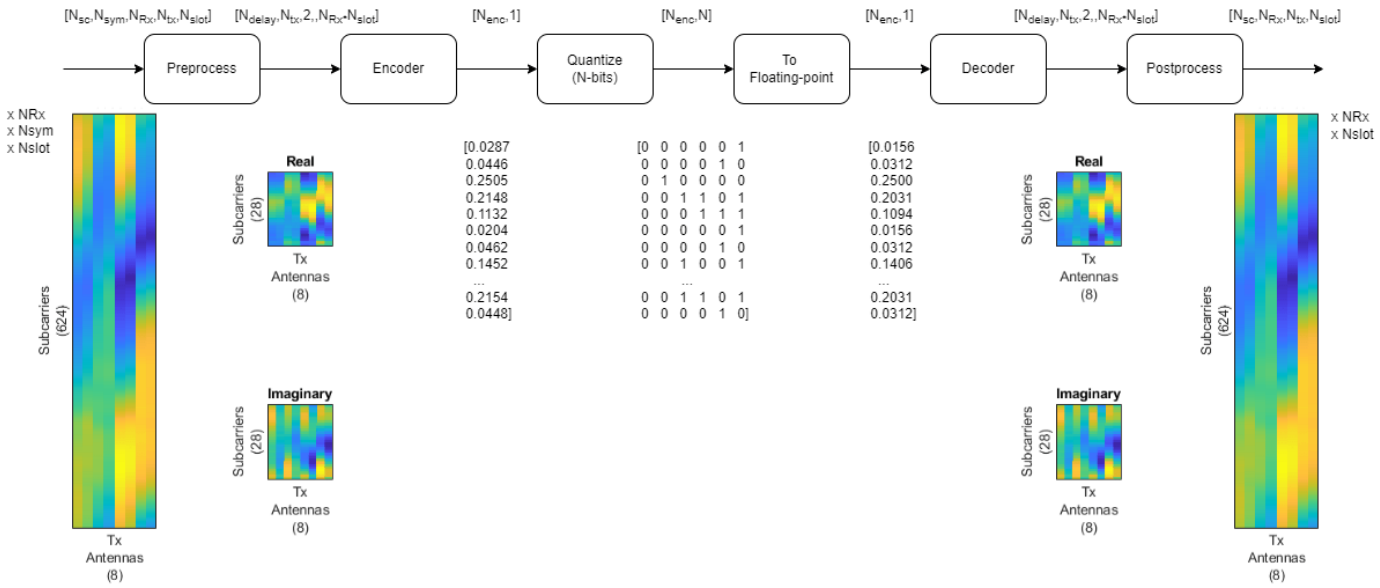
Calculate the correlation and NMSE for the end-to-end CSI feedback system.

```
H = squeeze(mean(Hest,2));
rhoE2E = zeros(nRx,nSlots);
nmseE2E = zeros(nRx,nSlots);
for rx=1:nRx
    for n=1:nSlots
        out = Hhat(:,rx,:,n);
        in = H(:,rx,:,n);
        rhoE2E(rx,n) = helperCSINetCorrelation(in,out);
        nmseE2E(rx,n) = helperNMSE(in,out);
    end
end
figure
tiledlayout(2,1)
nexttile
histogram(rhoE2E,"Normalization","probability")
grid on
title(sprintf("End-to-End Correlation (Mean \\rho = %1.5f)", ...
  mean(rhoE2E,'all')))
xlabel("\rho"); ylabel("PDF")
nexttile
histogram(nmseE2E,"Normalization","probability")
grid on
title(sprintf("End-to-End NMSE (Mean NMSE = %1.2f dB)", ...
  mean(nmseE2E,'all')))
xlabel("NMSE (dB)"); ylabel("PDF")
```



Effect of Quantized Codewords

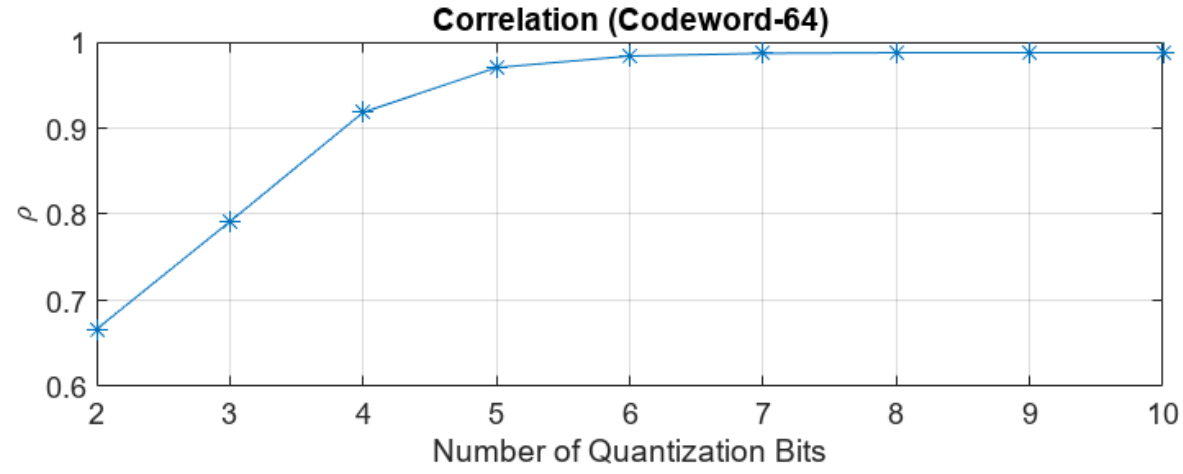
Practical systems require quantizing the encoded codeword by using a small number of bits. Simulate the effect of quantization across the range of [2, 10] bits. The results show that 6-bits is enough to closely approximate the single-precision performance.

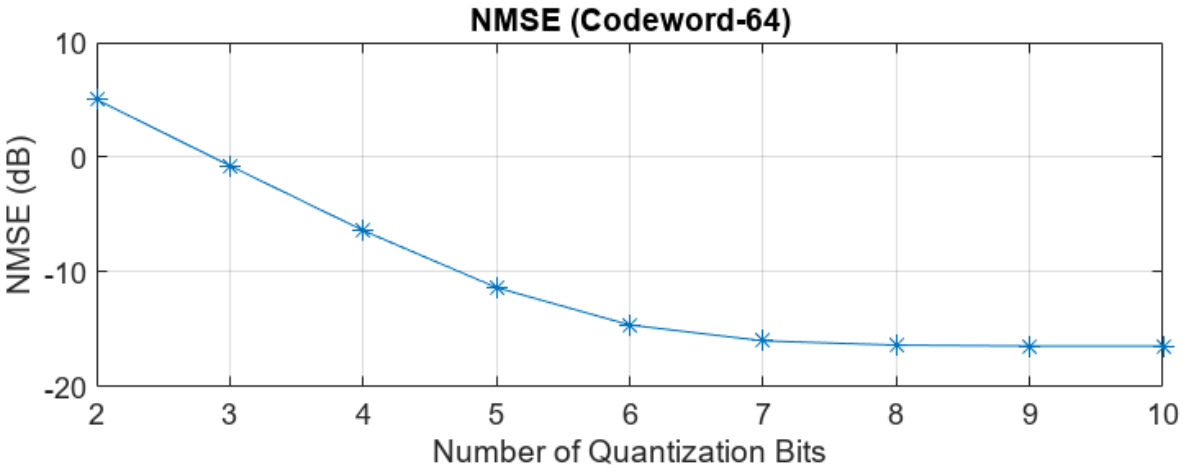



```
maxVal = 1;
minVal = -1;
idxBits = 1;
nBitsVec = 2:10;
rhoQ = zeros(nRx,nSlots,length(nBitsVec));
nmseQ = zeros(nRx,nSlots,length(nBitsVec));
for numBits = nBitsVec
    disp("Running for " + numBits + " bit quantization")

    % Quantize between 0:2^n-1 to get bits
    qCodeword = uencode(double(codeword*2-1), numBits);

    % Get back the floating point, quantized numbers
    codewordRx = (single(udecode(qCodeword,numBits))+1)/2;
    Hhat = helperCSINetDecode(decNet, codewordRx, autoEncOpt);
    H = squeeze(mean(Hest,2));
    for rx=1:nRx
        for n=1:nSlots
            out = Hhat(:,rx,:,n);
            in = H(:,rx,:,n);
            rhoQ(rx,n,idxBits) = helperCSINetCorrelation(in,out);
            nmseQ(rx,n,idxBits) = helperNMSE(in,out);
        end
    end
    idxBits = idxBits + 1;
end
```

```
Running for 2 bit quantization
Running for 3 bit quantization
Running for 4 bit quantization
Running for 5 bit quantization
Running for 6 bit quantization
Running for 7 bit quantization
Running for 8 bit quantization
```

```
Running for 9 bit quantization
Running for 10 bit quantization

figure
tiledlayout(2,1)
nexttile
plot(nBitsVec,squeeze(mean(rhoQ,[1 2])),'*-')
title("Correlation (Codeword-" + size(codeword,3) + ")")
xlabel("Number of Quantization Bits"); ylabel("\rho")
grid on
nexttile
plot(nBitsVec,squeeze(mean(nmseQ,[1 2])),'*-')
title("NMSE (Codeword-" + size(codeword,3) + ")")
xlabel("Number of Quantization Bits"); ylabel("NMSE (dB)")
grid on
```





Further Exploration

The autoencoder is able to compress a [624 8] single-precision complex channel estimate array into a [64 1] single-precision array with a mean correlation factor of 0.99 and an NMSE of –16 dB. Using 6-

bit quantization requires only 384 bits of CSI feedback data, which equates to a compression ratio of approximately 800:1.

```
display("Compression ratio is " + (624*8*32*2)/(64*6) + ":" + 1)
```

```
    "Compression ratio is 832:1"
```

Investigate the effect of `truncationFactor` on the system performance. Vary the 5G system parameters, channel parameters, and number of encoded symbols and then find the optimum values for the defined channel.

The “NR PDSCH Throughput Using Channel State Information Feedback” on page 1-187 example shows how to use channel state information (CSI) feedback to adjust the physical downlink shared channel (PDSCH) parameters and measure throughput. Replace the CSI feedback algorithm with the CSI compression autoencoder and compare performance.

Helper Functions

Explore the helper functions to see the detailed implementation of the system.

Training Data Generation

helperCSINetChannelEstimate

helperCSINetTrainingData

Network Definition and Manipulation

helperCSINetLayerGraph

helperCSINetAddResidualLayers

helperCSINetSplitEncoderDecoder

CSI Processing

helperCSINetPreprocessChannelEstimate

helperCSINetPostprocessChannelEstimate

helperCSINetEncode

helperCSINetDecode

Performance Measurement

helperCSINetCorrelation

helperNMSE

Appendix: Optimize Hyperparameters with Experiment Manager

Use the Experiment Manager app to find the optimal parameters. `CSITrainingProject.mlproj` is a preconfigured project. Extract the project.

```
if ~exist("CSITrainingProject","dir")
  projRoot = helperCSINetExtractProject();
```

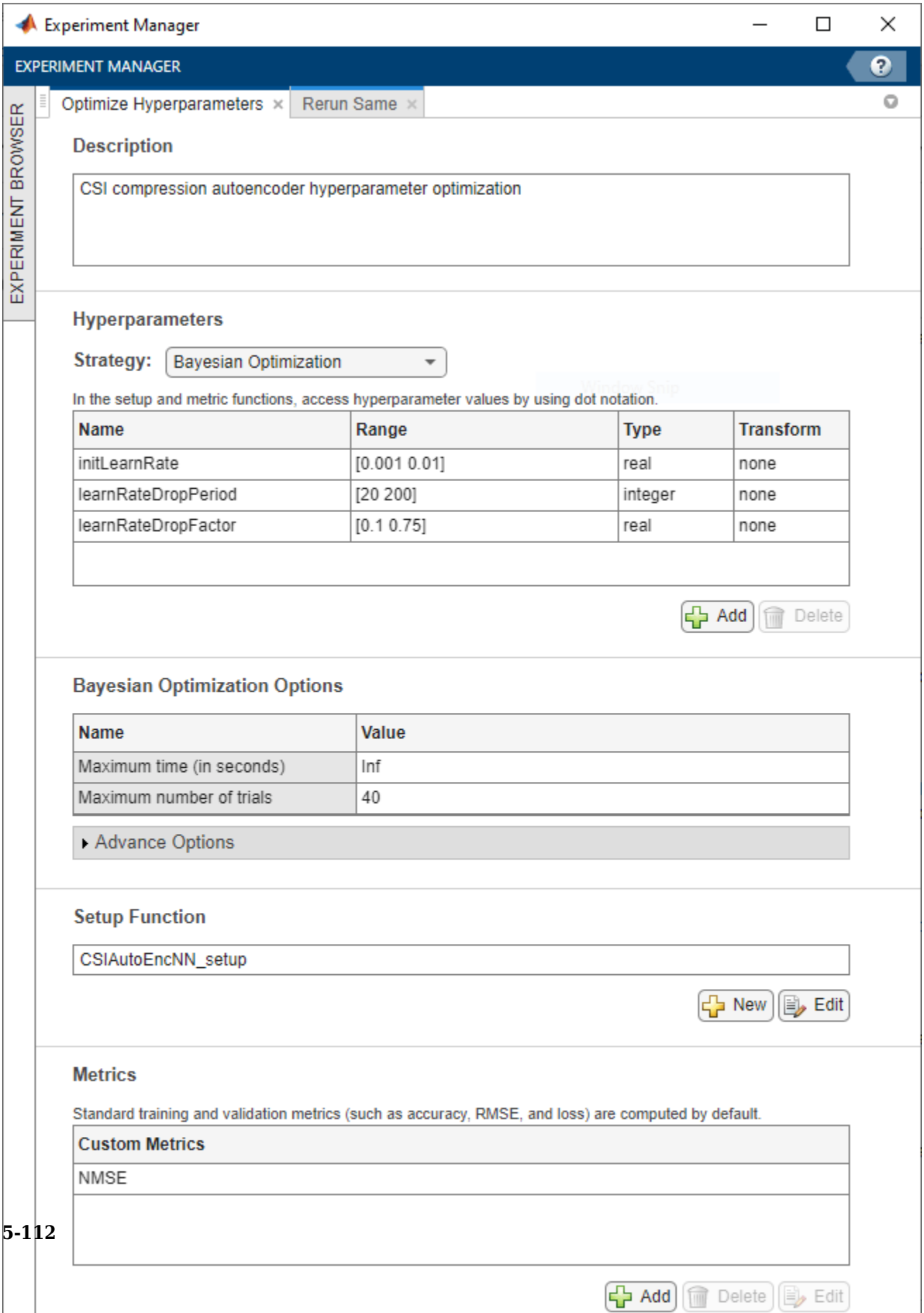
```
else
  projRoot = fullfile(exRoot(),"CSITrainingProject");
end
```

To open the project, start the Experiment Manager app and open the following file.

```
disp(fullfile(".","CSITrainingProject","CSITrainingProject.prj"))
```

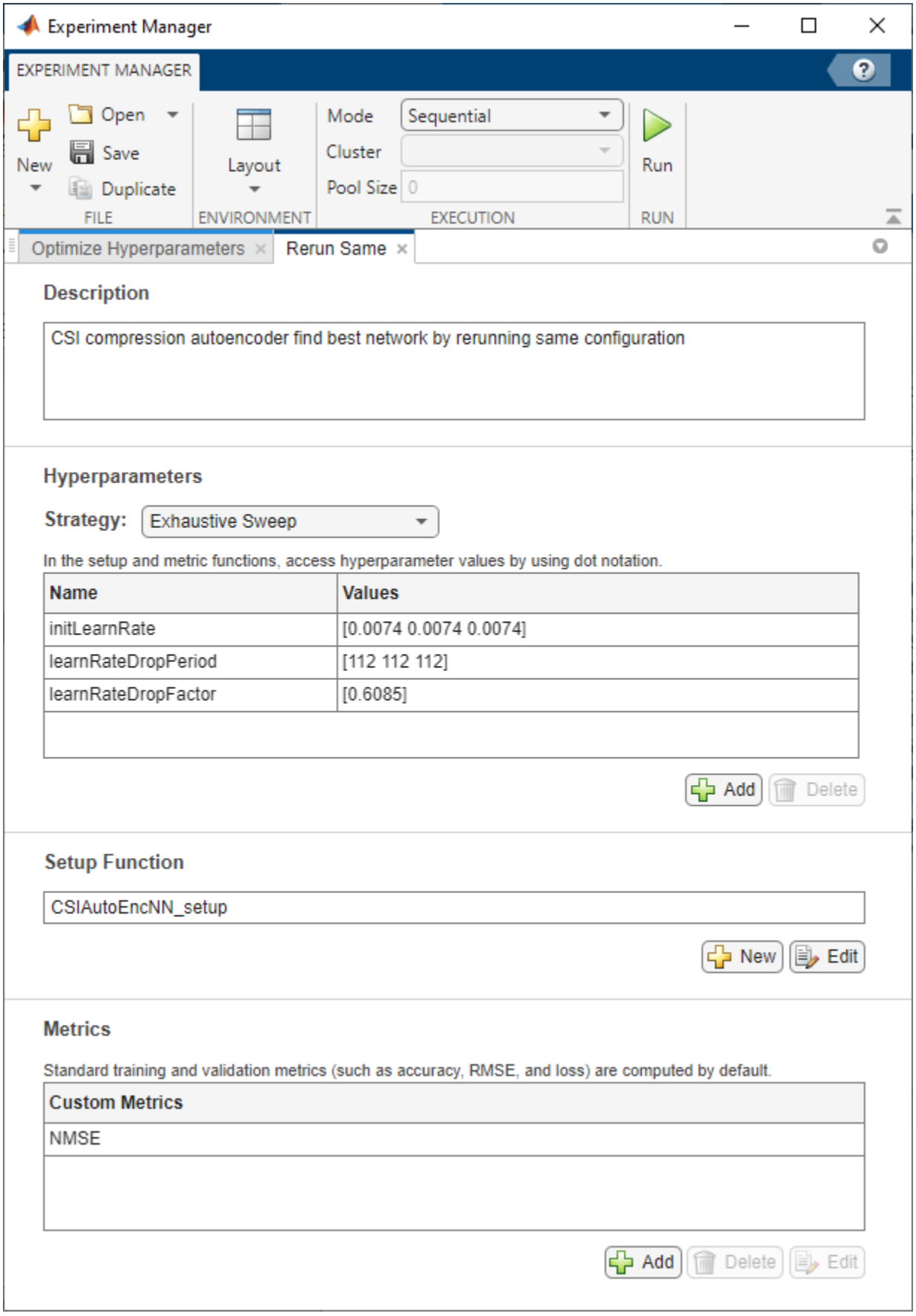
```
.\CSITrainingProject\CSITrainingProject.prj
```

The Optimize Hyperparameters experiment uses Bayesian optimization with hyperparameter search ranges specified as in the following figure. The experiment setup function is `CSIAutoEncNN_setup`. The custom metric function is `NMSE`.


Experiment Manager
EXPERIMENT MANAGER
EXPERIMENT BROWSER
Optimize Hyperparameters
Rerun Same
Description
CSI compression autoencoder hyperparameter optimization
Hyperparameters
Strategy:
Bayesian Optimization
In the setup and metric functions, access hyperparameter values by using dot notation.
Name
Range
Type
Transform
initLearnRate
[0.001 0.01]
real
none
learnRateDropPeriod
[20 200]
integer
none
learnRateDropFactor
[0.1 0.75]
real
none
Add
Delete
Bayesian Optimization Options
Name
Value
Maximum time (in seconds)
Inf
Maximum number of trials
40
Advance Options
Setup Function
CSIAutoEncNN_setup
New
Edit
Metrics
Standard training and validation metrics (such as accuracy, RMSE, and loss) are computed by default.
Custom Metrics
NMSE
Add
Delete
Edit

Trial	Status	Actions	Progress	Elapsed Time	initLearnRate	learnRateDropPeriod	learnRateDropFactor	Training RMSE	Training Loss	Validation RMSE	Validation Loss	NMSE
1	Complete ...		57.6%	0 hr 9 min 31 sec	0.0060	125.0000	0.5918	0.0792	0.0033	0.0798	0.0032	-14.1704
2	Complete ...		49.1%	0 hr 7 min 25 sec	0.0021	23.0000	0.5205	0.3231	0.0532	0.3221	0.0519	-2.6815
3	Complete ...		81.5%	0 hr 12 min 1 sec	0.0089	155.0000	0.1519	0.1217	0.0077	0.1217	0.0074	-11.0679
4	Complete ...		83.7%	0 hr 12 min 24 sec	0.0025	123.0000	0.1545	0.1628	0.0135	0.1625	0.0132	-8.3862
5	Complete ...		77.5%	0 hr 11 min 27 sec	0.0074	112.0000	0.6085	0.0626	0.0020	0.0641	0.0021	-15.9670
6	Complete ...		68.1%	0 hr 10 min 5 sec	0.0096	103.0000	0.1121	0.1642	0.0127	0.1586	0.0126	-8.7100
7	Complete ...		100.0%	0 hr 14 min 54 sec	0.0071	181.0000	0.4058	0.0663	0.0023	0.0679	0.0023	-15.5089
8	Complete ...		100.0%	0 hr 14 min 40 sec	0.0085	35.0000	0.7490	0.1318	0.0090	0.1322	0.0087	-10.3420
9	Complete ...		100.0%	0 hr 14 min 42 sec	0.0070	135.0000	0.3154	0.0737	0.0028	0.0738	0.0027	-14.8468
10	Complete ...		47.7%	0 hr 7 min 12 sec	0.0073	161.0000	0.7499	0.0880	0.0039	0.0788	0.0031	-14.2949
11	Complete ...		100.0%	0 hr 14 min 41 sec	0.0070	113.0000	0.5636	0.1074	0.0057	0.1066	0.0057	-12.0698
12	Complete ...		55.3%	0 hr 8 min 16 sec	0.0063	200.0000	0.4569	0.0751	0.0028	0.0699	0.0024	-15.2509
13	Complete ...		38.7%	0 hr 5 min 50 sec	0.0063	200.0000	0.2267	0.1303	0.0088	0.1331	0.0089	-10.0289
14	Complete ...		44.9%	0 hr 6 min 51 sec	0.0072	200.0000	0.5592	0.0957	0.0048	0.1027	0.0053	-11.9141
15	Complete ...		100.0%	0 hr 14 min 36 sec	0.0010	200.0000	0.6932	0.1076	0.0060	0.1083	0.0059	-11.7117
16	Complete ...		69.4%	0 hr 10 min 17 sec	0.0040	200.0000	0.7347	0.0691	0.0024	0.0681	0.0023	-15.5948
17	Complete ...		45.9%	0 hr 6 min 50 sec	0.0055	172.0000	0.7474	0.0899	0.0038	0.0898	0.0040	-13.2175
18	Complete ...		73.3%	0 hr 10 min 48 sec	0.0100	108.0000	0.7442	0.0764	0.0027	0.0722	0.0026	-15.0913
19	Complete ...		25.1%	0 hr 3 min 54 sec	0.0100	147.0000	0.6654	0.1427	0.0102	0.1342	0.0090	-9.9184
20	Complete ...		43.5%	0 hr 6 min 30 sec	0.0062	159.0000	0.7163	0.0792	0.0032	0.0842	0.0035	-13.5991
21	Complete ...		100.0%	0 hr 14 min 36 sec	0.0010	112.0000	0.7493	0.1170	0.0066	0.1138	0.0065	-11.2892
22	Complete ...		17.5%	0 hr 2 min 46 sec	0.0099	20.0000	0.1024	0.4519	0.1035	0.4439	0.0985	0.0593
23	Complete ...		60.7%	0 hr 9 min 5 sec	0.0053	160.0000	0.5462	0.0759	0.0029	0.0739	0.0027	-14.8119

The optimal parameters are 0.0074 for initial learning rate, 112 iterations for the learning rate drop period, and 0.6085 for learning rate drop factor. After finding the optimal hyperparameters, train the network with same parameters multiple times to find the best trained network. Increase the maximum iterations by a factor of two.


Experiment Manager
EXPERIMENT MANAGER
New
Open
Save
Duplicate
FILE
Layout
ENVIRONMENT
Mode
Sequential
Cluster
Pool Size
0
EXECUTION
Run
RUN
Optimize Hyperparameters
Rerun Same
Description
CSI compression autoencoder find best network by rerunning same configuration
Hyperparameters
Strategy:
Exhaustive Sweep
In the setup and metric functions, access hyperparameter values by using dot notation.
Name
Values
initLearnRate
[0.0074 0.0074 0.0074]
learnRateDropPeriod
[112 112 112]
learnRateDropFactor
[0.6085]
Add
Delete
Setup Function
CSIAutoEncNN_setup
New
Edit
Metrics
Standard training and validation metrics (such as accuracy, RMSE, and loss) are computed by default.
Custom Metrics
NMSE
Add
Delete
Edit

The sixth trial produced the best NMSE. This example uses this trained network as the saved network.

Trial	Status	Actions	Progress	Elapsed Time	initLearnRate	learnRateDro...	learnRateDro...	Training RMSE	Training Loss	Validation RM...	Validation Loss	NMSE
1	Complete (...		87.4%	0 hr 10 min 40 sec	0.0074	112.0000	0.6085	0.0672	0.0023	0.0656	0.0021	-16.2248
2	Complete (...		87.4%	0 hr 9 min 19 sec	0.0074	112.0000	0.6085	0.0660	0.0022	0.0649	0.0021	-16.2760
3	Complete (...		87.4%	0 hr 9 min 34 sec	0.0074	112.0000	0.6085	0.0655	0.0021	0.0641	0.0021	-16.3752
4	Complete (...		87.4%	0 hr 9 min 6 sec	0.0074	112.0000	0.6085	0.0659	0.0022	0.0649	0.0021	-16.2593
5	Complete (...		87.4%	0 hr 11 min 15 sec	0.0074	112.0000	0.6085	0.0663	0.0022	0.0651	0.0021	-16.2432
6	Complete (...		87.4%	0 hr 11 min 40 sec	0.0074	112.0000	0.6085	0.0650	0.0021	0.0636	0.0020	-16.4327
7	Complete (...		87.4%	0 hr 11 min 36 sec	0.0074	112.0000	0.6085	0.0676	0.0023	0.0664	0.0022	-16.0746
8	Complete (...		63.4%	0 hr 7 min 8 sec	0.0074	112.0000	0.6085	0.0788	0.0031	0.0771	0.0030	-14.7916
9	Complete (...		87.4%	0 hr 8 min 54 sec	0.0074	112.0000	0.6085	0.0651	0.0021	0.0642	0.0021	-16.3581

Configuring Batch Mode

When execution **Mode** is set to `Batch Sequential` or `Batch Simultaneous`, training data must be accessible to the workers in a location defined by the `dataDir` variable in the Prepare Data in Bulk section. Set `dataDir` to a network location that is accessible by the workers. For more information, see “Offload Experiments as Batch Jobs to Cluster” (Deep Learning Toolbox).

Local Functions

```
function plotChannelResponse(Hest)
%plotChannelResponse Plot channel response

figure
tiledlayout(2,2)
nexttile
waterfall(abs(Hest(:,:,1,1))')
xlabel("Subcarriers");
ylabel("Symbols");
zlabel("Channel Magnitude")
view(15,30)
colormap("cool")
title("Rx=1, Tx=1")
nexttile
plot(squeeze(abs(Hest(:,1,:,1))))
grid on
xlabel("Subcarriers");
ylabel("Channel Magnitude")
legend("Rx 1", "Rx 2")
title("Symbol=1, Tx=1")
nexttile
waterfall(squeeze(abs(Hest(:,1,1,:)))')
view(-45,75)
grid on
xlabel("Subcarriers");
ylabel("Tx");
zlabel("Channel Magnitude")
title("Symbol=1, Rx=1")
nexttile
nSubCarriers = size(Hest,1);
subCarrier = randi(nSubCarriers);
plot(squeeze(abs(Hest(subCarrier,1,:,:)))')
grid on
xlabel("Tx");
ylabel("Channel Magnitude")
legend("Rx 1", "Rx 2")
title("Subcarrier=" + subCarrier + ", Symbol=1")
```

```
end

function valid = validateTrainingFiles(dataDir,filePrefix,expN, ...
  opt,channel,carrier)
%validateTrainingFiles Validate training data files
%   V = validateTrainingFiles(DIR,PRE,N,OPT,CH,CR) checks the DIR directory
%   for training data files with a prefix of PRE. It checks if there are
%   N*OPT.NumRxAntennas files, channel configuration is same as CH, and
%   carrier configuration is same as CR.

valid = true;
files = dir(fullfile(dataDir,filePrefix+"*"));
if isempty(files)
  valid = false;
  return
end
if exist(fullfile(dataDir,"info.mat"),"file")
  infoStr = load(fullfile(dataDir,"info.mat"));
  if ~isequal(get(infoStr.channel),get(channel)) ...
      || ~isequal(infoStr.carrier,carrier)
    valid = false;
  end
else
  valid = false;
end
if valid
  valid = (expN == (length(files)*opt.NumRxAntennas));
  % Check size of Hest in the files
  load(fullfile(files(1).folder,files(1).name),'H')
  if ~isequal(size(H),[opt.NumSubcarriers opt.NumSymbols ...
      opt.NumRxAntennas opt.NumTxAntennas])
    valid = false;
  end
end
if ~valid
  disp("Removing invalid data directory: " + files(1).folder)
  rmdir(files(1).folder,'s')
end
end

function plotNetwork(net,encNet,decNet)
%plotNetwork Plot autoencoder network
%   plotNetwork(NET,ENC,DEC) plots the full autoencoder network together
%   with encoder and decoder networks.
fig = figure;
t1 = tiledlayout(1,2,'TileSpacing','Compact');
t2 = tiledlayout(t1,1,1,'TileSpacing','Tight');
t3 = tiledlayout(t1,2,1,'TileSpacing','Tight');
t3.Layout.Tile = 2;
nexttile(t2)
plot(net)
title("Autoencoder")
nexttile(t3)
plot(encNet)
title("Encoder")
nexttile(t3)
plot(decNet)
title("Decoder")
```

```
pos = fig.Position;
pos(3) = pos(3) + 200;
pos(4) = pos(4) + 300;
pos(2) = pos(2) - 300;
fig.Position = pos;
end

function plotPreprocessingSteps(Hmean,Hdft2,Htemp,Htrunc, ...
  nSub,nTx,maxDelay)
%plotPreprocessingSteps Plot preprocessing workflow

hfig = figure;
hfig.Position(3) = hfig.Position(3)*2;
subplot(2,5,[1 6])
himg = imagesc(abs(Hmean));
himg.Parent.YDir = "normal";
himg.Parent.Position(3) = 0.05;
himg.Parent.XTick=''; himg.Parent.YTick='';
xlabel(sprintf('Tx\nAntennas\n(%d)',nTx));
ylabel(sprintf('Subcarriers\n(%d)',nSub'));
title("Measured")
subplot(2,5,[2 7])
himg = image(abs(Hdft2));
himg.Parent.YDir = "normal";
himg.Parent.Position(3) = 0.05;
himg.Parent.XTick=''; himg.Parent.YTick='';
title("2-D DFT")
xlabel(sprintf('Tx\nAngle\n(%d)',nTx));
ylabel(sprintf('Delay Samples\n(%d)',nSub'));
subplot(2,5,[3 8])
himg = image(abs(Htemp));
himg.Parent.YDir = "normal";
himg.Parent.Position(3) = 0.05;
himg.Parent.Position(4) = himg.Parent.Position(4)*10*maxDelay/nSub;
himg.Parent.Position(2) = (1 - himg.Parent.Position(4)) / 2;
himg.Parent.XTick=''; himg.Parent.YTick='';
xlabel(sprintf('Tx\nAngle\n(%d)',nTx));
ylabel(sprintf('Delay Samples\n(%d)',maxDelay'));
title("Truncated")
subplot(2,5,[4 9])
himg = imagesc(abs(Htrunc));
himg.Parent.YDir = "normal";
himg.Parent.Position(3) = 0.05;
himg.Parent.Position(4) = himg.Parent.Position(4)*10*maxDelay/nSub;
himg.Parent.Position(2) = (1 - himg.Parent.Position(4)) / 2;
himg.Parent.XTick=''; himg.Parent.YTick='';
xlabel(sprintf('Tx\nAntennas\n(%d)',nTx));
ylabel(sprintf('Subcarriers\n(%d)',maxDelay'));
title("2-D IDFT")
subplot(2,5,5)
himg = imagesc(real(Htrunc));
himg.Parent.YDir = "normal";
himg.Parent.Position(3) = 0.05;
himg.Parent.Position(4) = himg.Parent.Position(4)*10*maxDelay/nSub;
himg.Parent.Position(2) = himg.Parent.Position(2) + 0.18;
himg.Parent.XTick=''; himg.Parent.YTick='';
xlabel(sprintf('Tx\nAntennas\n(%d)',nTx));
ylabel(sprintf('Subcarriers\n(%d)',maxDelay'));
```

```
title("Real")
subplot(2,5,10)
himg = imagesc(imag(Htrunc));
himg.Parent.YDir = "normal";
himg.Parent.Position(3) = 0.05;
himg.Parent.Position(4) = himg.Parent.Position(4)*10*maxDelay/nSub;
himg.Parent.Position(2) = himg.Parent.Position(2) + 0.18;
himg.Parent.XTick=''; himg.Parent.YTick='';
xlabel(sprintf('Tx\nAntennas\n(%d)',nTx));
ylabel(sprintf('Subcarriers\n(%d)',maxDelay'));
title("Imaginary")
end

function rootDir = exRoot()
%exRoot Example root directory
rootDir = fileparts(which("helperCSINetLayerGraph"));
end
```

References

[1] Wen, Chao-Kai, Wan-Ting Shih, and Shi Jin. “Deep Learning for Massive MIMO CSI Feedback.” IEEE Wireless Communications Letters 7, no. 5 (October 2018): 748–51. https://doi.org/10.1109/LWC.2018.2818160.

[2] Zimaglia, Elisa, Daniel G. Riviello, Roberto Garello, and Roberto Fantini. “A Novel Deep Learning Approach to CSI Feedback Reporting for NR 5G Cellular Systems.” In 2020 IEEE Microwave Theory and Techniques in Wireless Communications (MTTW), 47–52. Riga, Latvia: IEEE, 2020. https://doi.org/10.1109/MTTW51045.2020.9245055.

See Also

Related Examples

- “NR PDSCH Throughput Using Channel State Information Feedback” on page 1-187
- “5G NR Downlink CSI Reporting” on page 5-47

6

System-Level Simulation

Simulation Visualizations

System-level simulations model multinode networks with nodes modeling protocol stack that includes physical (PHY), medium access control (MAC), and radio link control (RLC) layers. You can model NR cells using the 5G Toolbox™ and evaluate the network performance with different MAC scheduling strategies and PHY algorithms. You can also measure metrics like throughput, scheduling fairness, block error rate (BLER), and spectrum efficiency, and use visualization capabilities to analyze the performance of the network.

This topic page explains the visualizations with respect to a sample 5G NR cell consisting of a set of user equipments (UEs) connected to a gNB. These are some of the elements you can customize in a simulation.

- Configuration parameters for each layer
- Scheduling strategies
- PHY layer — Passthrough PHY layer (for faster MAC focused simulations) or real PHY layer
- Channel impairments

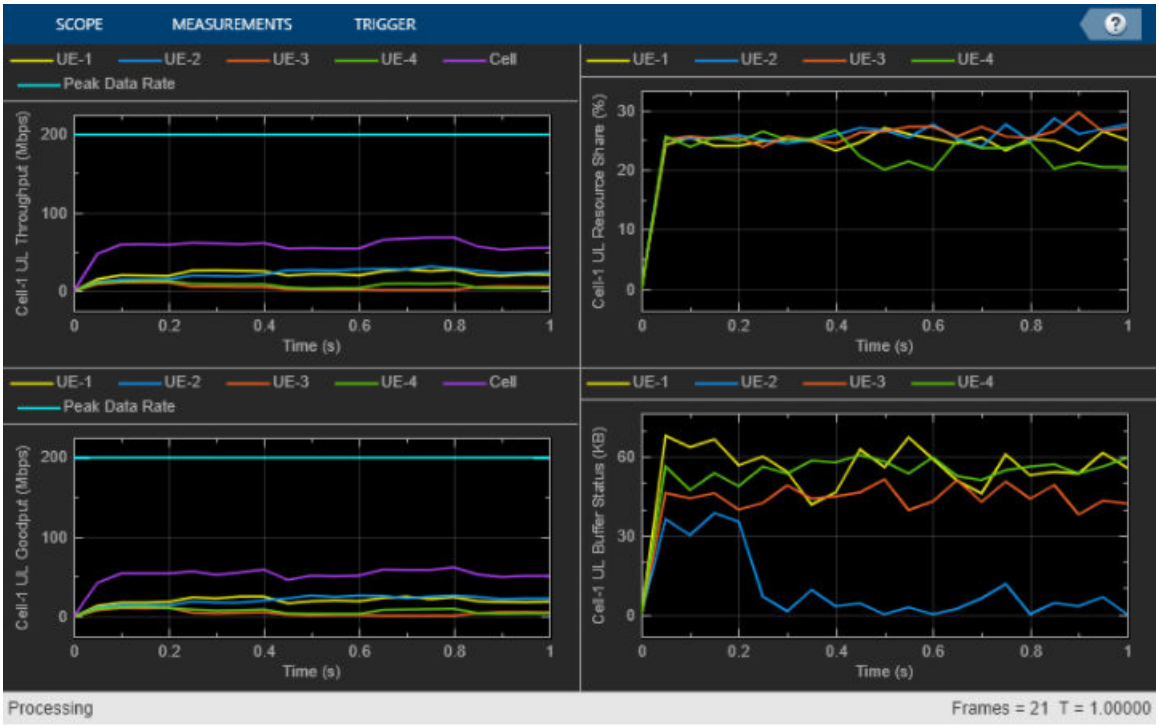
Visualizations

These run-time visualizations display the network performance metrics.

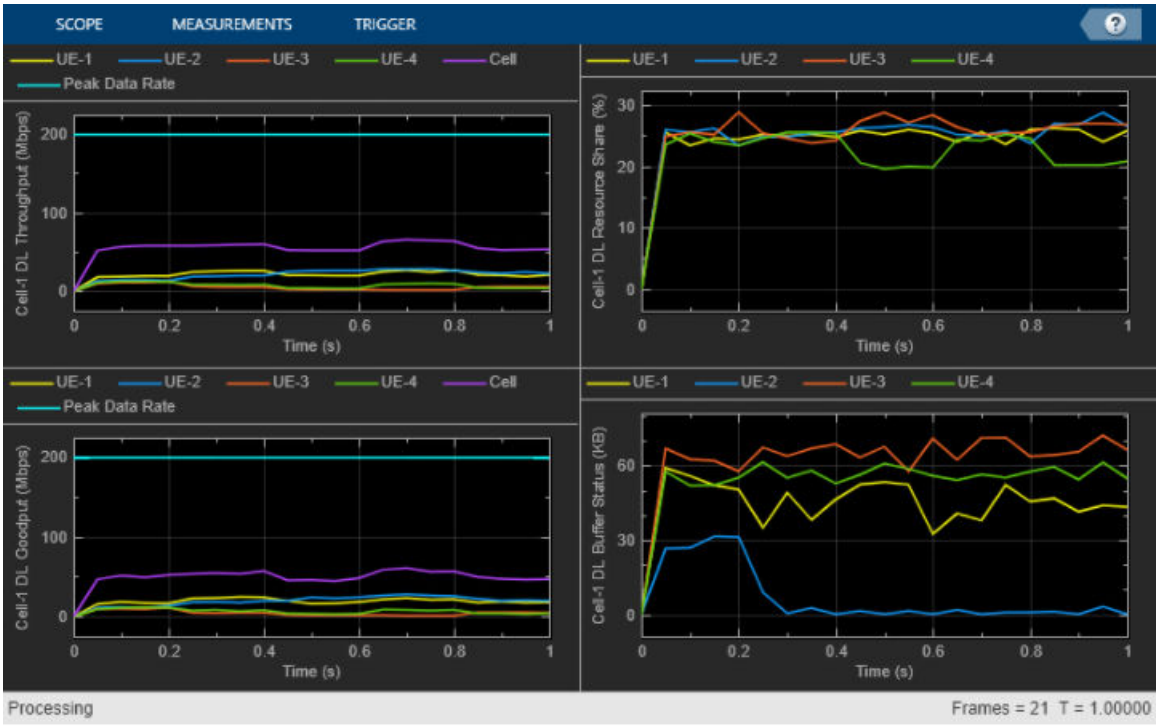
Cell Performance Metrics

This figure includes four plots that display these uplink (UL) metrics.

- UL throughput for each UE and the cell. This value includes both new transmissions and retransmissions.
- UL goodput for each UE and the cell. This value includes new transmissions only.
- Resource share percentage among the UEs with respect to the total UL resources. This value represents the fairness of UL scheduling.
- Pending UL buffer status of the UEs to determine whether the UEs are getting sufficient resources.



Similar to the UL metrics visualization, this figure includes identical plots that display the downlink (DL) metrics.



Set the *NumMetricsSteps* parameter to configure the number of times the plot updates for an entire simulation. These metrics plots update for every *metricsStepSize* slots, where *metricsStepSize* is equal to the total slots in the simulation time divided by *NumMetricsSteps*.

Note *NumMetricsSteps* parameter also configures the plot update times for "RLC Metrics" on page 6-4 and "Block Error Rate Metrics" on page 6-5 in an entire simulation.

The peak spectral efficiency value is calculated using the following formula:

$$SEp = \frac{vlayers * Qm * f * R\max * \frac{N_{PRB}^{BW_\mu} * 12}{T_s^\mu} * (1 - OH)}{BW}$$

where:

- v_{layers} is the maximum number of layers.
- Q_m is the maximum modulation order.
- f is the scaling factor and is assumed as 1.
- R_{max} = 948/1024
- $N_{PRB}^{BW_\mu}$ is the maximum resource block (RB) allocation in the bandwidth with a subcarrier spacing of μ.
- T_s^μ is the average orthogonal frequency division multiplexing (OFDM) symbol duration in a subframe.
- *OH* is the overhead calculated as the average ratio of the number of occupied resource elements (REs) to the total number of REs in effective bandwidth time product. For these metrics visualizations, *OH* is assumed to be 0.
- *BW* is the UE-supported maximum bandwidth in the given band.

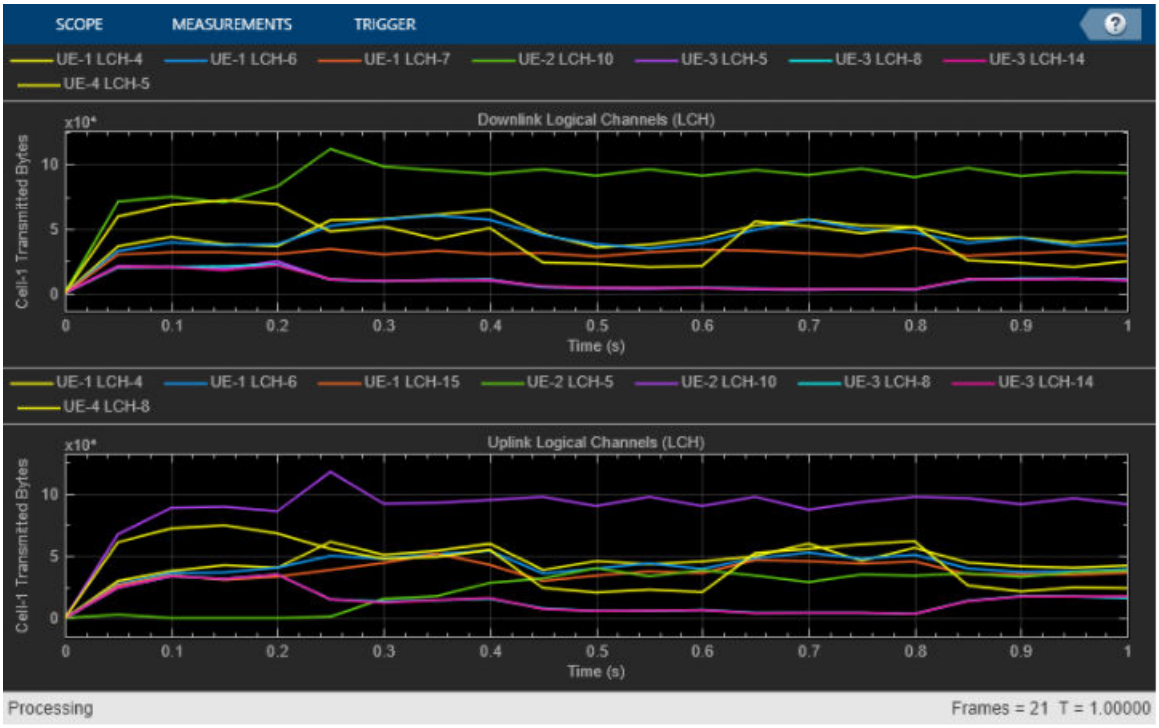
For more information, refer to 3GPP TR 37.910 Section 5.1.1 [1].

The maximum achievable throughput corresponds to the numerator in the above formula and is shown with a dashed line in throughput and goodput plots.

RLC Metrics

These two subplots show the number of bytes transmitted by the RLC layer per logical channel for each UE in the UL and DL direction, respectively. As described in "Cell Performance Metrics" on page 6-2 section, *NumMetricsSteps* parameter configures the number of times the plot updates for an entire simulation.

Pass a value of `0` or `1` to `hNRMetricsVisualizer` constructor to toggle the display of the plot before the start of a simulation. A value of `1` enables this metrics plot. A value of `0` disables this metrics plot.



Block Error Rate Metrics

These two subplots show the block error rates (BLER) observed for each UE in the UL and DL direction, respectively. As described in “Cell Performance Metrics” on page 6-2 section, *NumMetricsSteps* parameter configures the number of times the plot updates for an entire simulation.

Pass a value of `0` or `1` to `hNRMetricsVisualizer` constructor to toggle the display of the plot before the start of a simulation. A value of `1` enables this metrics plot. A value of `0` disables this metrics plot.

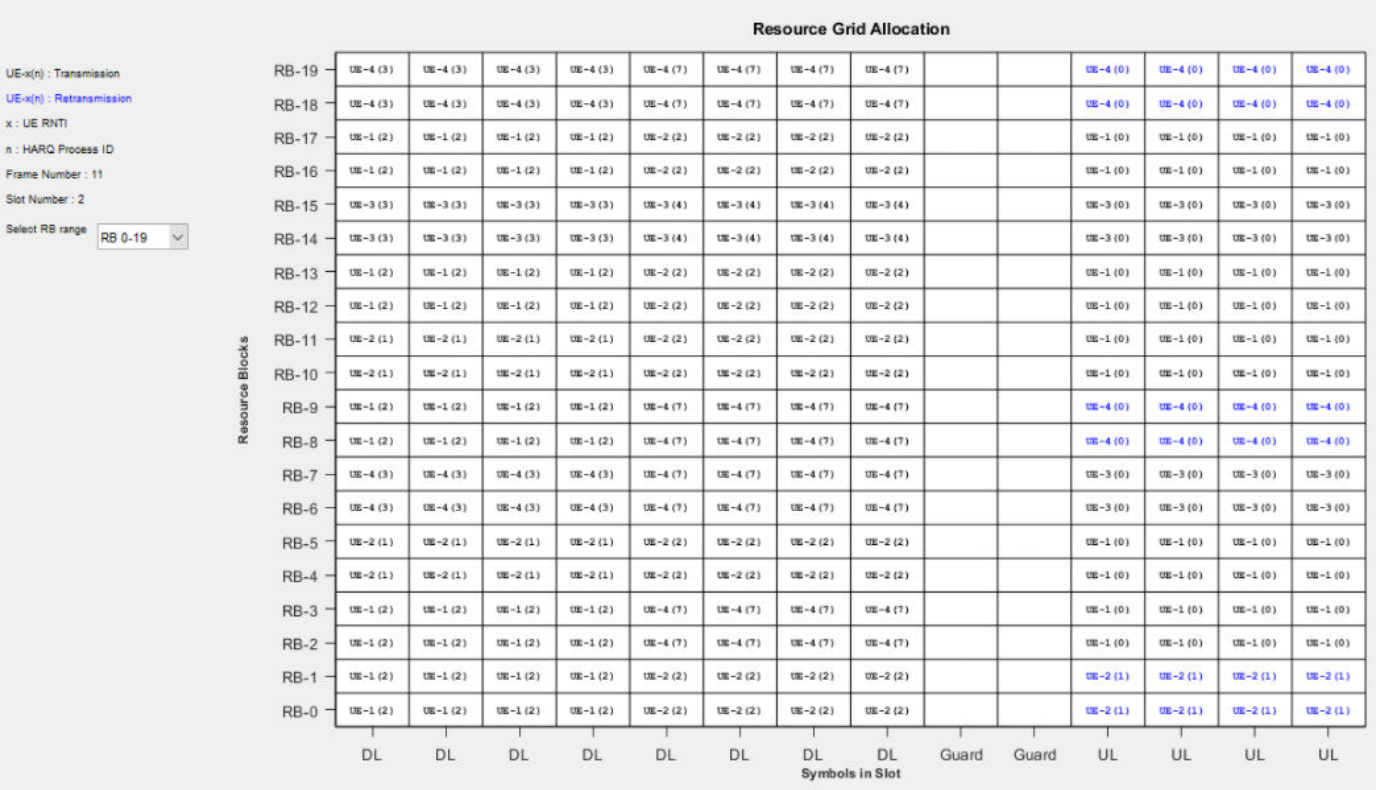
Resource Grid Allocation

This 2-D time-frequency grid shows the resource allocation to the UEs on a frame-by-frame basis. The hybrid automatic repeat request (HARQ) process for the physical uplink shared channel (PUSCH) and physical downlink shared channel (PDSCH) assignments is also shown alongside with the radio network temporary identifier (RNTI) of the UEs. New transmissions are shown in black and retransmissions are shown in blue using the HARQ process ID of each UE. Using the HARQ ID, you can map a retransmission assignment to its previously failed transmission.

You can visualize the resource allocations for a selected RB range from the drop down present in the top left of the plot.

The plot is updated for each frame and shows the RB allocation to the UEs in the last complete frame.

Set *simParameters.RBVisualization* parameter to toggle the display of the plot before the start of a simulation. A value of `1` enables this metrics plot. A value of `0` disables this metrics plot.

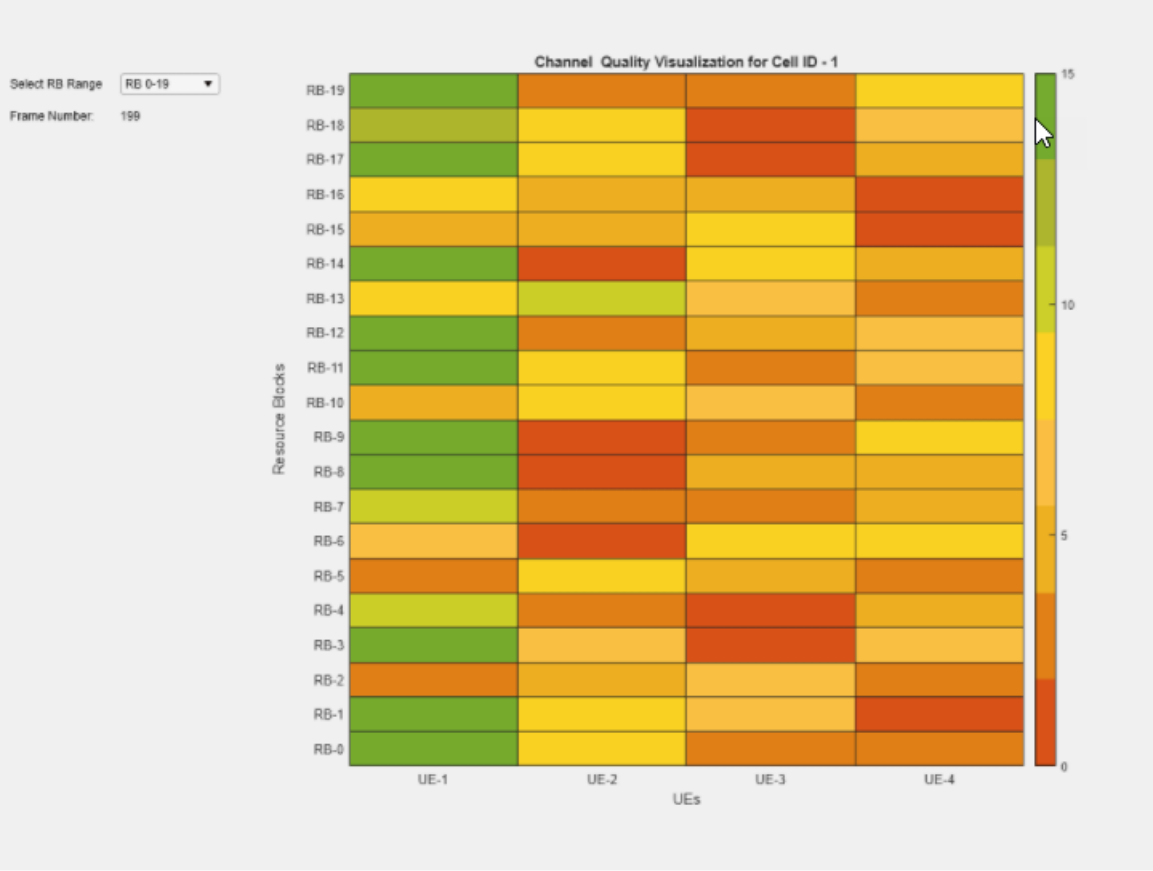


Channel Quality

This figure shows the channel quality index (CQI) of the UEs in a selected frame number over the PUSCH or PDSCH bandwidth. You can select and visualize the CQI values of 20 RBs at a time from the drop down present in the top left of the plot.

The plot is updated for each frame.

Set *simParameters.CQIVisualization* parameter to toggle the display of the plot before the start of a simulation. A value of `1` enables this metrics plot. A value of `0` disables this metrics plot.



Results

At the end of the simulation, you can compare the achieved value for system performance indicators to their theoretical peak values considering zero overheads.

Performance indicators displayed in this sample result summary are achieved data rate (UL and DL), achieved spectral efficiency (UL and DL), and BLER observed for UEs (DL and UL). The peak values are calculated as per 3GPP TR 37.910 [1].

```
Peak UL Throughput: 124.42 Mbps. Achieved Cell UL Throughput: 27.44 Mbps
Achieved UL Throughput for each UE: [9.66        9.22        8.12        0.44]
Achieved Cell UL Goodput: 27.44 Mbps
Achieved UL Goodput for each UE: [9.66        9.22        8.12        0.44]
Peak UL spectral efficiency: 24.88 bits/s/Hz.
Achieved UL spectral efficiency for cell: 5.49 bits/s/Hz

Peak DL Throughput: 62.21 Mbps. Achieved Cell DL Throughput: 36.06 Mbps
Achieved DL Throughput for each UE: [10.73         9.85         9.14         6.34]
Achieved Cell DL Goodput: 36.06 Mbps
Achieved DL Goodput for each UE: [10.73         9.85         9.14         6.34]
Peak DL spectral efficiency: 12.44 bits/s/Hz.
Achieved DL spectral efficiency for cell: 7.21 bits/s/Hz

Block error rate for each UE in the uplink direction: [0  0  0  0]
Block error rate for each UE in the downlink direction: [0  0  0  0]
```

Post-Simulation

Logs of the simulation are available after the entire simulation is run. You can run the script `NRPostSimVisualization` to get a quick post-simulation visualization of the logs.

In the post-simulation script, the variable *LogReplay* provides these options to display *Resource Grid Allocation* and *Channel Quality* figures.

- Set *LogReplay* to `true` for a replay of the simulation logs.
- Set *LogReplay* to `false` to analyze the details of a particular frame or a particular slot of a frame. In the *Resource Grid Allocation* window, input the frame number and slot number to visualize the resource assignment of the particular slot, if scheduling type is symbol-based. For slot-based scheduling, enter the frame number to visualize the resource assignment for the entire frame. The frame number entered here controls the frame number for *Channel Quality* figure too.

References

[1] 3GPP TR 37.910. “Study on self evaluation towards IMT-2020 submission.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

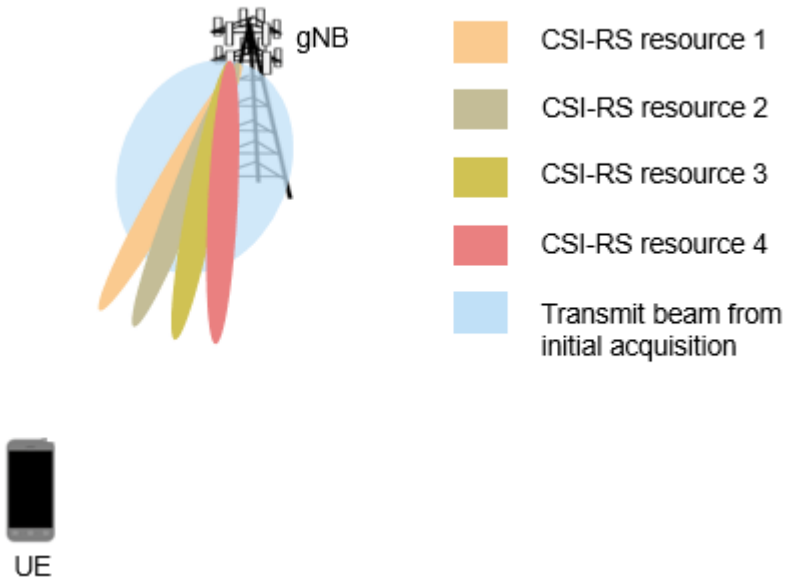
- “NR Cell Performance Evaluation with MIMO” on page 6-41
- “NR TDD Symbol Based Scheduling Performance Evaluation” on page 6-68
- “NR FDD Scheduling Performance Evaluation” on page 6-56
- “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87

NR Cell Performance Evaluation with Beam Management

This example models a 5G New Radio (NR) cell with transmit-end beam refinement procedure in the downlink (DL) direction and evaluates the network performance. This example shows how to transmit multiple single-port channel state information reference signal (CSI-RS) resources in different directions for Layer 1 (physical layer) reference signal received power (L1-RSRP) measurements. The gNB scheduler uses the L1-RSRP reports to beamform the physical downlink shared channel (PDSCH) transmissions for better signal-to-noise-ratio (SNR) at the UEs. You can modify the gNB scheduler to select the beam direction for PDSCH transmissions.

Introduction

Downlink beamforming improves the network performance by utilising the multi-antenna configuration to focus energy of the signals in a direction which results in better SNR at a UE. Additionally, beamforming also limits the interference in other directions. For efficient DL beamforming, gNB relies on the channel state information resource indicator (CRI) and L1-RSRP feedback from the UEs. The UEs measure the L1-RSRP on multiple CSI-RS resources where each resource corresponds to one direction, and reports the gNB with the CRI having the highest L1-RSRP.



The example models multiple sets of CSI-RS beams for L1-RSRP measurements. Each set corresponds to a wider synchronization signal block (SSB) beam and contains finer CSI-RS beams within the SSB beam. The example does not model the SSB beam sweeping process for initial acquisition, and assumes the SSB beam associated with each UE to be the beam with angular range covering the line-of-sight path from the gNB.

This example models:

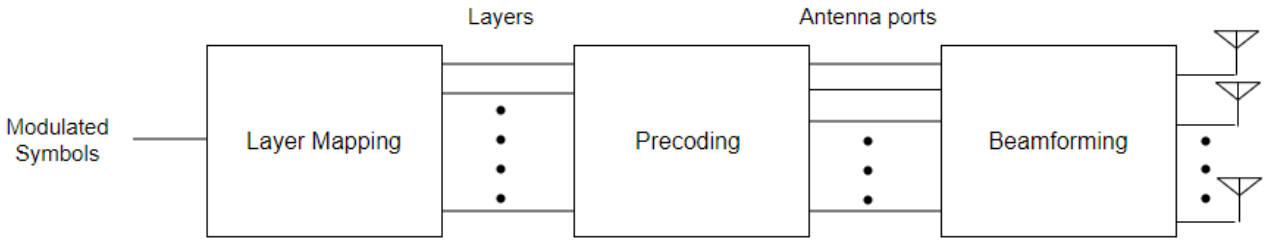
- DL channel quality measurement and reporting by UEs based on the multi-port CSI-RS received from the gNB. The report includes rank indicator (RI), precoding matrix indicator (PMI), and channel quality indicator (CQI). The example supports Type-1 single-panel codebook for PMI.

- DL L1-RSRP measurement and CRI reporting by UEs based on the single-port CSI-RS received from the gNB.
- Transmit end beam refinement in DL direction using CRI-RSRP feedback.
- Single-codeword DL spatial multiplexing to perform multi-layer transmission. Single-codeword limits the number of transmission layers to 4.
- Precoding to map the transmission layers to antenna ports.
- Digital beamforming of PDSCH based on the CRI feedback from the UEs.
- Free space path loss (FSPL), additive white Gaussian noise (AWGN), and clustered delay line (CDL) propagation channel model.

Nodes send the control packets (buffer status report (BSR), DL assignment, UL grants, PDSCH feedback, CSI report, and CRI-RSRP reports) out of band, without the need of resources for transmission and assured error-free reception.

MIMO

The key aspects of multiple-input multiple-output (MIMO) include precoding, beamforming, channel measurement and reporting.



Layer mapping

The layer mapping process maps the modulated symbols of the codeword onto different layers.

Precoding

Precoding, which follows the layer mapping, maps the transmission layers to antenna ports. Precoding applies a precoding matrix to the transmission layers and outputs data streams to the antenna ports.

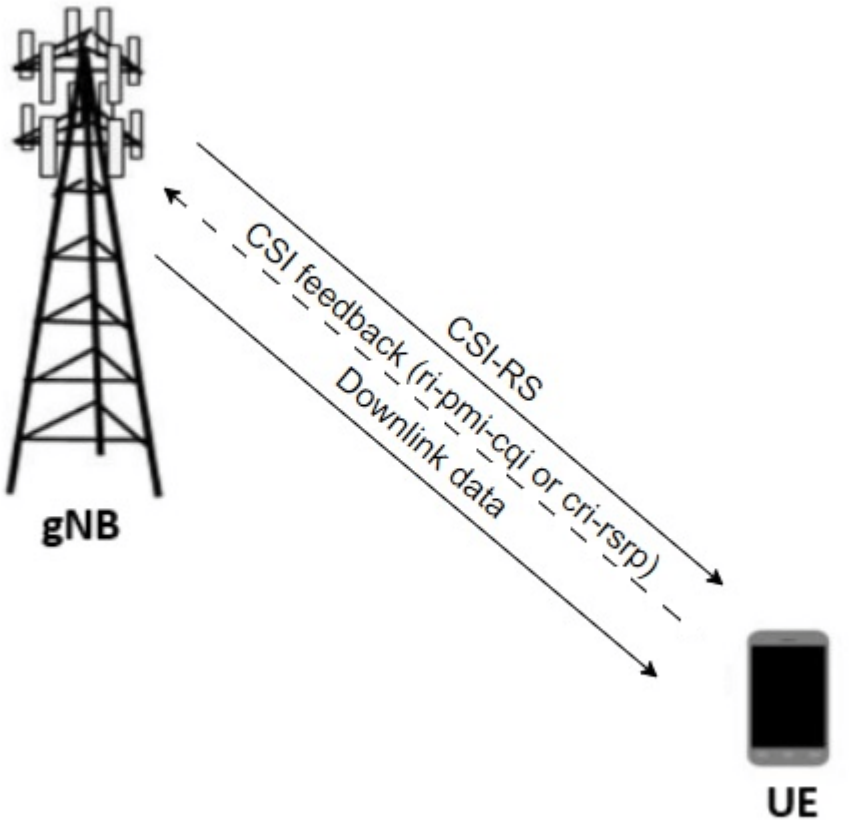
Beamforming

Beamforming involves the use of multiple radiating elements transmitting the same signal to produce a longer and narrower beam in a particular direction. The higher the number of antennas, the narrower the beamwidth.

Beam management is a set of physical (PHY) layer and medium access control (MAC) layer procedures to acquire and maintain a set of beam pair links (a beam used at gNB paired with a beam used at UE). There are three procedures defined for beam management. For more detailed information about the beam management procedure, refer to “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113 example.

The current example focuses on the Procedure 2 (P-2) beam measurement and beam reporting procedures which involves sending single port CSI-RS resources over highly directional beams to each UE. The L1-RSRP measurements over each of these resources are then reported back to the gNB which is used to perform transmit beam refinement for all subsequent downlink transmissions intended for the UE till the next L1-RSRP measurement.

DL Channel Measurement and Reporting



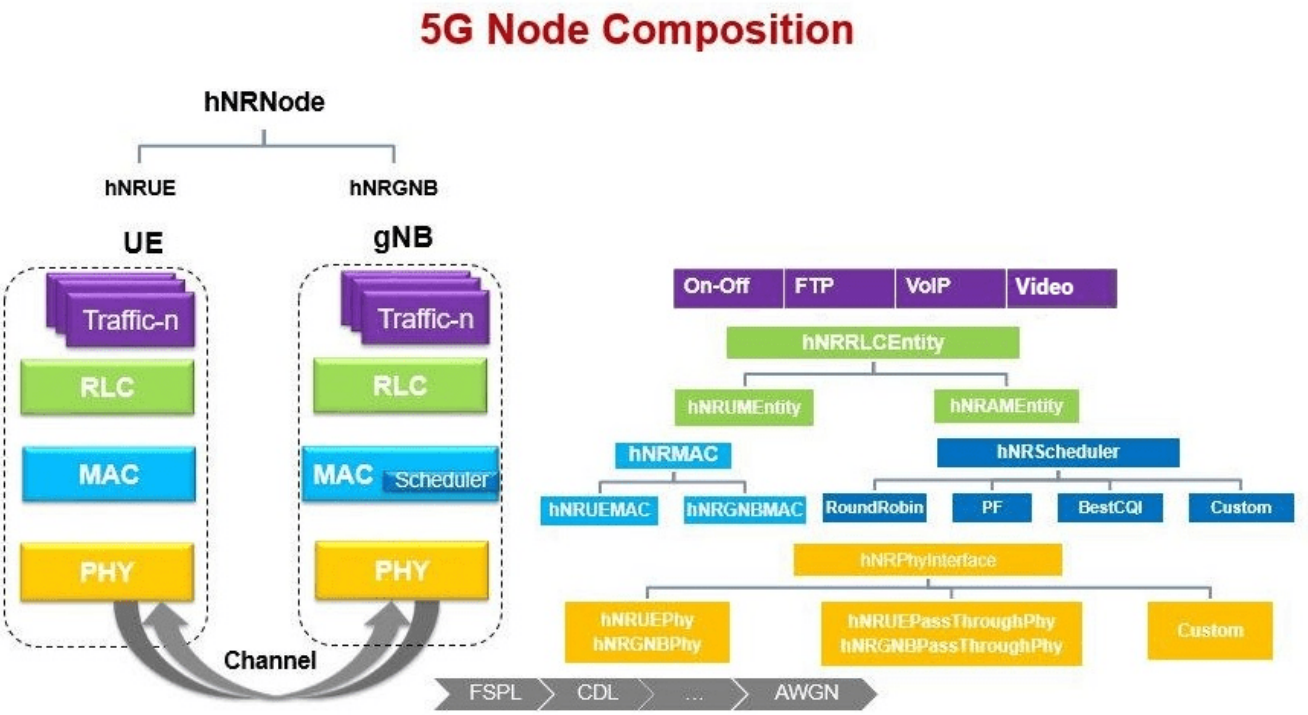
CSI reporting is the process by which a UE, for DL transmissions, advises a suitable number of transmission layers (rank), PMI, and CQI values to the gNB. The UE estimates these values by performing channel measurements on its configured CSI-RS resources. For more details, see the “5G NR Downlink CSI Reporting” on page 5-47 example. The gNB scheduler uses this advice to decide the number of transmission layers, precoding matrix, and modulation and coding scheme (MCS) for PDSCHs.

The UE also uses the CSI report to indicate the CRI, and the L1-RSRP to aid the gNB in transmit beam refinement.

NR Protocol Stack

A node (gNB or UE) is a composition of NR stack layers. The helper classes hNRGNB.m and hNRUE.m create gNB and UE nodes, respectively, containing the radio link control (RLC), MAC and

PHY layer. For more details, see the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.



Scenario Configuration

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Configure simulation parameters in the `simParameters` structure.

```
rng('default');                 % Reset the random number generator
simParameters = [];             % Clear the simParameters variable
simParameters.NumFramesSim = 10; % Simulation time in terms of number of 10 ms frames
```

Enable or disable DL beamforming.

```
enableBeamforming = true;
```

Specify the azimuth sweep angles and direction of SSBs.

Azimuth angle is the angle measured in the horizontal plane ranging from [-180, 180] and

Elevation angle is the angle measured in the vertical plane ranging from [-90, 90] between horizontal plane and line of sight (LOS) point in the Spherical coordinate system.

```
if enableBeamforming
    % Number of SSBs
```

```
    numSSBs = 4;
    % Total azimuthal range for all SSBs corresponding to a 120 degrees sectorized cell
    azSweepRange = [-60, 60];
    validateattributes(azSweepRange,{'numeric'},{'nonempty','real', 'vector', ...
        '>=',-60,'<=',60,'finite'}, ...
        'azSweepRange','azimuthal range');
    % Sweep range for non-overlapping SSB directions
    ssbSweepRange = diff(azSweepRange)/numSSBs;
    % Azimuthal angles for SSB beam sweeping in the azimuthal range
    ssbTxAngles = [-60 -20 20 60];
    % Elevation range for all SSBs sweeping in the azimuthal angles
    elSweepRange = [-45, 0];
end
```

Specify the number of UEs in the cell, assuming that UEs have sequential radio network temporary identifiers (RNTIs) from 1 to `simParameters.NumUEs`. If you change the number of UEs, ensure that the number of rows in `simParameters.UEPosition` is equal to the value of simParameters.NumUEs.

```
simParameters.NumUEs = 3;
% Position of gNB in (x,y,z) coordinates
simParameters.GNBPosition = [0, 0, 0];
% Specify UE position in spherical coordinates (r,azimuth,elevation)
% relative to gNB. Here 'r' is the distance, 'azimuth' is the angle in the
% horizontal plane and elevation is the angle in the vertical plane. When
% beamforming is enabled, all the UE positions must lie with in the azimuth
% and elevation range limits of SSBs as defined above. It is a matrix of
% size N-by-3 where N is the number of UEs. Row 'i' of the matrix contains
% the coordinates of UE with RNTI 'i'
ueRelPosition = [500, -5, -5;
    500, -60, -30;
    500, -10, -20];
if enableBeamforming
    validateattributes(ueRelPosition(:,2),{'numeric'},{'nonempty','real','vector', ...
        '>=',azSweepRange(1),'<=',azSweepRange(2),'finite'}, ...
        'ueRelPosition(:,2)','azimuth angles');
    validateattributes(ueRelPosition(:,3),{'numeric'},{'nonempty','real','vector', ...
        '>=',elSweepRange(1),'<=',elSweepRange(2),'finite'}, ...
        'ueRelPosition(:,3)','elevation angles');
end
% Convert Spherical to Cartesian coordinates considering gNB position as origin
[xPos, yPos, zPos] = sph2cart(deg2rad(ueRelPosition(:, 2)),deg2rad(ueRelPosition(:, 3)), ...
    ueRelPosition(:, 1));
% Global coordinates of UEs
simParameters.UEPosition = [xPos, yPos, zPos] + simParameters.GNBPosition;
% Validate the UE positions
validateattributes(simParameters.UEPosition, {'numeric'},{'nonempty','real', ...
    'nrows',simParameters.NumUEs,'ncols',3,'finite'}, ...
    'simParameters.UEPosition','UEPosition');
```

Set the channel bandwidth to 5 MHz and the subcarrier spacing (SCS) to 15 kHz as defined in 3GPP TS 38.104 Section 5.3.2.

```
simParameters.NumRBs = 25;
simParameters.SCS = 15;                 % kHz
simParameters.DLBandwidth = 5e6;        % Hz
simParameters.ULBandwidth = 5e6;        % Hz
```

```
simParameters.DLCarrierFreq = 2.646e9; % Hz
simParameters.ULCarrierFreq = 2.535e9; % Hz
```

Specify the transmit (Tx) and receive (Rx) antenna panel geometry at the gNB and UEs in the format `(M,N,P)`, where `M` and `N` are the number of rows and columns in the antenna array, respectively. `P` is the number of polarizations (1 or 2).

```
gNBTxArraySize = [2 4 2];
ueRxArraySize = repmat([1 1 2],simParameters.NumUEs,1);
```

Configure the gNB transmit antenna panel.

```
lambda = physconst('LightSpeed')/simParameters.DLCarrierFreq;
simParameters.TxAntPanel = phased.NRRectangularPanelArray('ElementSet', ...
    repmat({phased.NRAntennaElement},1,gNBTxArraySize(3)), ...
    'Size',[gNBTxArraySize(1:2) 1 1],'Spacing', ...
    [0.5*lambda 0.5*lambda 1 1]);
```

Specify the CSI-RS configuration for RI, PMI and CQI measurements.

```
csirs = cell(1,simParameters.NumUEs);
csirsOffset = [5 6 10 11];
for csirsIdx = 1:simParameters.NumUEs
    csirs{csirsIdx} = nrCSIRSConfig('NID',1,'NumRB',simParameters.NumRBs, ...
        'RowNumber',11,'SubcarrierLocations',[1 3 5 7], ...
        'SymbolLocations',0,'CSIRSPeriod',[20 csirsOffset(csirsIdx)]);
end
simParameters.CSIRSConfig = csirs;
```

Specify the CSI-RS resource set configuration associated with each SSB for DL beam refinement. Each resource set contains multiple single-port CSI-RS resources, with each resource corresponding to a particular beam direction within the SSB angular range.

```
if enableBeamforming
    simParameters.NumCSIRSBeams = 4;
    csirsConfigRSRP = cell(1,numSSBs);
    for ssbIdx = 1:numSSBs
        csirsConfig = nrCSIRSConfig;
        csirsConfig.NID = 1;
        csirsConfig.CSIRSType = repmat({'nzp'},1,simParameters.NumCSIRSBeams);
        csirsConfig.CSIRSPeriod = [40 0];
        csirsConfig.Density = repmat({'one'},1,simParameters.NumCSIRSBeams);
        csirsConfig.RowNumber = repmat(2,1,simParameters.NumCSIRSBeams);
        csirsConfig.SymbolLocations = {1,5,6,7};
        csirsConfig.SubcarrierLocations = repmat({ssbIdx-1},1, ...
            simParameters.NumCSIRSBeams);
        csirsConfig.NumRB = simParameters.NumRBs;
        csirsConfigRSRP{ssbIdx} = csirsConfig;
    end
    simParameters.CSIRSConfigRSRP = csirsConfigRSRP;
end
```

Specify the signal-to-interference-plus-noise ratio (SINR) to a CQI index mapping table for a block error rate (BLER) of 0.1. The lookup table corresponds to the CQI table as per 3GPP TS 38.214 Table 5.2.2.1-3.

```
simParameters.DownlinkSINR90pc = [-0.4600 4.5400 9.5400 14.0500 16.540 19.0400 ...
    20.5400 23.0400 25.0400 27.4300 29.9300 30.4300 ...
    32.4300 35.4300 38.4300];
```

Specify the gNB transmit power.

```
simParameters.GNBTxPower = 34; % In dBm
```

Configure the channel model.

```
channelModelDL = cell(1,simParameters.NumUEs);
waveformInfo = nrOFDMInfo(simParameters.NumRBs,simParameters.SCS);
dlchannel = nrCDLChannel('DelayProfile','CDL-D','DelaySpread',300e-9);
for ueIdx = 1:simParameters.NumUEs
    cdl = hMakeCustomCDL(dlchannel);
    % Configure the channel seed based on the UE number
    % Independent fading for each UE
    cdl.Seed = cdl.Seed + (ueIdx - 1);
    % Compute the LOS angle from gNB to UE
    [~,depAngle] = rangeangle(simParameters.UEPosition(ueIdx, :)', ...
        simParameters.GNBPosition');
    % Configure the azimuth and zenith angle offsets for this UE
    cdl.AnglesAoD(:) = cdl.AnglesAoD(:) + depAngle(1);
    % Convert elevation angle to zenith angle
    cdl.AnglesZoD(:) = cdl.AnglesZoD(:) - cdl.AnglesZoD(1) + (90 - depAngle(2));
    % Compute the range angle from UE to gNB
    [~,arrAngle] = rangeangle(simParameters.GNBPosition',...
        simParameters.UEPosition(ueIdx, :)');
    % Configure the azimuth and zenith arrival angle offsets for this UE
    cdl.AnglesAoA(:) = cdl.AnglesAoA(:) - cdl.AnglesAoA(1) + arrAngle(1);
    % Convert elevation angle to zenith angle
    cdl.AnglesZoA(:) = cdl.AnglesZoA(:) - cdl.AnglesZoA(1) + (90 - arrAngle(2));
    cdl.CarrierFrequency = simParameters.DLCarrierFreq;
    cdl.TransmitAntennaArray = simParameters.TxAntPanel;
    cdl.ReceiveAntennaArray.Size = [ueRxArraySize(ueIdx, :) 1 1];
    cdl.SampleRate = waveformInfo.SampleRate;
    channelModelDL{ueIdx} = cdl;
end
```

Initialize the beamforming weights table for DL beam refinement. It contains the steering vectors corresponding to direction of various CSI-RS beams across all the SSBs. It is of size M x N, where M represents number of transmit antenna elements and N represents number of CSI-RS beams of all SSBs.This table is stored at gNB PHY. For each PDSCH, scheduler selects a column index in this table (corresponding to the beam direction) and conveys the same to PHY.

```
% Number of transmit antennas
numTxAntennas = prod(gNBTxArraySize);
if enableBeamforming
    enableVerticalBeamSweep = true;
    simParameters.BeamWeightTable = zeros(numTxAntennas, ...
        simParameters.NumCSIRSBeams*numSSBs);
    txArrayStv = phased.SteeringVector('SensorArray',simParameters.TxAntPanel, ...
        'PropagationSpeed',physconst('LightSpeed'), ...
        'IncludeElementResponse',true);
    azBW = beamwidth(simParameters.TxAntPanel,simParameters.DLCarrierFreq, ...
        'Cut','Azimuth');
    elBW = beamwidth(simParameters.TxAntPanel,simParameters.DLCarrierFreq, ...
        'Cut','Elevation');
```

```
    for beamIdx = 1:numSSBs
        % Get the azimuthal sweep range for every SSB based on the SSB transmit
        % beam direction and its beamwidth in azimuth plane
        azSweepRangeSSB = ssbTxAngles(beamIdx) + [-ssbSweepRange/2 ssbSweepRange/2];
        % Get the azimuth and elevation angle pairs for all CSI-RS transmit beams
        csirsBeamAng = hGetBeamSweepAngles(simParameters.NumCSIRSBeams,azSweepRangeSSB, ...
            elSweepRange,azBW,elBW,enableVerticalBeamSweep);
        for csirsBeamIdx = 1:simParameters.NumCSIRSBeams
            simParameters.BeamWeightTable(:, simParameters.NumCSIRSBeams*(beamIdx-1) ...
                + csirsBeamIdx) = txArrayStv(simParameters.DLCarrierFreq, ...
                csirsBeamAng(:,csirsBeamIdx))/sqrt(numTxAntennas);
        end
    end

    % Initialize the beam indices for all UEs with highest L1-RSRP (for visualization purpose)
    beamIndices = -1*ones(1,simParameters.NumUEs);
end
```

Specify the scheduling strategy and the maximum limit on the RBs allotted for PDSCH. The transmission limit applies only to new transmissions and not to the retransmissions.

```
% Supported scheduling strategies: 'PF', 'RR', and 'BestCQI'
simParameters.SchedulerStrategy = 'PF';
simParameters.RBAllocationLimitDL = 15; % For PDSCH
```

Logging and visualization configuration

The `CQIVisualization` and `RBVisualization` parameters control the display of the CQI visualization and the RB assignment visualization respectively. To enable the RB visualization plot, set the `RBVisualization` field to `true`.

```
simParameters.CQIVisualization = true;
simParameters.RBVisualization = false;
```

Set the `enableTraces` to `true` to log the traces. If the `enableTraces` is set to `false`, then `CQIVisualization` and `RBVisualization` are disabled automatically and traces are not logged in the simulation. To speed up the simulation, set the `enableTraces` to `false`.

```
enableTraces = true;
```

The example updates the metrics plots periodically. Set the number of updates during the simulation.

```
simParameters.NumMetricsSteps = 10;
```

Write the logs to MAT-files. You can use these log files for post-simulation analysis and visualization.

```
parametersLogFile = 'simParameters';         % For logging the simulation parameters
simulationLogFile = 'simulationLogs';        % For logging the simulation traces
simulationMetricsFile = 'simulationMetrics'; % For logging the simulation metrics
```

Application traffic configuration

Set the periodic DL application traffic pattern for UEs.

```
% DL application data rate in kilo bits per second (kbps)
dlAppDataRate = 40e3*ones(1,simParameters.NumUEs);
% Validate the DL application data rate
validateattributes(dlAppDataRate,{'numeric'},{'nonempty','vector','numel', ...
```

```
    simParameters.NumUEs,'finite','>',0},'dlAppDataRate', ...
    'dlAppDataRate');
```

Derived Parameters

Compute the derived parameters based on the primary configuration parameters specified in the previous section and set some example-specific constants.

```
simParameters.DuplexMode = 0;                    % FDD (Value as 0) or TDD (Value as 1)
simParameters.SchedulingType = 0;                % Slot-based scheduling
simParameters.NCellID = 1;                       % Physical cell ID
simParameters.GNBTxAnts = numTxAntennas;
simParameters.UERxAnts = prod(ueRxArraySize,2);
```

Specify the CSI report configuration.

```
% [N1 N2] as per 3GPP TS 38.214 Table 5.2.2.2.1-2
csiReportConfig.PanelDimensions = [8 1];
csiReportConfig.CQIMode = 'Wideband';              % 'Wideband' or 'Subband'
% Set codebook mode as 1 or 2. It is applicable only when the number of
% transmission layers is 1 or 2 and number of CSI-RS ports is greater than 2
csiReportConfig.CodebookMode = 1;
simParameters.CSIReportConfig = {csiReportConfig};
```

Compute the SSB beam corresponding to each UE.

```
if enableBeamforming
    simParameters.SSBIndex = computeSSBToUE(simParameters,ssbTxAngles,ssbSweepRange,ueRelPositio
end
```

Compute the number of slots in the simulation.

```
numSlotsSim = (simParameters.NumFramesSim * 10 * simParameters.SCS)/15;
```

Set the interval at which the example updates metrics visualization in terms of number of slots. Because this example uses a time granularity of one slot, the `MetricsStepSize` field must be an integer.

```
simParameters.MetricsStepSize = ceil(numSlotsSim/simParameters.NumMetricsSteps);
```

Specify one logical channel for each UE, and set the logical channel configuration for all nodes (UEs and gNBs) in the example.

```
numLogicalChannels = 1;
simParameters.LCHConfig.LCID = 4;
```

Specify the RLC entity type in the range [0, 3]. The values 0, 1, 2, and 3 indicate RLC UM unidirectional DL entity, RLC UM unidirectional UL entity, RLC UM bidirectional entity, and RLC AM entity, respectively.

```
simParameters.RLCConfig.EntityType = 2;
```

Create RLC channel configuration structure.

```
% Mapping between logical channel and logical channel group ID
rlcChannelConfigStruct.LCGID = 1;
% Priority of each logical channel
rlcChannelConfigStruct.Priority = 1;
```

```
% Prioritized bitrate (PBR), in kilobytes per second, of each logical channel
rlcChannelConfigStruct.PBR = 8;
% Bucket size duration (BSD), in ms, of each logical channel
rlcChannelConfigStruct.BSD = 10;
rlcChannelConfigStruct.EntityType = simParameters.RLCConfig.EntityType;
rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID;
```

gNB and UEs Setup

Create the gNB and UE objects, initialize the channel quality information for UEs, and set up the logical channel at the gNB and UE. The helper classes hNRGNB.m and hNRUE.m create the gNB node and the UE node, respectively, each containing the RLC, MAC and PHY.

```
simParameters.Position = simParameters.GNBPosition;
% Create gNB node
gNB = hNRGNB(simParameters);
% Create scheduler
switch(simParameters.SchedulerStrategy)
    % Round robin scheduler
    case 'RR'
        scheduler = hNRSchedulerRoundRobin(simParameters);
    % Proportional fair scheduler
    case 'PF'
        scheduler = hNRSchedulerProportionalFair(simParameters);
    % Best CQI scheduler
    case 'BestCQI'
        scheduler = hNRSchedulerBestCQI(simParameters);
end

% Add scheduler to gNB
addScheduler(gNB,scheduler);
% Create the PHY instance
gNB.PhyEntity = hNRGNBPhy(simParameters);
% Configure the PHY
configurePhy(gNB,simParameters);
% Set the interface to PHY
setPhyInterface(gNB);

% Create the set of UE nodes
UEs = cell(simParameters.NumUEs,1);
ueParam = simParameters;
for ueIdx=1:simParameters.NumUEs
    % Position of the UE
    ueParam.Position = simParameters.UEPosition(ueIdx,:);
    ueParam.UERxAnts = simParameters.UERxAnts(ueIdx);
    if enableBeamforming
        ueParam.SSBIdx = simParameters.SSBIndex(ueIdx);
    end
    % Assuming same CSI report configuration for all UEs
    ueParam.CSIReportConfig = simParameters.CSIReportConfig{1};
    ueParam.ChannelModel = channelModelDL{ueIdx};
    UEs{ueIdx} = hNRUE(ueParam,ueIdx);
    % Create the PHY instance
    UEs{ueIdx}.PhyEntity = hNRUEPhy(ueParam,ueIdx);
    % Configure the PHY
    configurePhy(UEs{ueIdx}, ueParam);
    % Set up the interface to PHY
    setPhyInterface(UEs{ueIdx});
```

```
    % Set up logical channel at gNB for the UE
    configureLogicalChannel(gNB,ueIdx,rlcChannelConfigStruct);

    % Set up logical channel at UE
    configureLogicalChannel(UEs{ueIdx},ueIdx,rlcChannelConfigStruct);

    % Set up application traffic
    % Create an object for on-off network traffic pattern for the specified
    % UE and add it to the gNB. This object generates the downlink data
    % traffic on the gNB for the UE
    dlApp = networkTrafficOnOff('GeneratePacket',true, ...
        'OnTime',simParameters.NumFramesSim*10e-3,'OffTime',0, ...
        'DataRate',dlAppDataRate(ueIdx));
    addApplication(gNB,ueIdx,simParameters.LCHConfig.LCID,dlApp);
end
```

Simulation

```
% Initialize wireless network simulator
nrNodes = [{gNB}; UEs];
networkSimulator = hWirelessNetworkSimulator(nrNodes);
```

Create objects for MAC and PHY metrics logging.

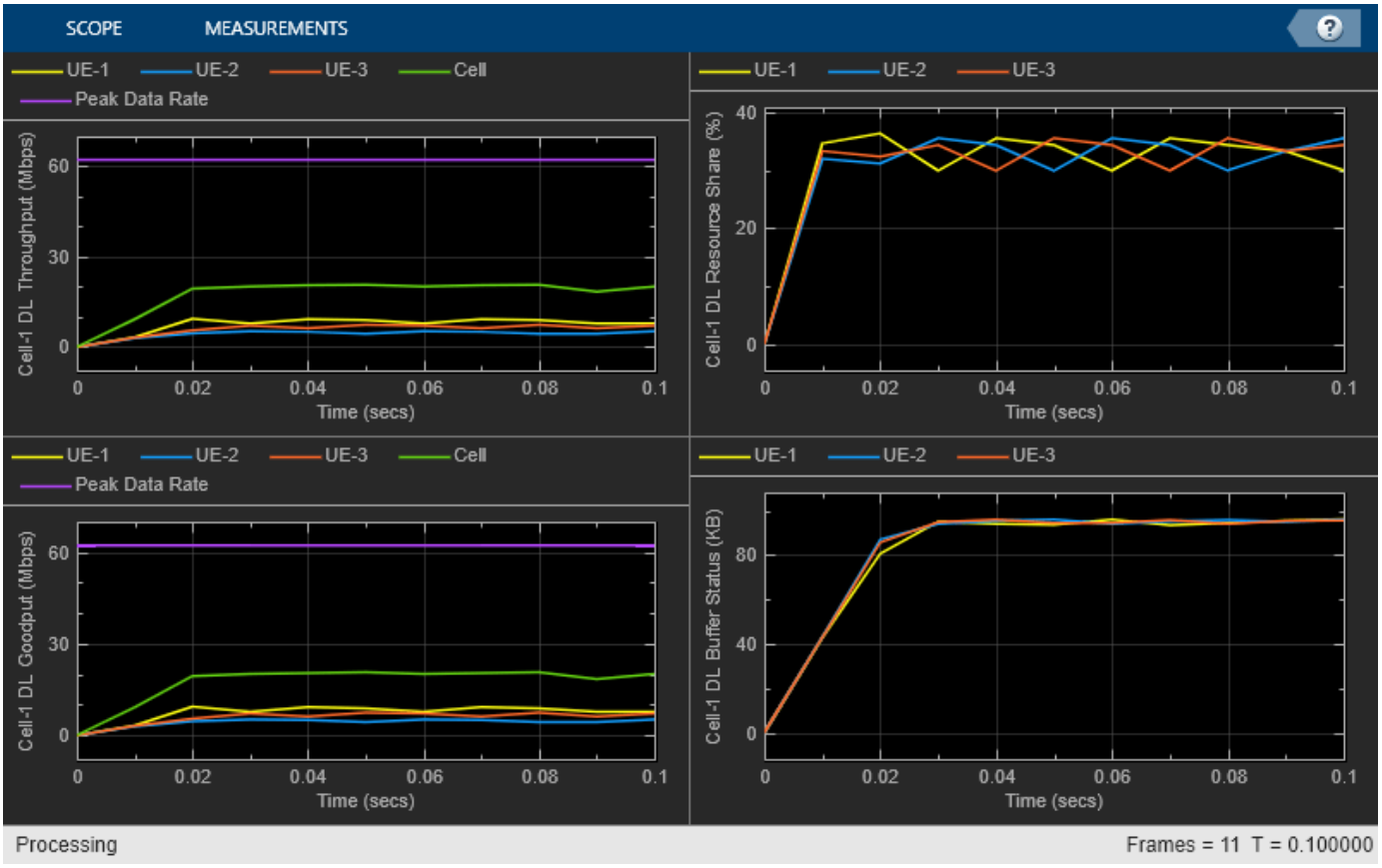
```
nodes = struct('UEs', {UEs}, 'GNB', gNB);
linkDir = 0; % Indicates DL
if enableTraces
    % Create an object for MAC traces logging
    simSchedulingLogger = hNRSchedulingLogger(simParameters, networkSimulator, gNB, UEs, linkDir)

    % Create an object for PHY traces logging
    simPhyLogger = hNRPhyLogger(simParameters, networkSimulator, gNB, UEs);

    % Create an object for CQI and RB grid visualization
    if simParameters.CQIVisualization || simParameters.RBVisualization
        gridVisualizer = hNRGridVisualizer(simParameters, 'MACLogger', simSchedulingLogger, 'Visu
    end
end
```

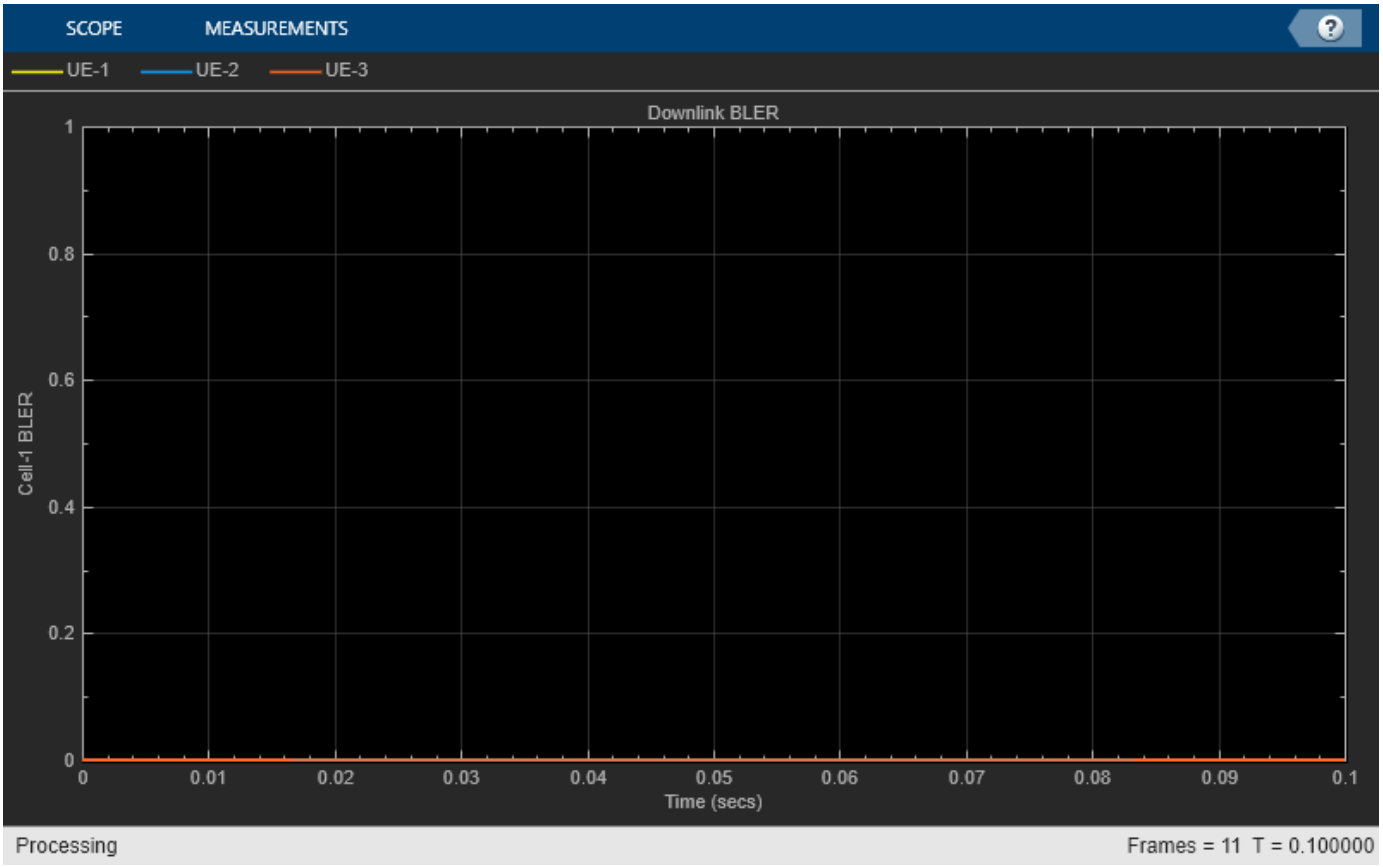
Create an object for MAC and PHY metrics visualization.

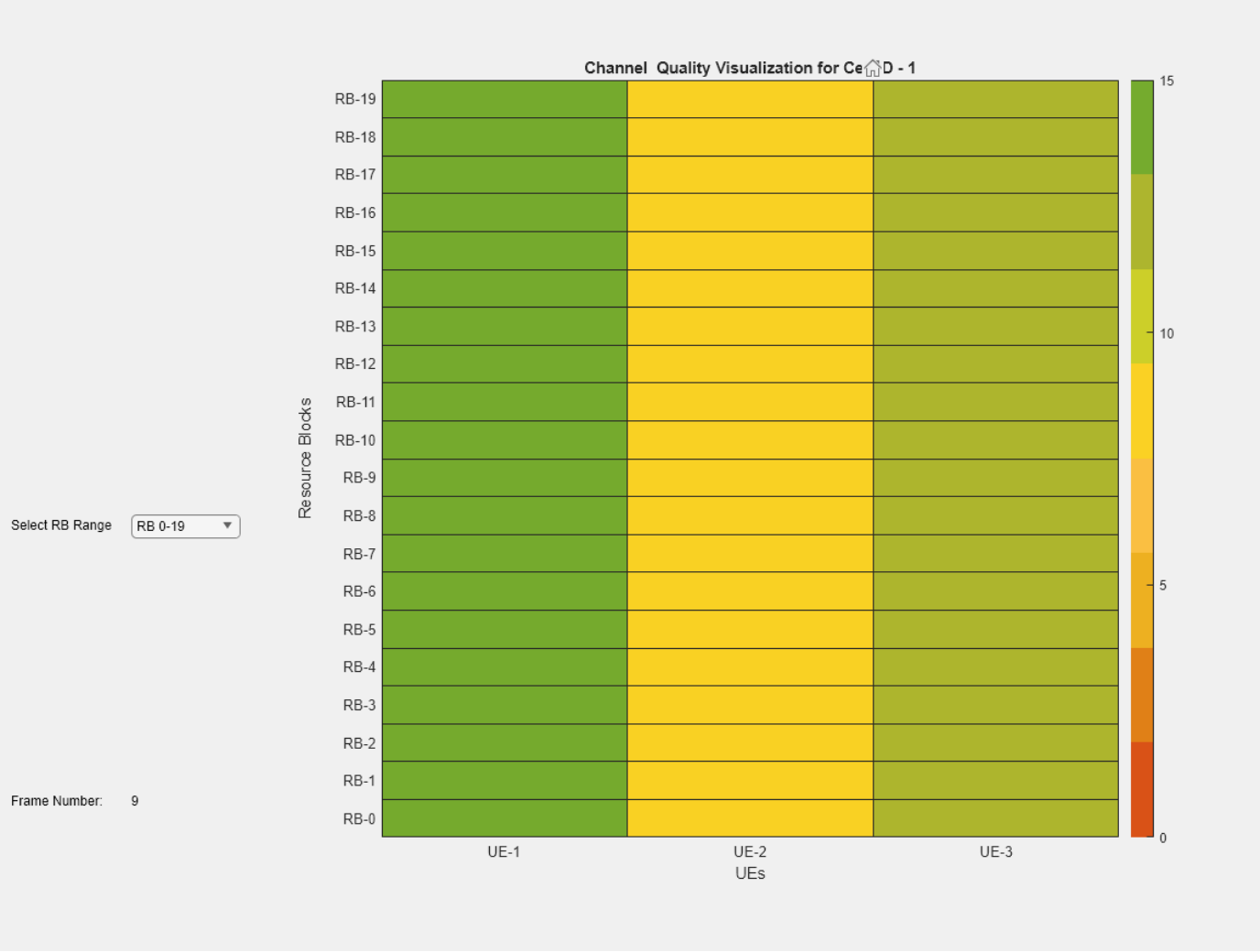
```
metricsVisualizer = hNRMetricsVisualizer(simParameters, 'EnableSchedulerMetricsPlots', true, 'Ena
    'NetworkSimulator', networkSimulator, 'GNB', gNB, 'UEs', UEs);
```

Run the simulation for the specified `NumFramesSim` frames.

```
% Calculate the simulation duration (in seconds) from 'NumFramesSim'
simulationTime = simParameters.NumFramesSim * 1e-2;
% Run the simulation
run(networkSimulator, simulationTime);
```

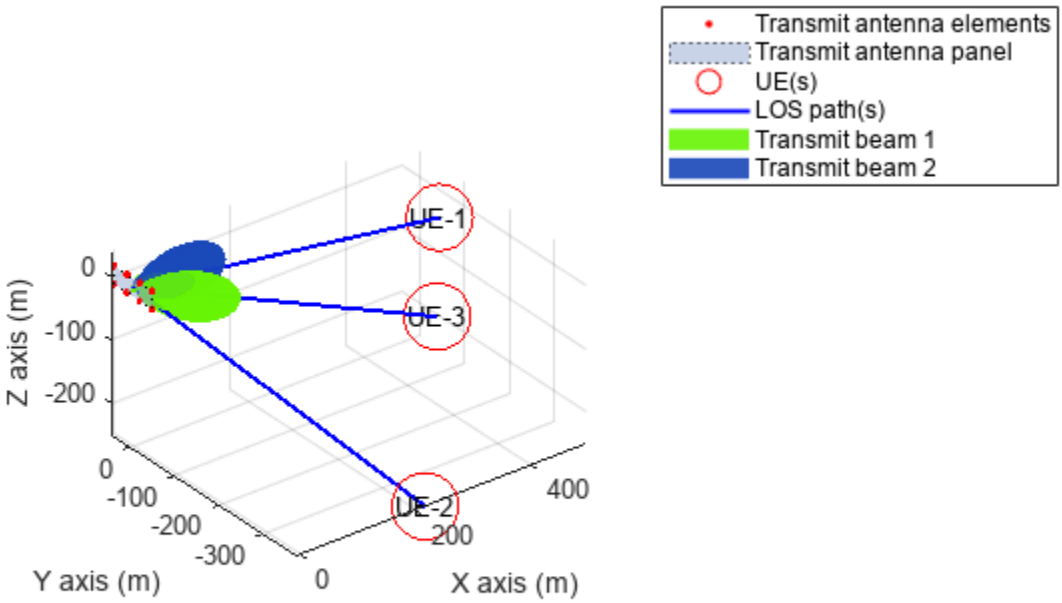

SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
Cell
Peak Data Rate
Cell-1 DL Throughput (Mbps)
Time (secs)
Cell-1 DL Resource Share (%)
Cell-1 DL Goodput (Mbps)
Cell-1 DL Buffer Status (KB)
Processing
Frames = 11 T = 0.100000


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
Downlink BLER
Cell-1 BLER
1
0.8
0.6
0.4
0.2
0
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Time (secs)
Processing
Frames = 11 T = 0.100000



Plot Tx beamforming pattern. This plot displays the selected CSI-RS beam for a UE with highest L1-RSRP. Each beam is given a unique beam index across the SSBs.

```
if enableBeamforming
    hPlotBeamformingPattern(simParameters, gNB, beamIndices);
end
```



Get the simulation metrics and save it in a MAT-file. The simulation metrics are saved in a MAT-file with the file name as `simulationMetricsFile`.

```
metrics = getMetrics(metricsVisualizer);
save(simulationMetricsFile,'metrics');
```

At the end of the simulation, the achieved value for system performance indicator is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (DL), achieved spectral efficiency (DL), and BLER observed for UEs (DL). The peak values are calculated as per 3GPP TR 37.910. The number of layers used for the peak DL data rate calculation is taken as the average value of the maximum layers possible for each UE in the respective direction. The maximum number of DL layers possible for a UE is minimum of its Rx antennas and gNB's Tx antennas.

```
displayPerformanceIndicators(metricsVisualizer);
```

```
Peak DL Throughput: 62.21 Mbps. Achieved Cell DL Throughput: 18.96 Mbps
Achieved DL Throughput for each UE: [7.98        4.63        6.35]
Achieved Cell DL Goodput: 18.96 Mbps
Achieved DL Goodput for each UE: [7.98        4.63        6.35]
Peak DL spectral efficiency: 12.44 bits/s/Hz. Achieved DL spectral efficiency for cell: 3.79 bits

Block error rate for each UE in the downlink direction: [0  0  0]
```

The simulation results show the performance metrics with DL beamforming.

Simulation Visualization

The five types of run-time visualization shown are:

- *Display of CQI values for UEs over the channel bandwidth*: For details, see the 'Channel Quality Visualization' figure description in the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.
- *Display of resource grid assignment to UEs*: The 2D time-frequency grid shows the resource allocation to the UEs. You can enable this visualization in the 'Scenario Configuration' section. For details, see the 'Resource Grid Allocation' figure description in the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.
- *Display of DL scheduling metrics plots*: For details, see 'Downlink Scheduler Performance Metrics ' figure description in the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.
- *Display of DL Block Error Rates*: For details, see the 'Block Error Rate (BLER) Visualization' figure description in the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.
- *Display of DL beamforming pattern*: The figure shows the beam with highest L1-RSRP for each UE. gNB uses this beam directions for beamforming PDSCH and CSI-RS (for RI, CQI, PMI measurements) to the UEs. Two or more UEs can report same beam as its highest L1-RSRP beam.

Results With Beamforming Disabled

These are the simulation results when DL beamforming is disabled. You can observe the network performance and CQI index for each UE.

```
Peak DL Throughput: 62.21 Mbps. Achieved Cell DL Throughput: 13.89 Mbps
Achieved DL Throughput for each UE: [6        2.96        4.93]
Achieved Cell DL Goodput: 13.84 Mbps
Achieved DL Goodput for each UE: [6        2.92        4.93]
Peak DL spectral efficiency: 12.44 bits/s/Hz. Achieved DL spectral efficiency for cell: 2.77 bits/s/Hz

Block error rate for each UE in the downlink direction: [0       0.015          0]
```

For details of simulation logs, see the 'Simulation Logs' section in the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.

```
if enableTraces
    simulationLogs = cell(1,1);
    if simParameters.DuplexMode == 0 % FDD
        logInfo = struct('DLTimeStepLogs',[],'SchedulingAssignmentLogs',[],'BLERLogs',[],'AvgBLE
        [logInfo.DLTimeStepLogs,~] = getSchedulingLogs(simSchedulingLogger);
    else % TDD
        logInfo = struct('TimeStepLogs',[],'SchedulingAssignmentLogs',[], ...
            'BLERLogs',[],'AvgBLERLogs',[]);
        logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger);
    end
    % BLER logs
    [logInfo.BLERLogs,logInfo.AvgBLERLogs] = getBLERLogs(simPhyLogger);
    % Scheduling assignments log
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger);
    simulationLogs{1} = logInfo;
    % Save simulation parameters in a MAT-file
    save(parametersLogFile,'simParameters');
    % Save simulation logs in a MAT-file
    save(simulationLogFile,'simulationLogs');
end
```

You can run the script NRPostSimVisualization to get a post-simulation visualization of logs. For more details about the options to run this script, refer to the “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.

Further Exploration

You can use this example to further explore custom scheduling.

Custom scheduling

You can modify the existing scheduling strategy to implement a custom one. “Plug In Custom Scheduler in System-Level Simulation” on page 6-118 example explains how to create a custom scheduling strategy and plug it into system-level simulation. MIMO configuration appends more fields to the scheduling assignment structure. Populate the fields of scheduling assignments with values for precoding matrix, number of layers, beamforming table index as per your custom scheduling strategy. The beamforming table index corresponds to the columns in the table 'simParameters.BeamWeightTable'. For more information about the information fields of a scheduling assignment, see the description of the `scheduleDLResourcesSlot` and `scheduleULResourcesSlot` functions in the hNRScheduler.m helper file.

Local functions

```
function ssbIndex = computeSSBToUE(simParameters,ssbTxAngles,ssbSweepRange,ueRelPosition)
%   SSBINDEX = computeSSBtoUE(SIMPARAMETERS,SSBTXANGLES,SSBSWEEPRANGE,UERELPOSITION)
%   computes an SSB to UE based on SSBTXANGLES and SSBSWEEPRANGE. If a UE
%   falls within the sweep range of an SSB then the corresponding SSBINDEX
%   is assigned. If a UE does not fall under any of the SSBs then the
%   function approximates UE to its nearest SSB.
ssbIndex = zeros(1,simParameters.NumUEs);
for i=1:simParameters.NumUEs
    if (ueRelPosition(i,2) > (ssbTxAngles(1) - ssbSweepRange/2)) && ...
            (ueRelPosition(i,2) <= (ssbTxAngles(1) + ssbSweepRange/2))
        ssbIndex(i) = 1;
    elseif (ueRelPosition(i,2) > (ssbTxAngles(2) - ssbSweepRange/2)) && ...
            (ueRelPosition(i,2) <= (ssbTxAngles(2) + ssbSweepRange/2))
        ssbIndex(i) = 2;
    elseif (ueRelPosition(i,2) > (ssbTxAngles(3) - ssbSweepRange/2)) && ...
            (ueRelPosition(i,2) <= (ssbTxAngles(3) + ssbSweepRange/2))
        ssbIndex(i) = 3;
    elseif (ueRelPosition(i,2) > (ssbTxAngles(4) - ssbSweepRange/2)) && ...
            (ueRelPosition(i,2) <= (ssbTxAngles(4) + ssbSweepRange/2))
        ssbIndex(i) = 4;
    else % Approximate UE to the nearest SSB
        % Initialize an array to store the difference between the UE azimuth and
        % SSB azimuth beam sweep angles
        angleDiff = zeros(1,length(ssbTxAngles));
        for angleIdx = 1: length(ssbTxAngles)
            angleDiff(angleIdx) = abs(ssbTxAngles(angleIdx) - ueRelPosition(i,2));
        end
        % Minimum azimuth angle difference
        [~,idx] = min(angleDiff);
        ssbIndex(i) = idx;
    end
end
end
```

References

[1] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TR 38.901. “Study on channel model for frequencies from 0.5 to 100 GHz.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “NR Downlink Transmit-End Beam Refinement Using CSI-RS” on page 1-113
- “5G NR Downlink CSI Reporting” on page 5-47
- “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87
- “NR FDD Scheduling Performance Evaluation” on page 6-56

NR Interference Modeling with Toroidal Wrap-Around

This example shows how to model a 19-site cluster with toroidal wrap-around, as described in ITU-R M.2101-0. This example uses the system-level channel model that is specified in 3GPP TR 38.901. The wrap-around provides uniform interference at the cluster edge. All the cells in the cluster operate in the same frequency band with the serving gNB at the center of the cell. You can enable or disable the wrap-around to observe that with wrap-around, the performance metrics of an edge cell become similar to the center cell.

Introduction

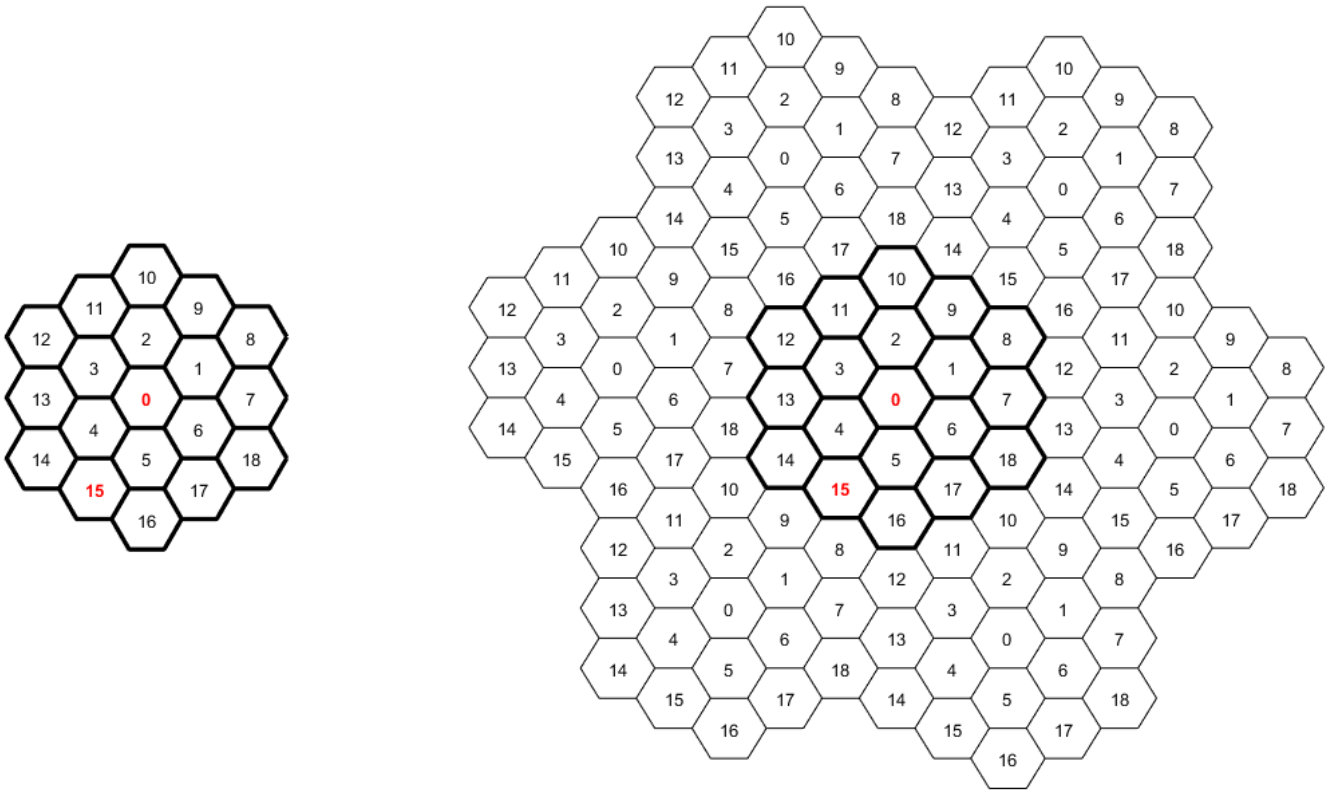
This example models:

- A 19-site cluster with three cells/site, giving a total of 57 cells. A site is represented by 3 colocated gNBs with directional antennas covering 120 degrees area (that is, 3 sectors per site).
- Co-channel intercell interference with wrap-around modeling for removing edge effects.
- System-level channel model based on 3GPP TR 38.901.
- Downlink shared channel (DL-SCH) data transmission and reception.
- DL channel quality measurement by UEs, based on the CSI-RS received from the gNB.
- Uplink shared channel (UL-SCH) data transmission and reception.
- UL channel quality measurement by gNBs, based on the SRS received from the UEs.

Nodes send the control packets (buffer status report (BSR), DL assignment, UL grants, PDSCH feedback, and CSI report) out of band, without the need of resources for transmission and assured error-free reception.

Toroidal Wrap-around Modeling

To simulate the behavior of a cellular network without introducing edge effects, this example models an infinite cellular network by using toroidal wrap-around. The entire network region relevant for simulations is a cluster of 19 sites (shown bold in this figure). The left-hand figure shows the network region of 19 sites without wrap-around. Site 0 of the central cluster, shown in red, is uniformly surrounded and experiences interference from all sides. A cell in an edge site like site 15 experiences less interference. In the right-hand figure, the wrap-around repeats the original cluster six times to uniformly surround the central cluster.



In the wrap-around model, the signal or interference from any UE to a cell is treated as if that UE is in the original cell cluster and the gNB in any of the seven clusters as specified in ITU-R M.2101-0. The distances used to compute the path loss from a transmitter node at (a, b) to a receiver node at (x, y) is the minimum of these seven distances.

- Distance between (x, y) and (a, b)
- Distance between (x, y) and $(a - \sqrt{3}D, b - 4D)$, where D is the distance between two adjacent gNBs (inter-site distance)
- Distance between (x, y) and $(a + \sqrt{3}D, b + 4D)$
- Distance between (x, y) and $\left(a - \frac{3\sqrt{3}}{2}D, b + \frac{7}{2}D\right)$
- Distance between (x, y) and $\left(a + \frac{3\sqrt{3}}{2}D, b - \frac{7}{2}D\right)$
- Distance between (x, y) and $\left(a - \frac{5\sqrt{3}}{2}D, b - \frac{1}{2}D\right)$
- Distance between (x, y) and $\left(a + \frac{5\sqrt{3}}{2}D, b + \frac{1}{2}D\right)$

These equations are derived from the equations in ITU-R M.2101-0 Attachment 2 to Annex 1. In 3GPP TR 38.901 and in the figure above, the rings of 6 and 12 sites around the central site are orientated differently from the orientations in ITU-R M.2101-0, so modified equations are required.

If you disable wrap-around, then the distance between nodes is the Euclidean distance.

Scenario Configuration

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Create a wireless network simulator.

```
rng('default');         % Reset the random number generator
numFrameSimulation = 1; % Simulation time in terms of number of 10 ms frames
networkSimulator = wirelessNetworkSimulator.init();
```

Create cell sites, sectors, and UEs in each cell for the urban macro (UMa) scenario.

```
numCellSites = 19;  % Number of cell sites
isd          = 500;                    % Inter-site distance in meters

numSectors   = 3;      % Number of sectors for each cell site
numUEs       = 2;                      % Number of UEs to drop per cell

wrapping     = true;      % Enable toroidal wrap-around modeling of cells. Set i

% Create the scenario builder
scenario = h38901Scenario(Scenario="UMa",FullBufferTraffic="on",NumCellSites=numCellSites,InterSi

% Build the scenario, adding GNB and UE nodes to the simulator
configureSimulator(scenario,networkSimulator);
```

Create the system-level channel model for the scenario, and connect it to the simulator.

```
% Create the channel model
channel = h38901Channel(Scenario="UMa");

% Connect the simulator and channel model
connect(channel,networkSimulator,scenario.CellSites);
```

Logging and Visualization Configuration

Set the `enableTraces` to `true` to log the traces. If the `enableTraces` is set to `false`, then the simulation does not log traces. To speed up the simulation, set it to `false`.

```
enableTraces = true;
```

Set up scheduling logger and phy logger for the cells of interest.

```
% Select the cells (that is, the sites and sectors) for which traces and metrics have to be colle
siteOfInterest = [0 15];
sectorOfInterest = [2 2];

numCellsOfInterest = length(siteOfInterest);

if enableTraces
```

```
    simSchedulingLogger = cell(numCellsOfInterest,1);
    simPhyLogger = cell(numCellsOfInterest,1);

    for cellIdx = 1:numCellsOfInterest

        % Get the gNB and UEs for the cell of interest from the scenario object
        [gNB,UEs] = getNodesForCell(scenario,siteOfInterest,sectorOfInterest,cellIdx);

        % Create an object for scheduler traces logging
        simSchedulingLogger{cellIdx} = helperNRSchedulingLogger(numFrameSimulation,gNB,UEs);

        % Create an object for PHY traces logging
        simPhyLogger{cellIdx} = helperNRPhyLogger(numFrameSimulation,gNB,UEs);

    end

end
```

The example updates the metrics plots periodically. Set the number of updates during the simulation.

```
numMetricsSteps = 5;
```

Set up metric visualizers.

```
metricsVisualizer = cell(numCellsOfInterest,1);

for cellIdx = 1:numCellsOfInterest

    % Get the gNB and UEs for the cell of interest from the scenario object
    [gNB,UEs] = getNodesForCell(scenario,siteOfInterest,sectorOfInterest,cellIdx);

    % Create visualization object for MAC and PHY metrics
    metricsVisualizer{cellIdx} = helperNRMetricsVisualizer(gNB,UEs,NumMetricsSteps=numMetricsSte

end
```

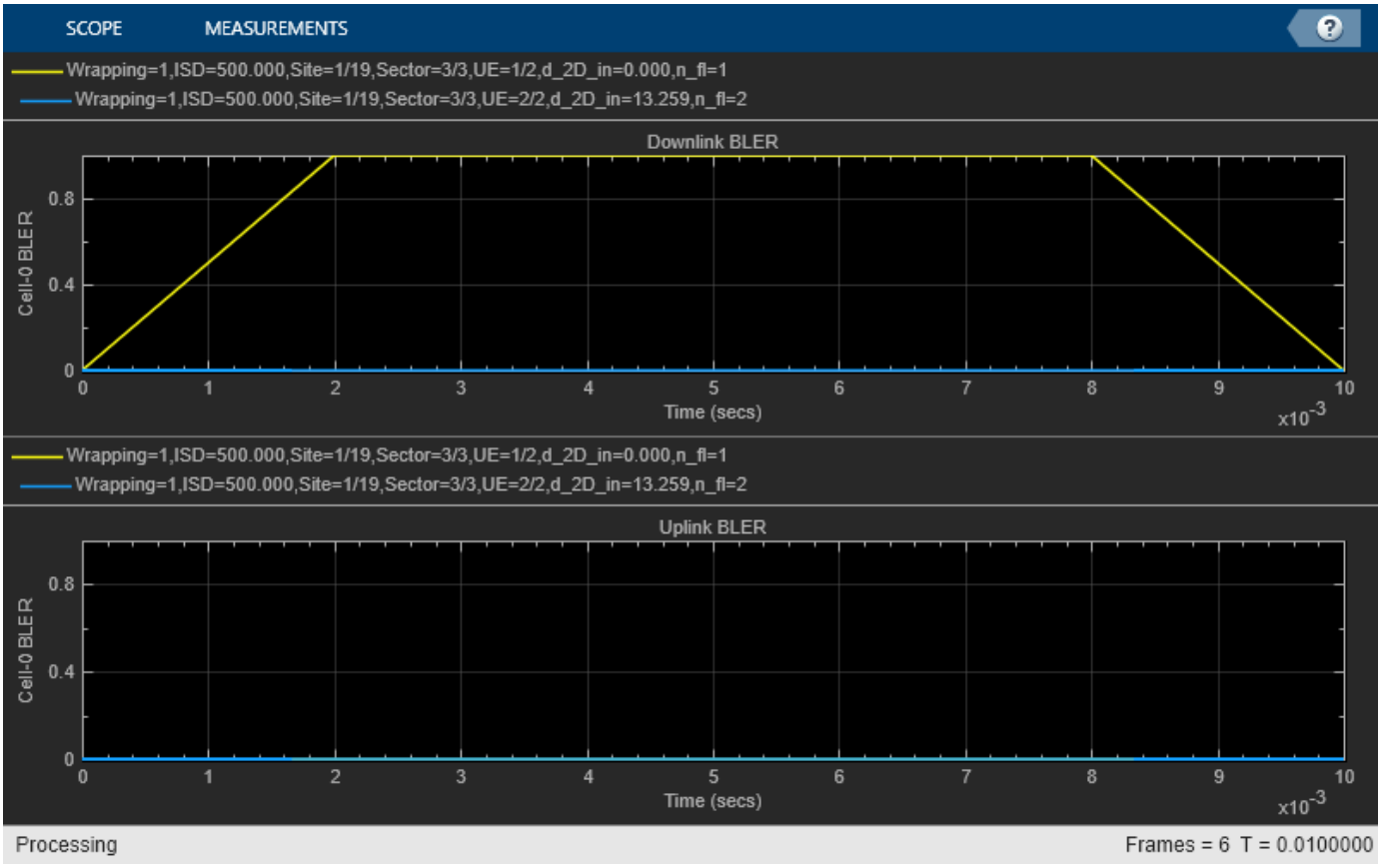
Write the logs to MAT-files. You can use these logs for post-simulation analysis.

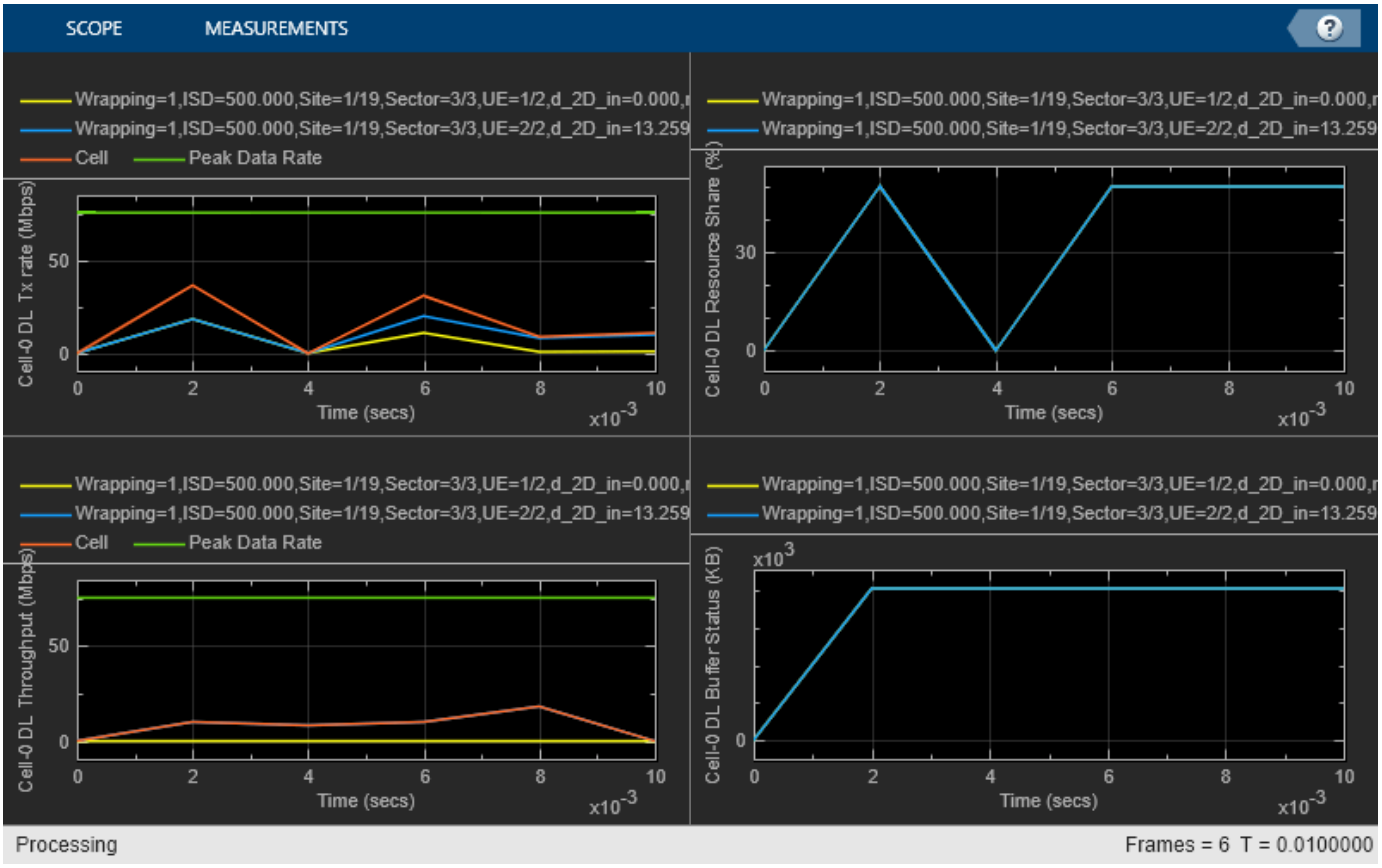
```
simulationLogFile = "simulationLogs";
```

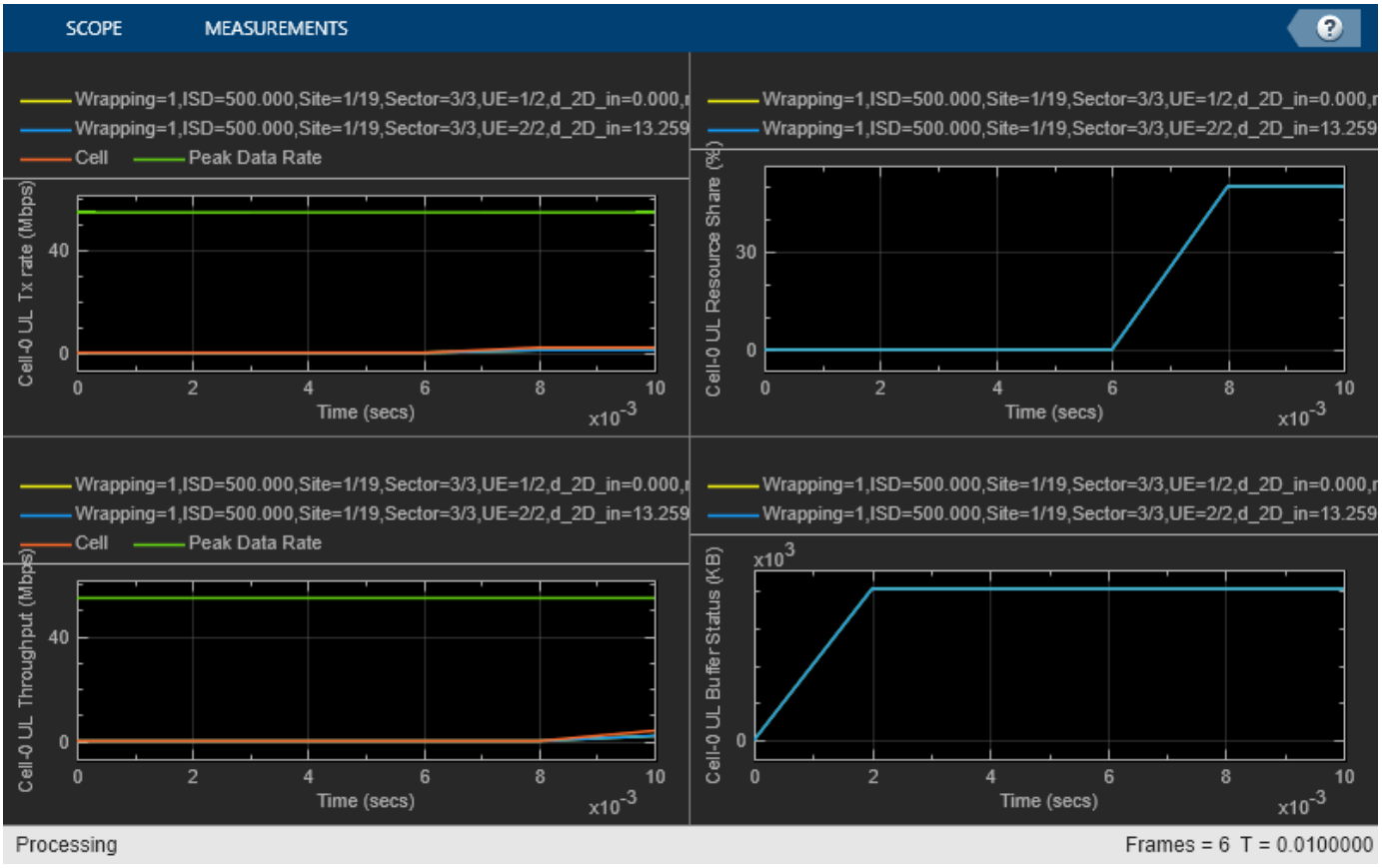
Simulation

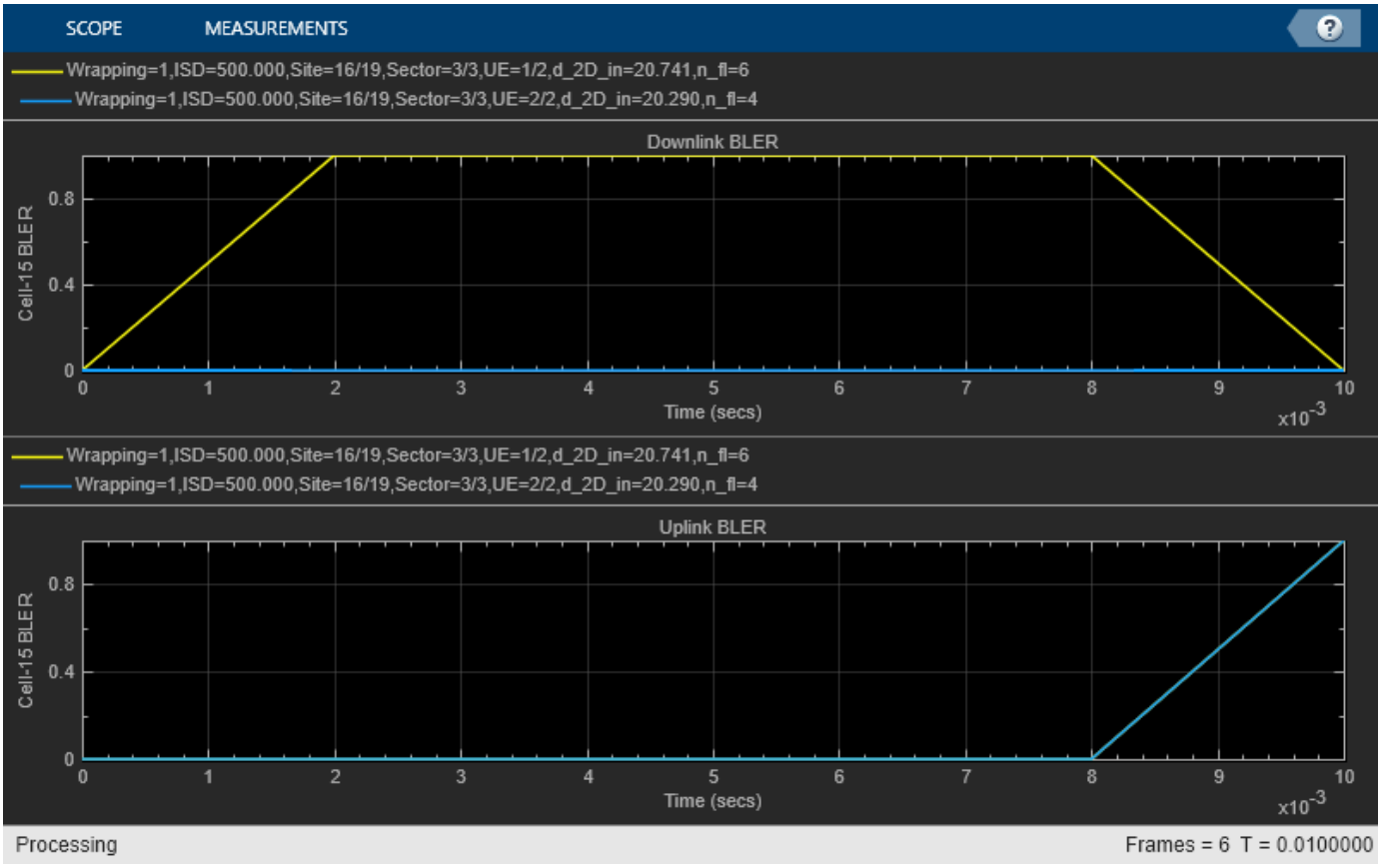
Run the simulation for the specified `numFrameSimulation` frames.

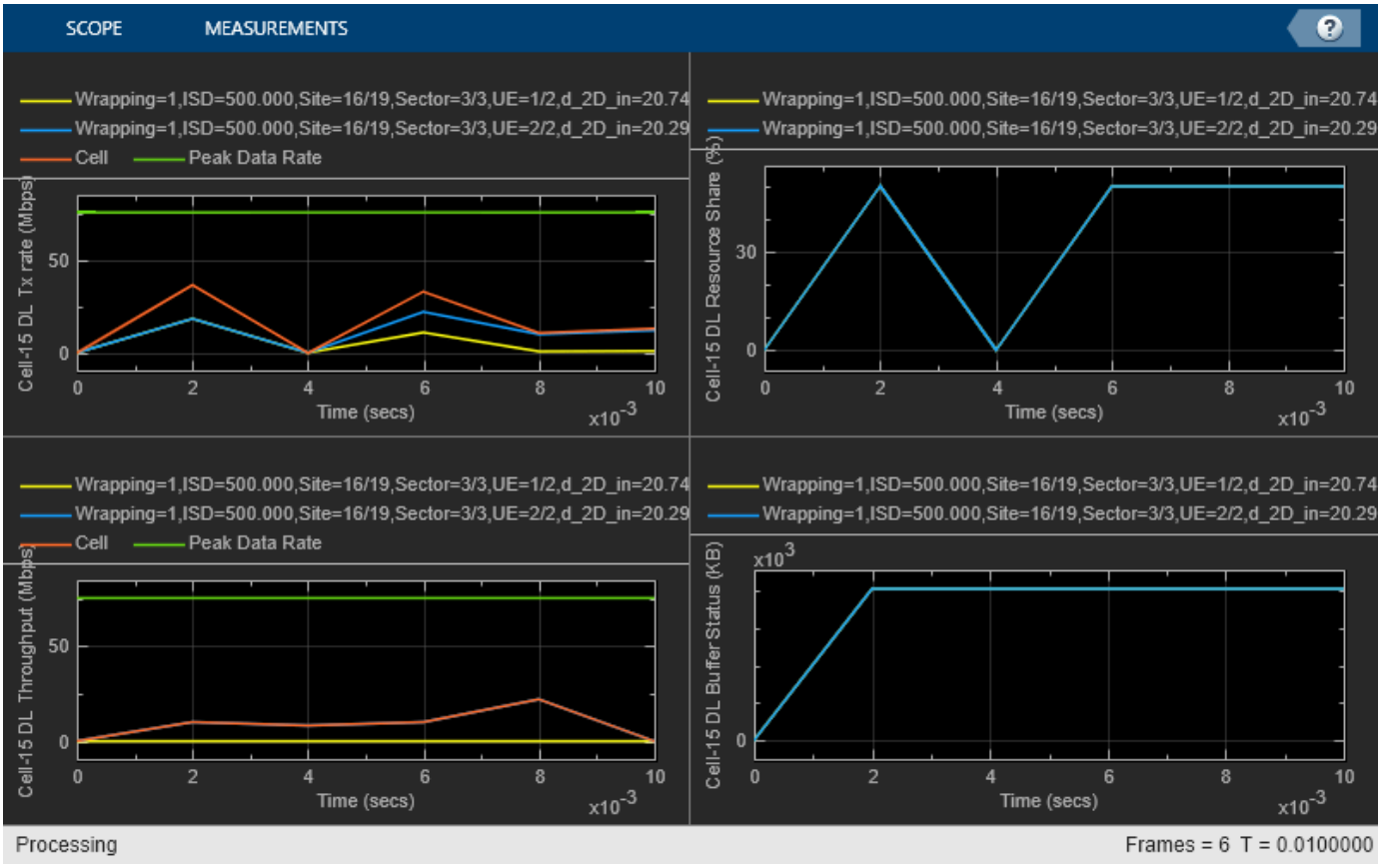
```
% Calculate the simulation duration (in seconds)
simulationTime = numFrameSimulation * 1e-2;

% Run the simulation
run(networkSimulator,simulationTime);
```

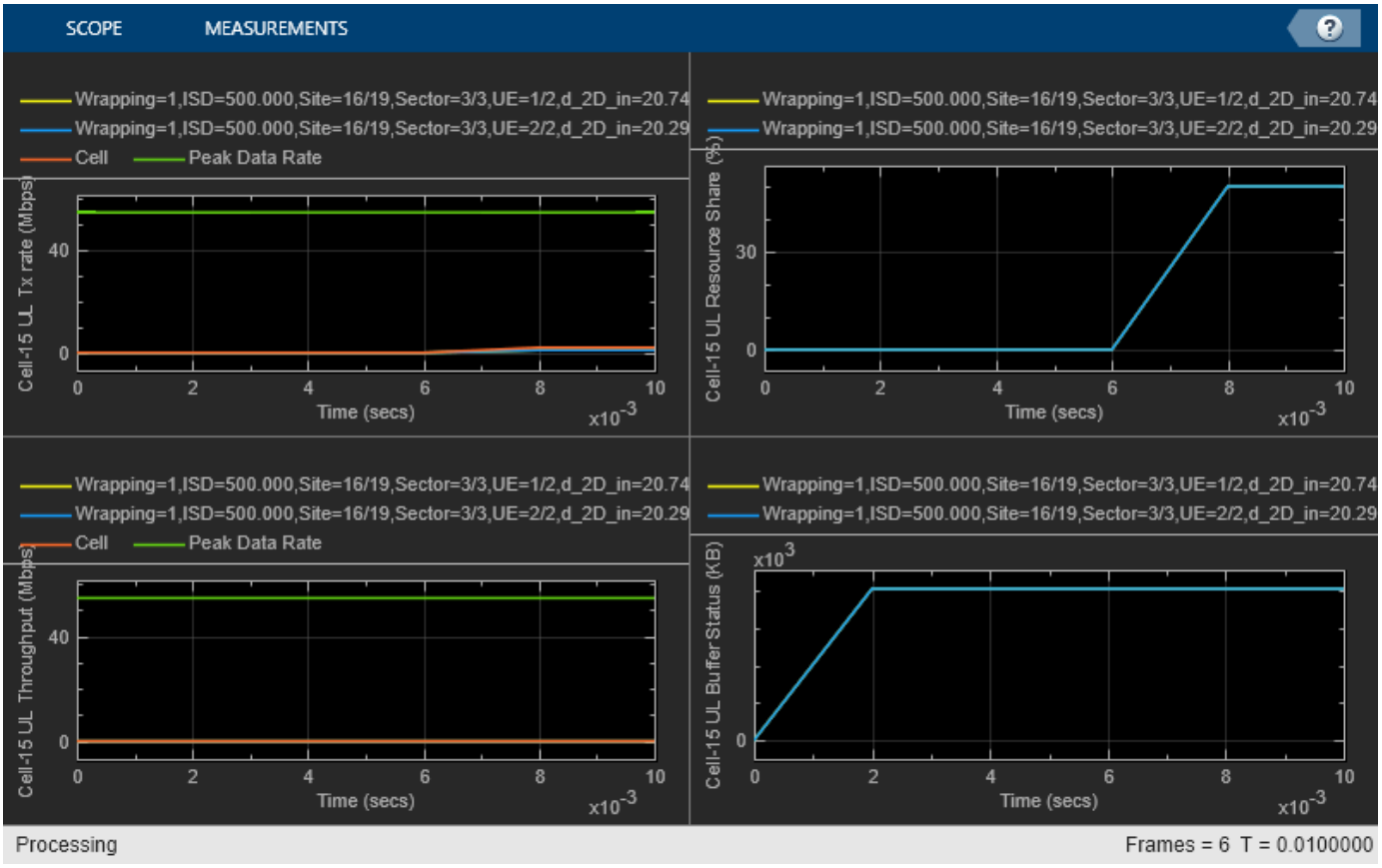

SCOPE
MEASUREMENTS
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=1/2,d_2D_in=0.000,n_fl=1
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=2/2,d_2D_in=13.259,n_fl=2
Downlink BLER
Cell-0 BLER
0.8
0.4
0
0 1 2 3 4 5 6 7 8 9 10
Time (secs)
x10^-3
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=1/2,d_2D_in=0.000,n_fl=1
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=2/2,d_2D_in=13.259,n_fl=2
Uplink BLER
Cell-0 BLER
0.8
0.4
0
0 1 2 3 4 5 6 7 8 9 10
Time (secs)
x10^-3
Processing
Frames = 6 T = 0.0100000


SCOPE
MEASUREMENTS
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=1/2,d_2D_in=0.000,
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=2/2,d_2D_in=13.259
Cell
Peak Data Rate
Cell-0 DL Tx rate (Mbps)
Time (secs)
x10^-3
Cell-0 DL Resource Share (%)
Cell-0 DL Throughput (Mbps)
Cell-0 DL Buffer Status (KB)
x10^3
Processing
Frames = 6 T = 0.0100000


SCOPE
MEASUREMENTS
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=1/2,d_2D_in=0.000,
Wrapping=1,ISD=500.000,Site=1/19,Sector=3/3,UE=2/2,d_2D_in=13.259
Cell
Peak Data Rate
Cell-0 UL Tx rate (Mbps)
Cell-0 UL Resource Share (%)
Cell-0 UL Throughput (Mbps)
Cell-0 UL Buffer Status (KB)
Time (secs)
x10^-3
x10^3
Processing
Frames = 6 T = 0.0100000


SCOPE
MEASUREMENTS
Wrapping=1,ISD=500.000,Site=16/19,Sector=3/3,UE=1/2,d_2D_in=20.741,n_fl=6
Wrapping=1,ISD=500.000,Site=16/19,Sector=3/3,UE=2/2,d_2D_in=20.290,n_fl=4
Downlink BLER
Cell-15 BLER
0.8
0.4
0
0 1 2 3 4 5 6 7 8 9 10
Time (secs)
x10-3
Wrapping=1,ISD=500.000,Site=16/19,Sector=3/3,UE=1/2,d_2D_in=20.741,n_fl=6
Wrapping=1,ISD=500.000,Site=16/19,Sector=3/3,UE=2/2,d_2D_in=20.290,n_fl=4
Uplink BLER
Cell-15 BLER
0.8
0.4
0
0 1 2 3 4 5 6 7 8 9 10
Time (secs)
x10-3
Processing
Frames = 6 T = 0.0100000


SCOPE
MEASUREMENTS
Wrapping=1,ISD=500.000,Site=16/19,Sector=3/3,UE=1/2,d_2D_in=20.74
Wrapping=1,ISD=500.000,Site=16/19,Sector=3/3,UE=2/2,d_2D_in=20.29
Cell
Peak Data Rate
Cell-15 DL Tx rate (Mbps)
Cell-15 DL Resource Share (%)
Cell-15 DL Throughput (Mbps)
Cell-15 DL Buffer Status (KB)
Time (secs)
x10^-3
x10^3
Processing
Frames = 6 T = 0.0100000



Simulation Visualization

For the cells of interest, run time visualizations show various performance indicators at multiple time steps during the simulation. For a detailed description, see the “NR Cell Performance Evaluation with MIMO” on page 6-41 example.

At the end of the simulation, the achieved values for system performance indicators are compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (UL and DL), achieved spectral efficiency (UL and DL), and block error rate (BLER) observed for UEs (UL and DL). The peak values are calculated as per 3GPP TR 37.910.

Note that to get meaningful results, the simulation has to be run for a longer duration and for a larger number of UEs per cell.

```
for cellIdx = 1:numCellsOfInterest

    fprintf('\n\nMetrics for site %d, sector %d :\n\n',siteOfInterest(cellIdx),sectorOfInterest(
    displayPerformanceIndicators(metricsVisualizer{cellIdx});

end
```

```
Metrics for site 0, sector 2 :

Peak UL Throughput: 54.64 Mbps. Achieved Cell UL Throughput: 0.81 Mbps
Achieved UL Throughput for each UE: [0.4         0.4]
Peak UL spectral efficiency: 2.73 bits/s/Hz. Achieved UL spectral efficiency for cell: 0.04 bits/
Block error rate for each UE in the uplink direction: [0  0]
```

```
Peak DL Throughput: 75.37 Mbps. Achieved Cell DL Throughput: 9.27 Mbps
Achieved DL Throughput for each UE: [0        9.27]
Peak DL spectral efficiency: 3.77 bits/s/Hz. Achieved DL spectral efficiency for cell: 0.46 bits/
Block error rate for each UE in the downlink direction: [1  0]

Metrics for site 15, sector 2 :

Peak UL Throughput: 54.64 Mbps. Achieved Cell UL Throughput: 0.00 Mbps
Achieved UL Throughput for each UE: [0  0]
Peak UL spectral efficiency: 2.73 bits/s/Hz. Achieved UL spectral efficiency for cell: 0.00 bits/
Block error rate for each UE in the uplink direction: [1  1]

Peak DL Throughput: 75.37 Mbps. Achieved Cell DL Throughput: 10.04 Mbps
Achieved DL Throughput for each UE: [0        10.04]
Peak DL spectral efficiency: 3.77 bits/s/Hz. Achieved DL spectral efficiency for cell: 0.50 bits/
Block error rate for each UE in the downlink direction: [1  0]
```

Simulation Logs

Save the simulation logs related to cells of interest into a MAT file. For a description of the simulation logs format, see the “NR Cell Performance Evaluation with MIMO” on page 6-41 example.

```
if enableTraces

    simulationLogs = cell(numCellsOfInterest,1);

    for cellIdx = 1:numCellsOfInterest

        % Get the gNB for the cell of interest from the scenario object
        gNB = getNodesForCell(scenario,siteOfInterest,sectorOfInterest,cellIdx);

        if gNB.DuplexMode == "FDD"
            logInfo = struct('DLTimeStepLogs',[],'ULTimeStepLogs',[],...
                'SchedulingAssignmentLogs',[],'PhyReceptionLogs',[]);
            [logInfo.DLTimeStepLogs,logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogg
        else % TDD
            logInfo = struct('TimeStepLogs',[],'SchedulingAssignmentLogs',[],'PhyReceptionLogs',
            logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger{cellIdx});
        end

        % Get the scheduling assignments log
        logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger{cellIdx});

        % Get the phy reception logs
        logInfo.PhyReceptionLogs = getReceptionLogs(simPhyLogger{cellIdx});

        simulationLogs{cellIdx} = logInfo;

    end

    % Save simulation logs in a MAT-file
    save(simulationLogFile,'simulationLogs');

end
```

Local Functions

```
function [gNB,UEs] = getNodesForCell(scenario,siteOfInterest,sectorOfInterest,cellIdx)

    siteIdx = siteOfInterest(cellIdx) + 1;
    sectorIdx = sectorOfInterest(cellIdx) + 1;
    numCellSites = numel(scenario.CellSites);
    if (siteIdx > numCellSites)
        warning('The cell site of interest (%d) does not exist, using the last cell site (%d).',s
        siteIdx = numCellSites;
    end
    numSectors = numel(scenario.CellSites(siteIdx).Sectors);
    if (sectorIdx > numSectors)
        warning('For cell site %d, the sector of interest (%d) does not exist, using the last sec
        sectorIdx = numSectors;
    end
    gNB = scenario.CellSites(siteIdx).Sectors(sectorIdx).BS;
    UEs = scenario.CellSites(siteIdx).Sectors(sectorIdx).UEs;

end
```

References

[1] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 3GPP TS 38.323. “NR; Packet Data Convergence Protocol (PDCP) specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[6] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[7] 3GPP TR 37.910. “Study on self evaluation towards IMT-2020 submission.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Objects

`wirelessNetworkSimulator` | `nrGNB` | `nrUE`

Related Examples

- “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87
- “NR Intercell Interference Modeling” on page 6-102

NR Cell Performance Evaluation with MIMO

This example shows how to model a 5G New Radio (NR) cell with multiple-input multiple-output (MIMO) antenna configuration and evaluate the network performance. The example performs downlink (DL) and uplink (UL) channel measurements using multi-port channel state information reference signals (CSI-RS) and sounding reference signals (SRS), respectively. The gNB uses the measured channel characteristics to make MIMO scheduling decisions.

Introduction

MIMO improves network performance by improving the cell throughput and reliability. The example performs layer mapping and precoding to utilize MIMO in the DL and UL directions.

This example models:

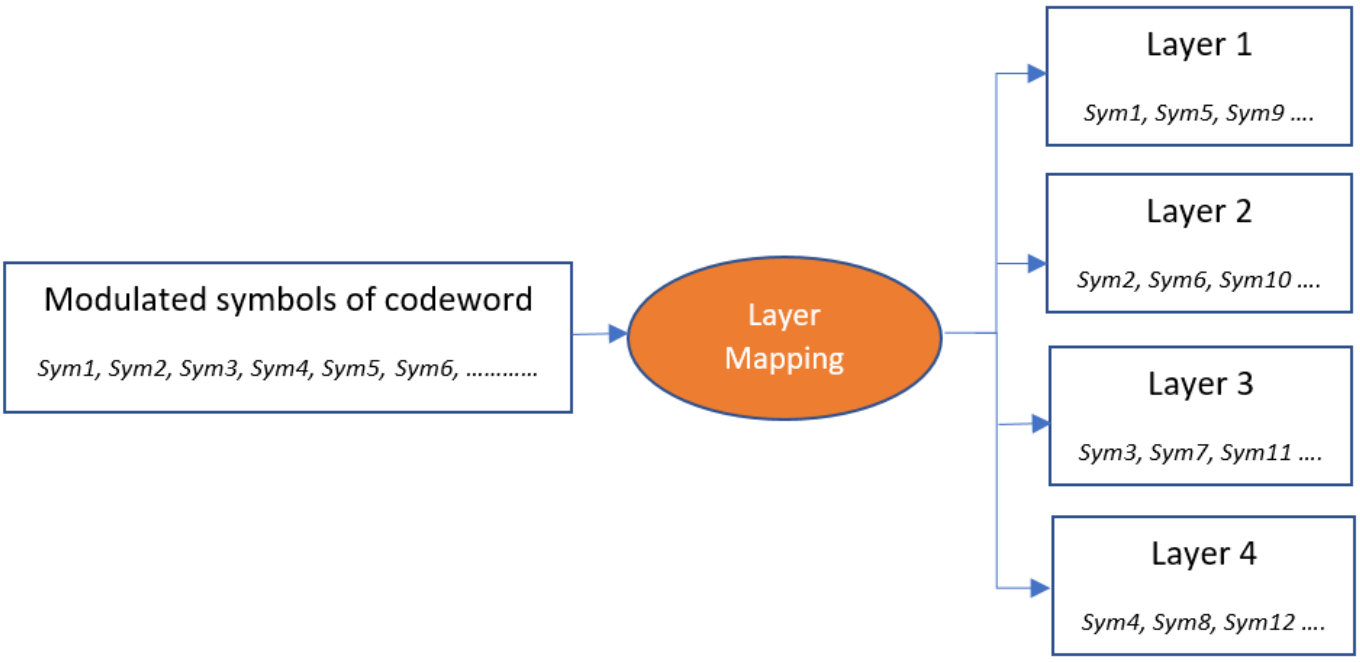
- Single-codeword DL spatial multiplexing to perform multi-layer transmission. Single-codeword limits the number of transmission layers to 4.
- Precoding to map the transmission layers to antenna ports. The example assumes one-to-one mapping from antenna ports to physical antennas.
- DL channel quality measurement by UEs based on the multi-port CSI-RS received from the gNB. All the UEs in the cell measure on same CSI-RS resource.
- UL channel quality measurement by gNB based on the multi-port SRS received from the UEs. The example does not support UL rank estimation and it is fixed to 1.
- DL rank indicator (RI), precoding matrix indicator (PMI), and channel quality indicator (CQI) reporting by UEs. The example supports Type-1 single-panel codebook for PMI.
- Free space path loss (FSPL) and clustered delay line (CDL) propagation channel model.
- Slot-based DL and UL round robin scheduling.
- Link-to-system-mapping-based abstract physical layer (PHY).

Nodes send the control packets (buffer status report (BSR), DL assignment, UL grants, PDSCH feedback, and CSI report) out of band, without the need of resources for transmission and assured error-free reception.

MIMO

The key aspects of MIMO include spatial multiplexing, precoding, channel measurement and reporting.

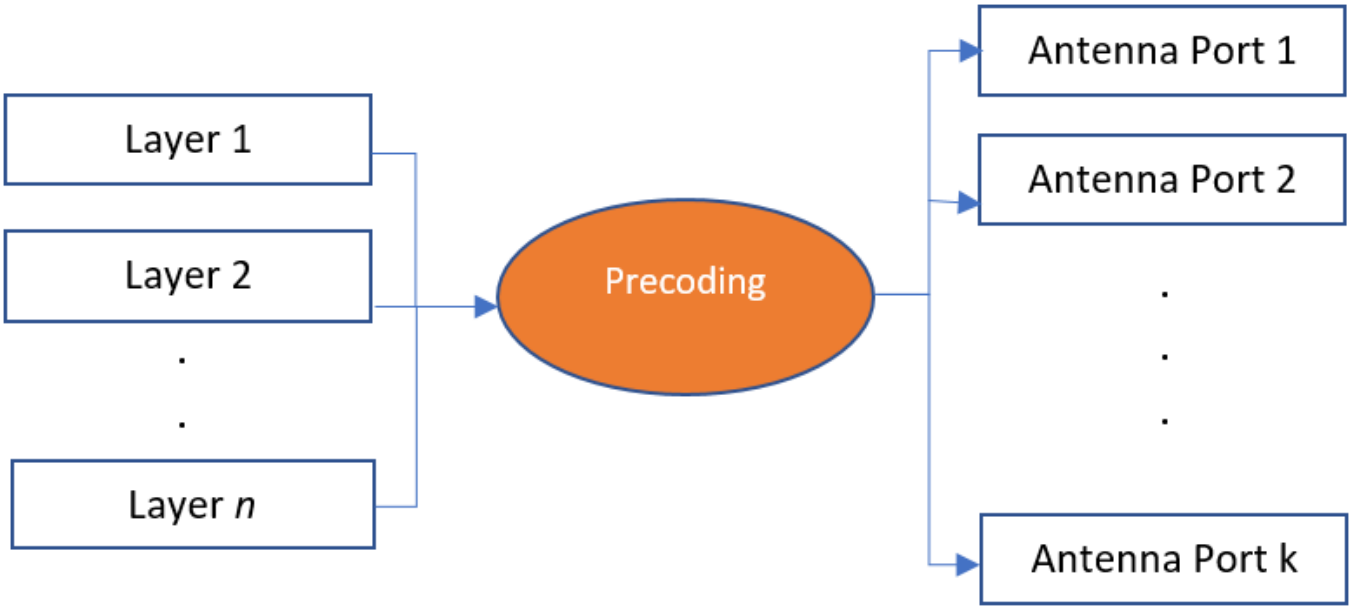
Spatial multiplexing



Spatial multiplexing utilizes MIMO to perform multi-layer transmission. The minimum of number of transmit and receive antennas limits the number of layers (or maximum rank). The layer mapping process maps the modulated symbols of the codeword onto different layers. It maps every n^{th} symbol of the codeword to n^{th} layer. For instance, this figure shows the mapping of a codeword onto four layers.

Furthermore, in the DL direction, NR specification also allows two codewords and up to a maximum of 8 transmission layers. The example currently only supports single codeword for both DL and UL.

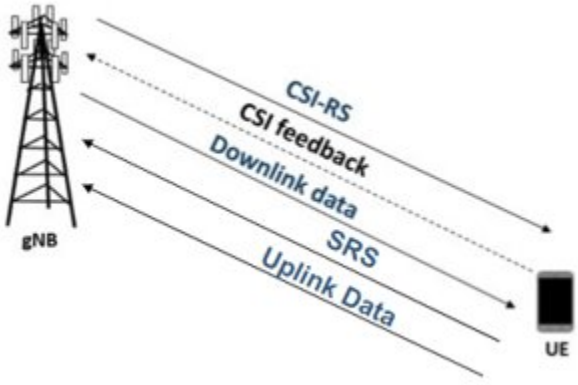
Precoding



Precoding, which follows the layer mapping, maps the transmission layers to antenna ports. Precoding applies a precoding matrix to the transmission layers and outputs data streams to the antenna ports.

Channel measurement and reporting

It consists of DL channel measurement and reporting by the UEs and UL channel measurement by gNB.



DL channel measurement and reporting

To measure the DL channel characteristics, UEs use CSI-RS. A gNB automatically configures a full bandwidth CSI-RS resource which all the UEs in the cell use. gNB sets the CSI-RS configuration such that number of CSI-RS ports are equal to number of transmit antennas on gNB. The periodicity for CSI-RS transmission is 10 slots for FDD mode. For TDD mode, the periodicity is a multiple of the length of the DL-UL pattern (in slots). In this case, the periodicity is the least value greater than or equal to 10 slots.

CSI reporting is the process by which a UE, for DL transmissions, advises a suitable number of transmission layers (rank), PMI, and CQI values to the gNB. The UE estimates these values by performing channel measurements on its configured CSI-RS resources. For more details, see the “5G NR Downlink CSI Reporting” on page 5-47 example. The gNB scheduler uses this advice to decide the number of transmission layers, precoding matrix, and modulation and coding scheme (MCS) for PDSCHs.

UL channel measurement

The UL channel measurements serve as an important input to the scheduler to decide the number of transmission layers, precoding matrix and MCS for PUSCHs.

For UL channel measurements, gNB reserves one symbol for the sounding reference signal (SRS) every 'N' slots, across the entire bandwidth. 'N' is 5 for FDD. For TDD, 'N' is the least value greater than or equal to 5 slots such that it is multiple of length of the DL-UL pattern in slots. The gNB sets the SRS configuration such that number of SRS ports are equal to number of transmit antennas on UE. The gNB configures the UEs to share the reservation of SRS by differing the comb offset, cyclic shift, or transmission time. The comb size and maximum cyclic shift are both assumed as 4. In this case, gNB can configure up to 16 connected UEs to transmit SRS with periodicity as 'N' slots. If more than 16 UEs connect, the scheduler increases the SRS transmission periodicity for each UE to the next higher value (a multiple of 'N') to accommodate more UEs.

Scenario simulation

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Create a wireless network simulator.

```
rng("default") % Reset the random number generator
numFrameSimulation = 10; % Simulation time in terms of number of 10 ms frames
networkSimulator = wirelessNetworkSimulator.init;
```

Create a gNB. Specify the carrier frequency, number of transmit antennas, number of receive antennas, and receive gain of the node.

```
gNB = nrGNB(CarrierFrequency=2.6e9,ChannelBandwidth=5e6,SubcarrierSpacing=15e3,...
    NumTransmitAntennas=16,NumReceiveAntennas=8,ReceiveGain=11);
```

Create 4 UEs. Specify the name, position, number of transmit antennas, number of receive antennas, and receive gain of each UE.

```
uePositions = [300 0 0; 700 0 0; 1200 0 0; 3000 0 0];
ueNames = "UE-" + (1:size(uePositions,1));
UEs = nrUE(Name=ueNames,Position=uePositions,NumTransmitAntennas=4,NumReceiveAntennas=2,ReceiveGa
```

Connect the UEs to gNB. Specify the RLC bearer configuration that will be used to establish RLC bearer between gNB and each UE.

```
rlcBearerConfig = nrRLCBearerConfig(SNFieldLength=6,BucketSizeDuration=10);
connectUE(gNB,UEs,RLCBearerConfig=rlcBearerConfig)
```

Set the periodic DL and UL application traffic pattern for UEs.

```
appDataRate = 40e3; % Application data rate in kilo bits per second (kbps)
for ueIdx = 1:length(UEs)
    % Install DL application traffic on gNB for the UE
    dlApp = networkTrafficOnOff(GeneratePacket=true,OnTime=numFrameSimulation*10e-3,...
        OffTime=0,DataRate=appDataRate);
    addTrafficSource(gNB,dlApp,DestinationNode=UEs(ueIdx))

    % Install UL application traffic on UE for the gNB
    ulApp = networkTrafficOnOff(GeneratePacket=true,OnTime=numFrameSimulation*10e-3,...
        OffTime=0,DataRate=appDataRate);
    addTrafficSource(UEs(ueIdx),ulApp)
end
```

Add gNB and UEs to the network simulator.

```
addNodes(networkSimulator,gNB)
addNodes(networkSimulator,UEs)
```

Create a N-by-N array of link-level channels where N is the number of nodes in the cell. An element at index (i,j) contains the channel instance from node i to node j. If element at index (i,j) is empty, it indicates that there is no channel from node i to node j. Here i and j are the node IDs.

```
channelConfig = struct("DelayProfile","CDL-C","DelaySpread",300e-9);
channels = createCDLChannels(channelConfig,gNB,UEs);
```

Create a custom channel model using `channels` and install the custom channel on the simulator. Network simulator applies the channel for a packet in transit before passing it to the receiver.

```
customChannelModel = hNRCustomChannelModel(channels);
addChannelModel(networkSimulator,@customChannelModel.applyChannelModel)
```

Set the `enableTraces` to `true` to log the traces. If the `enableTraces` is set to `false`, then traces are not logged in the simulation. To speed up the simulation, set the `enableTraces` to `false`.

```
enableTraces = true;
```

Set up scheduling logger and phy logger.

```
if enableTraces
    % Create an object for scheduler traces logging
    simSchedulingLogger = helperNRSchedulingLogger(numFrameSimulation,gNB,UEs);
    % Create an object for PHY traces logging
    simPhyLogger = helperNRPhyLogger(numFrameSimulation,gNB,UEs);
end
```

The example updates the metrics plots periodically. Set the number of updates during the simulation.

```
numMetricsSteps = 10;
```

Set up metric visualizer.

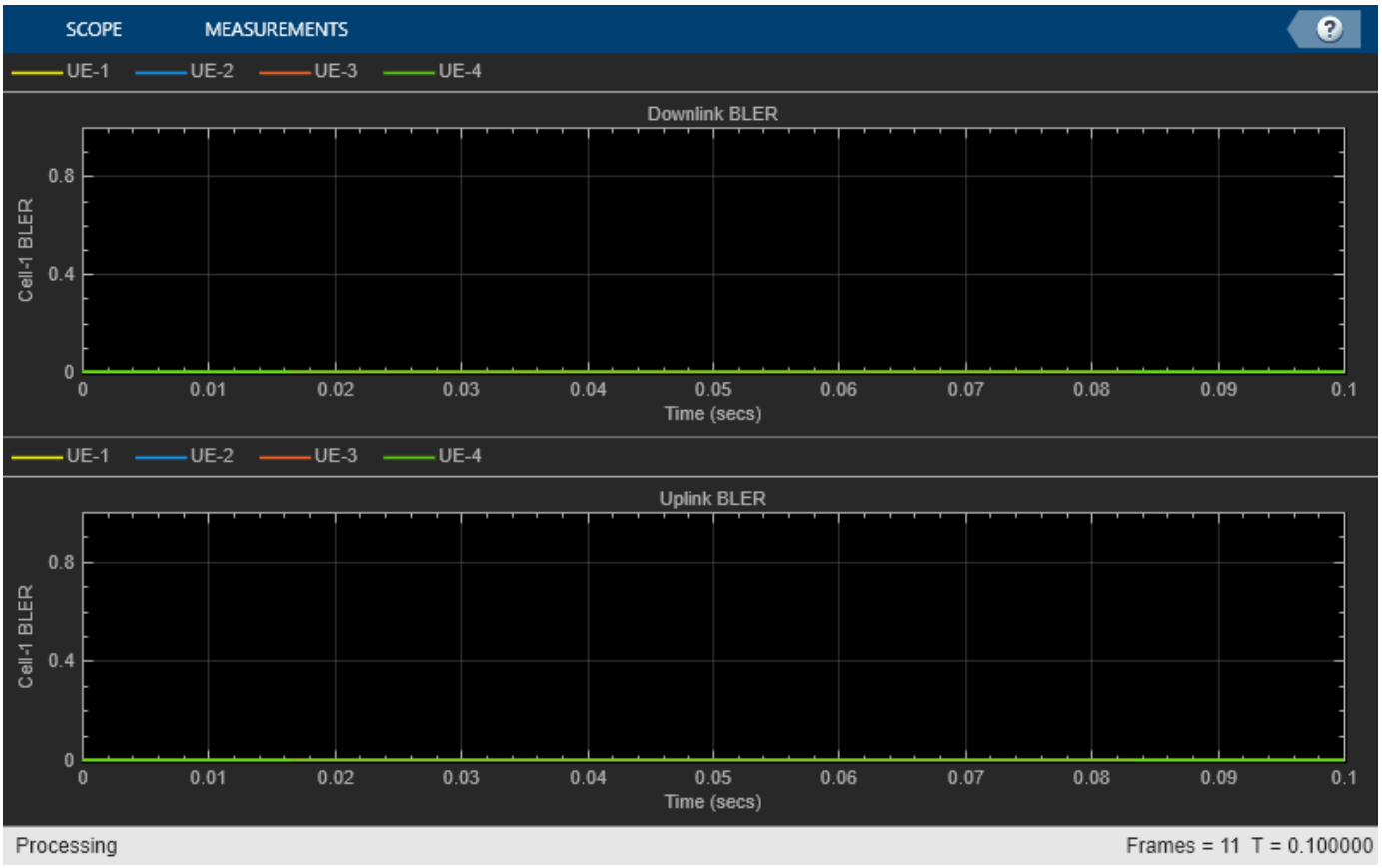
```
metricsVisualizer = helperNRMetricsVisualizer(gNB,UEs,NumMetricsSteps=numMetricsSteps,...
    PlotSchedulerMetrics=true,PlotPhyMetrics=true);
```

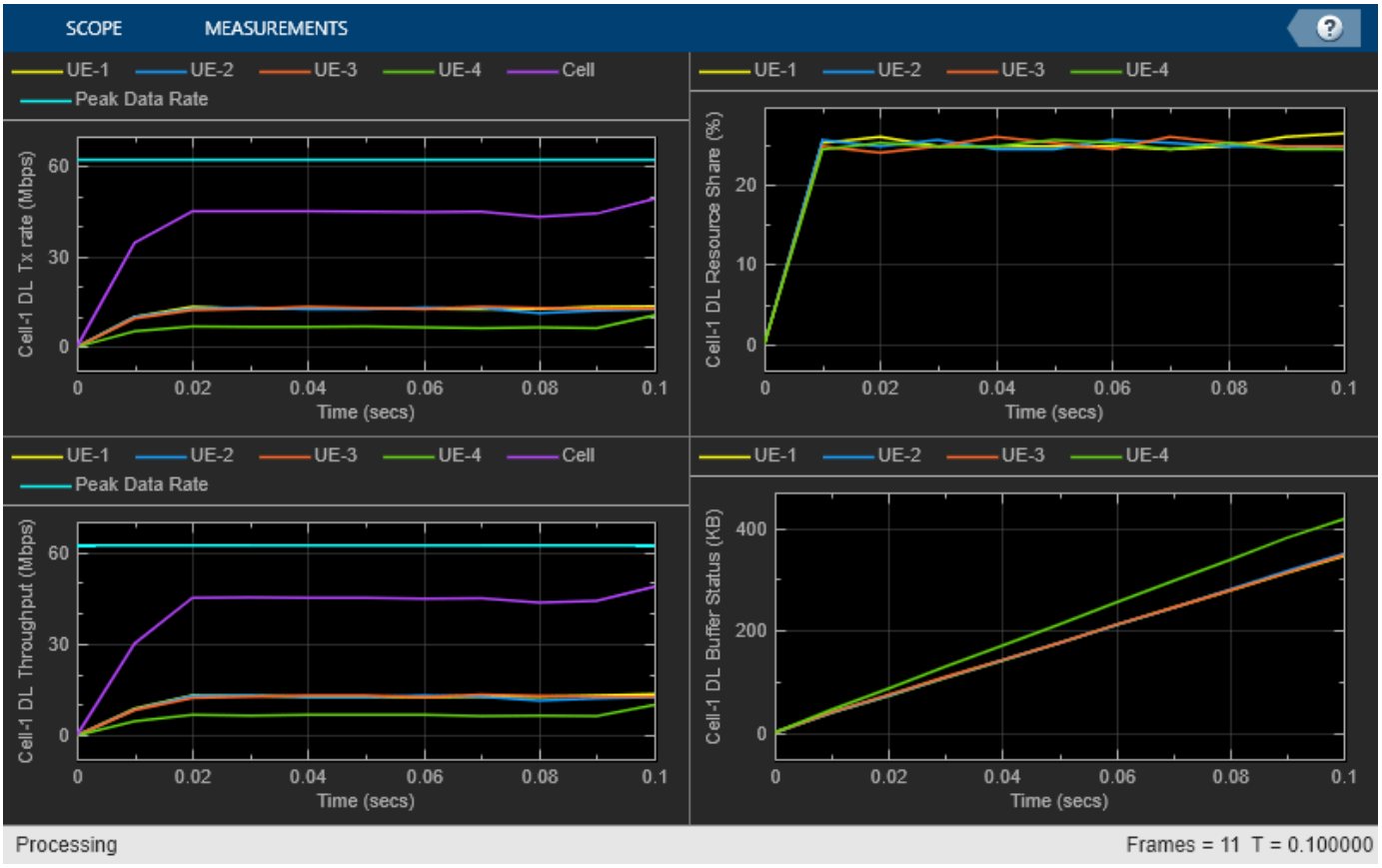
Write the logs to MAT-file. You can use these logs for post-simulation analysis.

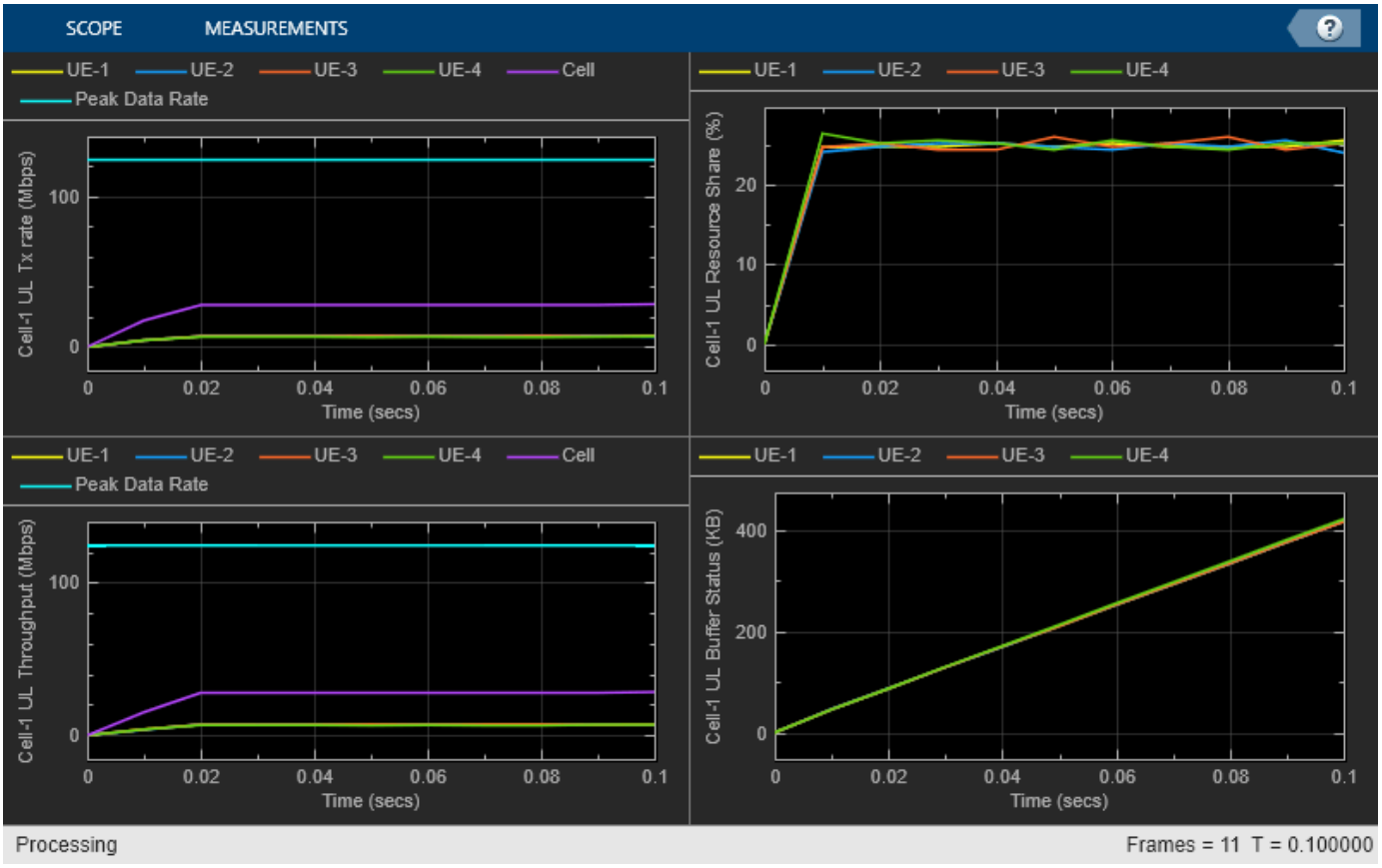
```
simulationLogFile = "simulationLogs"; % For logging the simulation traces
```

Run the simulation for the specified `numFrameSimulation` frames.

```
% Calculate the simulation duration (in seconds)
simulationTime = numFrameSimulation * 1e-2;
% Run the simulation
run(networkSimulator,simulationTime);
```




SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Cell
Peak Data Rate
Cell-1 DL Tx rate (Mbps)
Time (secs)
Cell-1 DL Resource Share (%)
Cell-1 DL Throughput (Mbps)
Cell-1 DL Buffer Status (KB)
Processing
Frames = 11 T = 0.100000



Read per-node stats.

```
gNBStats = statistics(gNB);
ueStats = statistics(UEs);
```

At the end of the simulation, the achieved value for system performance indicators is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (UL and DL), achieved spectral efficiency (UL and DL), and block error rate (BLER) observed for UEs (UL and DL). The peak values are calculated as per 3GPP TR 37.910.

```
displayPerformanceIndicators(metricsVisualizer)
```

```
Peak UL Throughput: 124.42 Mbps. Achieved Cell UL Throughput: 26.64 Mbps
Achieved UL Throughput for each UE: [6.76        6.72        6.78        6.38]
Peak UL spectral efficiency: 24.88 bits/s/Hz. Achieved UL spectral efficiency for cell: 5.33 bits
Block error rate for each UE in the uplink direction: [0  0  0  0]

Peak DL Throughput: 62.21 Mbps. Achieved Cell DL Throughput: 43.66 Mbps
Achieved DL Throughput for each UE: [12.47        12.13        12.37        6.69]
Peak DL spectral efficiency: 12.44 bits/s/Hz. Achieved DL spectral efficiency for cell: 8.73 bits
Block error rate for each UE in the downlink direction: [0  0  0  0]
```

Simulation Visualization

The run-time visualization shown are:

- *Display of UL scheduling metrics plots*: The Uplink Scheduler Performance Metrics figure includes these plots: UL transmission rate (per UE and cell), UL throughput (per UE and cell), resource share percentage among UEs (out of the total UL resources) to convey the fairness of scheduling, and pending UL buffer status of the UEs to show whether UEs are getting sufficient resources. To show the maximum achievable data rate value for UL throughput, this example uses a dashed line in the throughput and transmission rate plots. The UL transmission rate plot shows the rate of MAC data transmission for each UE and across the cell. The calculation includes transport block size (TBS) of new transmissions as well as the TBS of retransmissions. The units are in megabits per second (Mbps). The throughput plot shows the rate (in Mbps) of successful MAC data reception at gNB from each UE and across the cell.
- *Display of DL scheduling metrics plots*: The Downlink Scheduler Performance Metrics figure includes these plots: DL transmission rate (per UE and cell), DL throughput (per UE and cell), resource share percentage among UEs (out of the total DL resources) to convey the fairness of scheduling, and pending DL buffer status of the UEs to show whether UEs are getting sufficient resources. To show the maximum achievable data rate value for DL throughput, this example uses a dashed line in the throughput and transmission rate plots. The DL transmission rate plot shows the rate (in Mbps) of MAC data transmission for each UE and across the cell. The calculation includes TBS of new transmissions as well as the TBS of retransmissions. The throughput plot shows the rate (in Mbps) of successful MAC data reception at each UE and across the cell.
- *Display of DL and UL Block Error Rates*: The two sub-plots displayed in 'Block Error Rate (BLER) Visualization' shows the BLER (for each UE) observed in the uplink and downlink directions, as the simulation progresses.

Simulation Logs

The simulation logs are saved in MAT-files for post-simulation analysis. The per time step logs, scheduling assignment logs, and Phy reception logs are saved in the MAT-file `simulationLogFile`. After the simulation, open the file to load `DLTimeStepLogs`, `ULTimeStepLogs`, `SchedulingAssignmentLogs`, `PhyReceptionLogs` in the workspace.

Time step logs: Both the DL and UL time step logs follow the same format. The table shows sample time step entries.

'Timestamp'	'Frame'	'Slot'	'Frequency Allocation'	'MCS'	'HARQ Process'	'NDI'	'Tx Type'	'Channel Quality'	'HARQ NDI Status'	'Transmitted Bytes'	'Buffer Status of UEs'
0	0	0	[0,0;0,0;0,0;0,0]	[-1;-1;-1;-1]	[-1;-1;-1;-1]	[-1;-1;-1;-1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[0;0;0;0]	[3008;3008;3008;3008]
1	0	1	[0,6;6,6;12,6;18,7]	[11;11;11;11]	[0;0;0;0]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[285;285;285;333]	[8745;8745;8745;8697]
2	0	2	[0,6;6,6;12,6;18,7]	[11;11;11;11]	[1;1;1;1]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[285;285;285;333]	[12978;12978;12978;12882]
3	0	3	[0,6;6,6;12,7;19,6]	[11;11;11;11]	[2;2;2;2]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[285;285;333;285]	[17211;17211;17163;17115]
4	0	4	[0,6;6,7;13,6;19,6]	[27;27;27;15]	[3;3;3;3]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1793;1537;800]	[20192;19936;20144;20833]
5	0	5	[0,6;6,6;12,7;19,6]	[27;27;27;15]	[0;0;0;0]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1537;1793;800]	[24677;24421;24373;26055]
6	0	6	[0,6;6,6;12,7;19,6]	[27;27;27;15]	[1;1;1;1]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1537;1793;800]	[27658;27402;27099;29773]
7	0	7	[0,6;6,7;13,6;19,6]	[27;27;27;15]	[2;2;2;2]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1793;1537;800]	[30639;30128;30080;33491]
8	0	8	[0,6;6,6;12,6;18,7]	[27;27;27;15]	[3;3;3;3]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1537;1537;944]	[35124;34614;34565;38569]
9	0	9	[0,6;6,6;12,7;19,6]	[27;27;27;15]	[0;0;0;0]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1537;1793;800]	[38105;37595;37290;42288]
10	1	0	[0,6;6,7;13,6;19,6]	[27;27;27;15]	[1;1;1;1]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1793;1537;800]	[42590;41824;41775;47510]
11	1	1	[0,7;7,6;13,6;19,6]	[27;27;27;15]	[2;2;2;2]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1793;1537;1537;800]	[45316;44805;44756;51229]
12	1	2	[0,6;6,6;12,7;19,6]	[27;27;27;15]	[3;3;3;3]	[1;1;1;1]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1537;1793;800]	[48298;47786;47481;54947]
13	1	3	[0,7;7,6;13,6;19,6]	[27;27;27;15]	[0;0;0;0]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1793;1537;1537;800]	[51023;50767;50462;58660]
14	1	4	[0,7;7,6;13,6;19,6]	[27;27;27;27]	[1;1;1;1]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1793;1537;1537;768]	[55252;55252;54947;63914]
15	1	5	[0,6;6,6;12,6;18,7]	[27;27;27;27]	[2;2;2;2]	[0;0;0;0]	*4x1 cell*	*4x1 cell*	*4x16 double*	[1537;1537;1537;896]	[58233;58233;57928;67536]

Each row of the table represents a slot and contains the following information:

- *Timestamp*: Time (in milliseconds) since the start of simulation.
- *Frame*: Frame number.

- *Slot*: Slot number in the frame.
- *Frequency Allocation*: N-by-P matrix. N is the number of UEs. For frequency resource allocation type-1, P is 2. The first element of frequency resource allocation indicates the starting RB. The second element indicates the number of RBs allocated. For example, [0,6;6,6;12,6;18,7] indicates this: the scheduler has assigned 6 RBs (starting from 0) to UE-1, 6 RBs (starting from 6) to UE-2, 6 RBs (starting from 12) to UE-3, and 7 RBs (starting from 18) to UE-4. For frequency resource allocation type-0, P is the number of resource block groups (RBGs). The scheduler assigns an RBG to a particular UE by setting the corresponding bit to 1. For example, [0 0 1 1 0 1 0 1 0 1 0 0 0; 1 1 0 0 0 0 0 0 0 1 0 0; 0 0 0 0 1 0 1 0 1 0 0 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0] indicates that the bandwidth has 13 RBGs. The scheduler has assigned the RBG indices 2, 3, 5, 7, and 9 to UE-1, the RBG indices 0, 1, and 10 to UE-2, and the RBG indices 4, 6, 8, 11, and 12 to UE-3. The scheduler has not assigned any RBGs to UE-4.
- *MCS*: Row vector of length *N,* where *N* is the number of UEs. Each value corresponds to the modulation and coding scheme (MCS) index for the transmission. For example, [10 12 8 -1] indicates these: a) the scheduler has assigned the resources to UE-1, UE-2, and UE-3 for this slot and b) UE-1, UE-2 AND UE-3 use the MCS values 10, 12, and 8, respectively. The MCS index corresponds to 3GPP TS 38.214 - Table 5.1.3.1-2.
- *HARQ Process*: Row vector of length *N,* where *N* is the number of UEs. The value is the HARQ process ID used by UE for the transmission. For example, [0 3 6 -1] indicates these: a) the scheduler has assigned the resources to UE-1, UE-2, and UE-3 for this slot and b) UE-1, UE-2 AND UE-3 use the HARQ process IDs 0, 3, and 6, respectively.
- *NDI*: Row vector of length *N,* where *N* is the number of UEs. The value is the new data indicator (NDI) flag value in the assignment for transmission. For example, [0 0 1 -1] indicates these: a) The scheduler has assigned the resources to UE-1, UE-2, and UE-3 for this slot and b) UE-1, UE-2, and UE-3 use the NDI flag values 0, 0, and 1, respectively. The NDI flag determines whether the UEs have used a new transmission or a retransmission.
- *Tx Type*: Row vector of length *N,* where *N* is the number of UEs. Tx Type specifies the transmission type (new transmission or retransmission), specified as one of these values: `'newTx'`, `'reTx'`, or `'noTx'`. `'noTx'` indicates that the scheduler has not allocated the resources to the UE. For example, [`'newTx'` `'newTx'` `'reTx'` `'noTx'`] indicates that the scheduler has assigned the resources to UE-1, UE-2, and UE-3 for this slot. UE-1 and UE-2 transmit a new packet from the specified HARQ process while UE-3 retransmits the packet in the buffer of the specified HARQ process.
- *Channel Quality*: Cell array of length *N*, where *N* is the number of UEs. An element at index *i* contains the channel quality information for UE with RNTI *i*.
- *HARQ NDI Status*: *N-by-P* matrix, where *N* is the number of UEs and *P* is the number of HARQ processes on UEs. A matrix element at position *(i, j)* is the last received NDI flag at UE *i* for HARQ process ID *j*. For new transmissions, this value and the NDI flag in the assignment must toggle. For example, in slot 4 of frame 0 described in the scheduling log, the last NDI flag value for HARQ ID 3 at UE-1 is 1. To indicate a new transmission, the NDI flag value changes to 0.
- *Transmitted Bytes*: Row vector of length *N,* where *N* is the number of UEs. The values represent MAC bytes transmitted per UE in this slot. The value only includes new transmission bytes. For retransmissions, the value is zero.
- *Buffer Status of UEs*: Row vector of length *N,* where *N* is the number of UEs. The values represent the amount of pending buffer per UE.

Scheduling assignment logs: A log of all the scheduling assignments and related information. The table shows sample log entries.

'RNTI'	'Frame'	'Slot'	'Grant Type'	'Frequency Allocation'	'Start Symbol'	'Num Symbols'	'MCS'	'NumLayers'	'HARQ Process'	'NDI'	'RV'	'Tx Type'	'Feedback Slot Offset (DL grants only)'	'Channel Quality'
1	0	1	'DL'	'[0 6]'	0	14	11	1	0	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
2	0	1	'DL'	'[6 6]'	0	14	11	1	0	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
3	0	1	'DL'	'[12 6]'	0	14	11	1	0	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
4	0	1	'DL'	'[18 7]'	0	14	11	1	0	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
1	0	2	'DL'	'[0 6]'	0	14	11	1	1	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
2	0	2	'DL'	'[6 6]'	0	14	11	1	1	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
3	0	2	'DL'	'[12 6]'	0	14	11	1	1	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
4	0	2	'DL'	'[18 7]'	0	14	11	1	1	1	0	'newTx'	2	'[7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7]'
1	0	3	'DL'	'[0 6]'	0	14	11	1	2	1	0	'newTx'	2	'[15;15;15;15;15;15;15;15;15;15;15;15;15;15;15...
2	0	3	'DL'	'[6 6]'	0	14	11	1	2	1	0	'newTx'	2	'[15;15;15;15;15;15;15;15;15;15;15;15;15;15;15...
3	0	3	'DL'	'[12 7]'	0	14	11	1	2	1	0	'newTx'	2	'[15;15;15;15;15;15;15;15;15;15;15;15;15;15;15...
4	0	3	'DL'	'[19 6]'	0	14	11	1	2	1	0	'newTx'	2	'[9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9;9]'

Each row of the table represents a DL or UL allocation grant and contains these information:

- *RNTI*: Radio network temporary identifier of UE.
- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *Grant Type*: Grant type as 'DL' or 'UL'.
- *Frequency Allocation*: Row vector of length P. For frequency resource allocation type-1, P is 2. For frequency resource allocation type-0, P is the number of RBGs.
- *Start Symbol*: Start symbol for the transmission.
- *Num Symbols*: Number of symbols for the transmission.
- *MCS*: Modulation and coding scheme (MCS) index for the transmission.
- *NumLayers:* Number of spatial layers for the transmission.
- *HARQ Process*: HARQ process ID used for the transmission.
- *NDI*: NDI flag value for transmission.
- *Tx Type*: Transmission type, specified as one of these values: '`newTx`', '`reTx`'.
- *Feedback Slot Offset (DL grants only)*: Slot offset for the feedback of PDSCH transmission.
- *Channel Quality*: Row vector of length P, where *P* is the number of RBs in the bandwidth. An element at position *j* corresponds to the CQI value for UE at RB *j*.

Phy reception logs: A log of Phy packet reception information observed in the uplink and downlink directions. This table shows the sample log entries.

'Timestamp'	'Frame'	'Slot'	'Symbol'	'Number of Decode Failures(DL)'	'Number of Packets(DL)'	'Number of Decode Failures(UL)'	'Number of Packets(UL)'
2	0	2	0	[0;0;0;0]	[1;1;1;1]	[0;0;0;0]	[0;0;0;0]
2.0714	0	2	1	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.1429	0	2	2	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.2143	0	2	3	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.2857	0	2	4	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.3571	0	2	5	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.4286	0	2	6	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.5000	0	2	7	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.5714	0	2	8	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.6429	0	2	9	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.7143	0	2	10	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.7857	0	2	11	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.8571	0	2	12	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2.9286	0	2	13	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3	0	3	0	[0;0;0;0]	[1;1;1;1]	[0;0;0;0]	[1;1;1;1]
3.0714	0	3	1	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.1429	0	3	2	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.2143	0	3	3	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.2857	0	3	4	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.3571	0	3	5	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.4286	0	3	6	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.5000	0	3	7	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.5714	0	3	8	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.6429	0	3	9	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.7143	0	3	10	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.7857	0	3	11	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.8571	0	3	12	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3.9286	0	3	13	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
4	0	4	0	[0;0;0;0]	[1;1;1;1]	[0;0;0;0]	[1;1;1;1]

Each row of the table represents a slot and contains the following information:

- *Timestamp*: Time (in milliseconds) since the start of simulation.
- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *Symbol*: Symbol number in the slot.
- *Number of Decode Failures (DL)*: Row vector of length *N*, where *N* is the number of UEs. The values represent the number of decode failures at UEs in this slot.
- *Number of Packets (DL)*: Row vector of length *N*, where *N* is the number of UEs. The values represent the number of packets received at UEs in this slot.
- *Number of Decode Failures (UL)*: Row vector of length *N*, where *N* is the number of UEs. The values represent the number of decode failures at gNB for each UE in this slot.
- *Number of Packets (UL)*: Row vector of length *N*, where *N* is the number of UEs. The values represent the number of packets received at gNB for each UE in this slot.

Save the simulation logs in a MAT file.

```
if enableTraces
    simulationLogs = cell(1,1);
    if gNB.DuplexMode == "FDD"
        logInfo = struct("DLTimeStepLogs",[],"ULTimeStepLogs",[],...
            "SchedulingAssignmentLogs",[],"PhyReceptionLogs",[]);
```

```
        [logInfo.DLTimeStepLogs,logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger)
    else % TDD
        logInfo = struct("TimeStepLogs",[],"SchedulingAssignmentLogs",[],"PhyReceptionLogs",[]);
        logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger);
    end
    % Get the scheduling assignments log
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger);
    % Get the Phy reception logs
    logInfo.PhyReceptionLogs = getReceptionLogs(simPhyLogger);
    % Save simulation logs in a MAT-file
    simulationLogs{1} = logInfo;
    save(simulationLogFile,"simulationLogs")
end
```

Local functions

Set up CDL channel instances for the cell. For DL, set up a CDL channel instance from the gNB to each UE. For UL, set up a CDL channel instance from each UE to gNB.

```
function channels = createCDLChannels(channelConfig,gNB,UEs)
%createCDLChannels Create channels between gNB and UEs in a cell
%   CHANNELS = createCDLChannels(CHANNELCONFIG,GNB,UES) creates channels
%   between GNB and UES in a cell.
%
%   CHANNELS is a N-by-N array where N is the number of nodes in the cell.
%
%   CHANNLECONFIG is a struct with these fields - DelayProfile and
%   DelaySpread.
%
%   GNB is an nrGNB object.
%
%   UES is an array of nrUE objects.

numUEs = length(UEs);
numNodes = length(gNB) + numUEs;
% Create channel matrix to hold the channel objects
channels = cell(numNodes,numNodes);

% Get the sample rate of waveform
waveformInfo = nrOFDMInfo(gNB.NumResourceBlocks,gNB.SubcarrierSpacing/1e3);
sampleRate = waveformInfo.SampleRate;

for ueIdx = 1:numUEs
    % Configure the uplink channel model between gNB and UE
    channel = nrCDLChannel;
    channel.DelayProfile = channelConfig.DelayProfile;
    channel.DelaySpread = channelConfig.DelaySpread;
    channel.Seed = 73 + (ueIdx - 1);
    channel.CarrierFrequency = gNB.CarrierFrequency;
    channel = hArrayGeometry(channel, UEs(ueIdx).NumTransmitAntennas,gNB.NumReceiveAntennas,...
        "uplink");
    channel.SampleRate = sampleRate;
    channel.ChannelFiltering = false;
    channels{UEs(ueIdx).ID, gNB.ID} = channel;

    % Configure the downlink channel model between gNB and UE
    channel = nrCDLChannel;
    channel.DelayProfile = channelConfig.DelayProfile;
```

```
    channel.DelaySpread = channelConfig.DelaySpread;
    channel.Seed = 73 + (ueIdx - 1);
    channel.CarrierFrequency = gNB.CarrierFrequency;
    channel = hArrayGeometry(channel, gNB.NumTransmitAntennas,UEs(ueIdx).NumReceiveAntennas,...
        "downlink");
    channel.SampleRate = sampleRate;
    channel.ChannelFiltering = false;
    channels{gNB.ID, UEs(ueIdx).ID} = channel;
end
end
```

Appendix

The example uses these helper classes:

- helperNRMetricsVisualizer.m: Implements metrics visualization functionality
- helperNRSchedulingLogger.m: Implements scheduling information logging functionality
- helperNRPhyLogger.m: Implements Phy packet reception information logging functionality
- hNRCustomChannelModel.m: Implements channel modeling functionality
- hArrayGeometry.m: Configures antenna array geometry for CDL channel model

References

[1] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 3GPP TS 38.323. “NR; Packet Data Convergence Protocol (PDCP) specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[6] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[7] 3GPP TR 37.910. “Study on self evaluation towards IMT-2020 submission.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Objects

`wirelessNetworkSimulator` | `nrGNB` | `nrUE`

Related Examples

- “5G NR Downlink CSI Reporting” on page 5-47

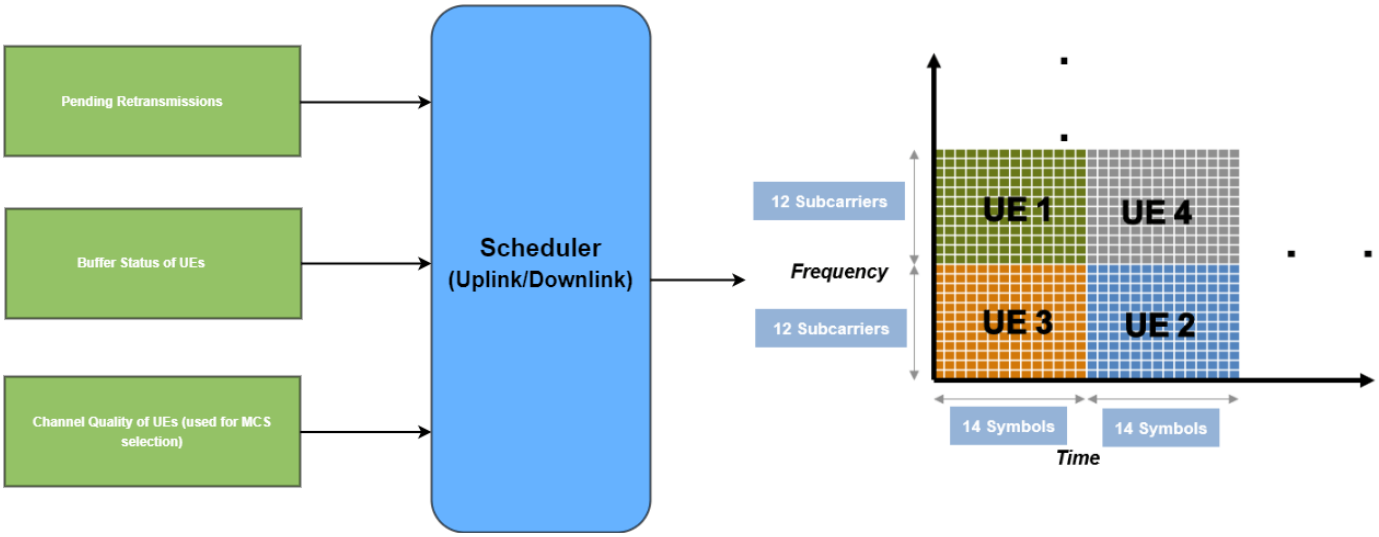
- “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87
- “NR FDD Scheduling Performance Evaluation” on page 6-56

NR FDD Scheduling Performance Evaluation

This example models scheduling of downlink (DL) and uplink (UL) resources and measures network performance in frequency division duplexing (FDD) mode. To evaluate the network performance with different data traffic patterns, the example also models the radio link control layer in unacknowledged mode (RLC-UM) with the logical channel prioritization (LCP) procedure. The example uses link-to-system-mapping-based abstract physical layer (PHY). The performance of the scheduling strategy is evaluated in terms of the achieved throughput and fairness in resource sharing.

Introduction

To assign DL and UL resources, this example uses a round robin scheduler. The scheduler takes resource allocation decisions based on these inputs: pending retransmissions, buffer status, and channel quality for the UEs.



This example models:

- Slot-based DL and UL scheduling.
- Multiple logical channels (LCHs) to support different kind of applications.
- Logical channel prioritization (LCP) to distribute the received assignment among logical channels per UE for UL and DL.

The control packets required are assumed to be sent out of band without the need of resources for transmission. The control packets are UL assignment, DL assignment, buffer status report (BSR), and PDSCH feedback.

Scenario simulation

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Create a wireless network simulator.

```
rng("default") % Reset the random number generator
numFrameSimulation = 50; % Simulation time in terms of number of 10 ms frames
networkSimulator = wirelessNetworkSimulator.init;
```

Create a gNB node. Specify the duplex mode, carrier frequency, channel bandwidth, subcarrier spacing, and receive gain of the node.

```
gNB = nrGNB(DuplexMode="FDD",CarrierFrequency=2.6e9,ChannelBandwidth=30e6,SubcarrierSpacing=15e3
```

Set the scheduler parameter `ResourceAllocationType` by using the `configureScheduler` function.

```
configureScheduler(gNB,ResourceAllocationType=0);
```

Create 4 UE nodes. Specify the name, position and receive gain of each UE node.

```
uePositions = [100 0 0; 250 0 0; 700 0 0; 750 0 0];
ueNames = "UE-" + (1:size(uePositions,1));
UEs = nrUE(Name=ueNames,Position=uePositions,ReceiveGain=11);
```

Load the application configuration table containing these fields. Each row in the table represents one application and has these properties as columns.

- DataRate - Application traffic generation rate (in kilobits per second).
- PacketSize - Size of the packet (in bytes).
- HostDevice - Defines the device (UE or gNB) on which the application is installed with the specified configuration. The device takes values 0 or 1. The values 0 and 1 indicate the configuration of application is at the gNB and the UE, respectively.
- RNTI - Radio network temporary identifier of a UE. This identifies the UE for which the application is installed.
- LogicalChannelID - Logical channel identifier.

```
load("NRFDDAppConfig.mat")
% Validate the host device type for the applications configured
validateattributes(AppConfig.HostDevice,{'numeric'},{"nonempty","integer",">=",0,"<=",1},"AppCon
    "HostDevice");
```

Load the RLC bearer configuration table. Each row in the table represents one RLC bearer and has these properties as columns.

- RNTI - Radio network temporary identifier of the UE.
- LogicalChannelID - Logical channel identifier.
- LogicalChannelGroup - Logical channel group identifier.
- SNFieldLength - Defines the sequence number field length. It takes either 6 or 12.
- BufferSize - Maximum Tx buffer size in terms of number of higher layer service data units (SDUs).
- ReassemblyTimer - Defines the reassembly timer (in ms).
- RLCEntityType - Defines the RLC entity type. It takes values "UMDL", "UMUL", and "UM", which indicates whether the RLC UM entity is unidirectional DL, unidirectional UL, or bidirectional UM, respectively.

- Priority - Priority of the logical channel.
- PrioritizedBitRate - Prioritized bit rate (in kilo bytes per second).
- BucketSizeDuration - Bucket size duration (in ms).

```
load("NRFDDRLCChannelConfig.mat")
```

Create a set of RLC bearer configuration objects.

```
rlcBearerConfig = cell(1, length(UEs));
for rlcBearerInfoIdx = 1:size(RLCChannelConfig, 1)
    rlcBearerConfigStruct = table2struct(RLCChannelConfig(rlcBearerInfoIdx, 2:end));
    ueIdx = RLCChannelConfig.RNTI(rlcBearerInfoIdx);

    % Create an RLC bearer configuration object with the specified logical
    % channel ID
    rlcBearerObj = nrRLCBearerConfig(LogicalChannelID=rlcBearerConfigStruct.LogicalChannelID);
    % Set the other properties of the configuration object with specified values
    rlcBearerObj.LogicalChannelGroup = rlcBearerConfigStruct.LogicalChannelGroup;
    rlcBearerObj.SNFieldLength = rlcBearerConfigStruct.SNFieldLength;
    rlcBearerObj.BufferSize = rlcBearerConfigStruct.BufferSize;
    rlcBearerObj.ReassemblyTimer=rlcBearerConfigStruct.ReassemblyTimer;
    rlcBearerObj.Priority=rlcBearerConfigStruct.Priority;
    rlcBearerObj.PrioritizedBitRate=rlcBearerConfigStruct.PrioritizedBitRate;
    rlcBearerObj.BucketSizeDuration=rlcBearerConfigStruct.BucketSizeDuration;
    rlcBearerObj.RLCEntityType=rlcBearerConfigStruct.RLCEntityType;

    rlcBearerConfig{ueIdx} = [rlcBearerConfig{ueIdx} rlcBearerObj];
end
```

Connect the UE nodes to gNB node. Specify the RLC bearer configuration that will be used to establish RLC bearer between gNB node and each UE node.

```
for ueIdx = 1:length(UEs)
    connectUE(gNB,UEs(ueIdx),BSRPeriodicity=5,RLCBearerConfig=rlcBearerConfig{ueIdx})
end
```

Set the periodic DL and UL application traffic pattern for UEs.

```
for appIdx = 1:size(AppConfig,1)

    % Create an object for On-Off network traffic pattern
    app = networkTrafficOnOff(PacketSize=AppConfig.PacketSize(appIdx),GeneratePacket=true, ...
            OnTime=numFrameSimulation/100,OffTime=0,DataRate=AppConfig.DataRate(appIdx));

    if AppConfig.HostDevice(appIdx) == 0
        % Add traffic pattern that generates traffic on downlink
        addTrafficSource(gNB,app,DestinationNode=UEs(AppConfig.RNTI(appIdx)),LogicalChannelID=App
    else
        % Add traffic pattern that generates traffic on uplink
        addTrafficSource(UEs(AppConfig.RNTI(appIdx)),app,LogicalChannelID=AppConfig.LogicalChanne
    end
end
```

Add gNB node and UE nodes to the network simulator.

```
addNodes(networkSimulator,gNB)
addNodes(networkSimulator,UEs)
```

Set the `enableTraces` to `true` to log the traces. If the `enableTraces` is set to `false`, then traces are not logged in the simulation. To speed up the simulation, set the `enableTraces` to `false`.

```
enableTraces = true;
```

The `cqiVisualization` and `rbVisualization` parameters control the display of the CQI visualization and the RB assignment visualization respectively. By default, these plots are enabled. You can disable them by setting the respective flags to false.

```
cqiVisualization = true;
rbVisualization = true;
```

Set up RLC logger, scheduling logger, and PHY logger.

```
if enableTraces
    % Create an object for RLC traces logging
    simRLCLogger = helperNRRLCLogger(numFrameSimulation,gNB,UEs);
    % Create an object for scheduler traces logging
    simSchedulingLogger = helperNRSchedulingLogger(numFrameSimulation,gNB,UEs);
    % Create an object for CQI and RB grid visualization
    gridVisualizer = helperNRGridVisualizer(numFrameSimulation,gNB,UEs,CQIGridVisualization=cqiV
        ResourceGridVisualization=rbVisualization,SchedulingLogger=simSchedulingLogger);
end
```

The example updates the metrics plots periodically. Set the number of updates during the simulation.

```
numMetricsSteps = 20;
```

Set up metric visualizer.

```
metricsVisualizer = helperNRMetricsVisualizer(gNB,UEs,NumMetricsSteps=numMetricsSteps, ...
    PlotSchedulerMetrics=true,PlotRLCMetrics=true);
```

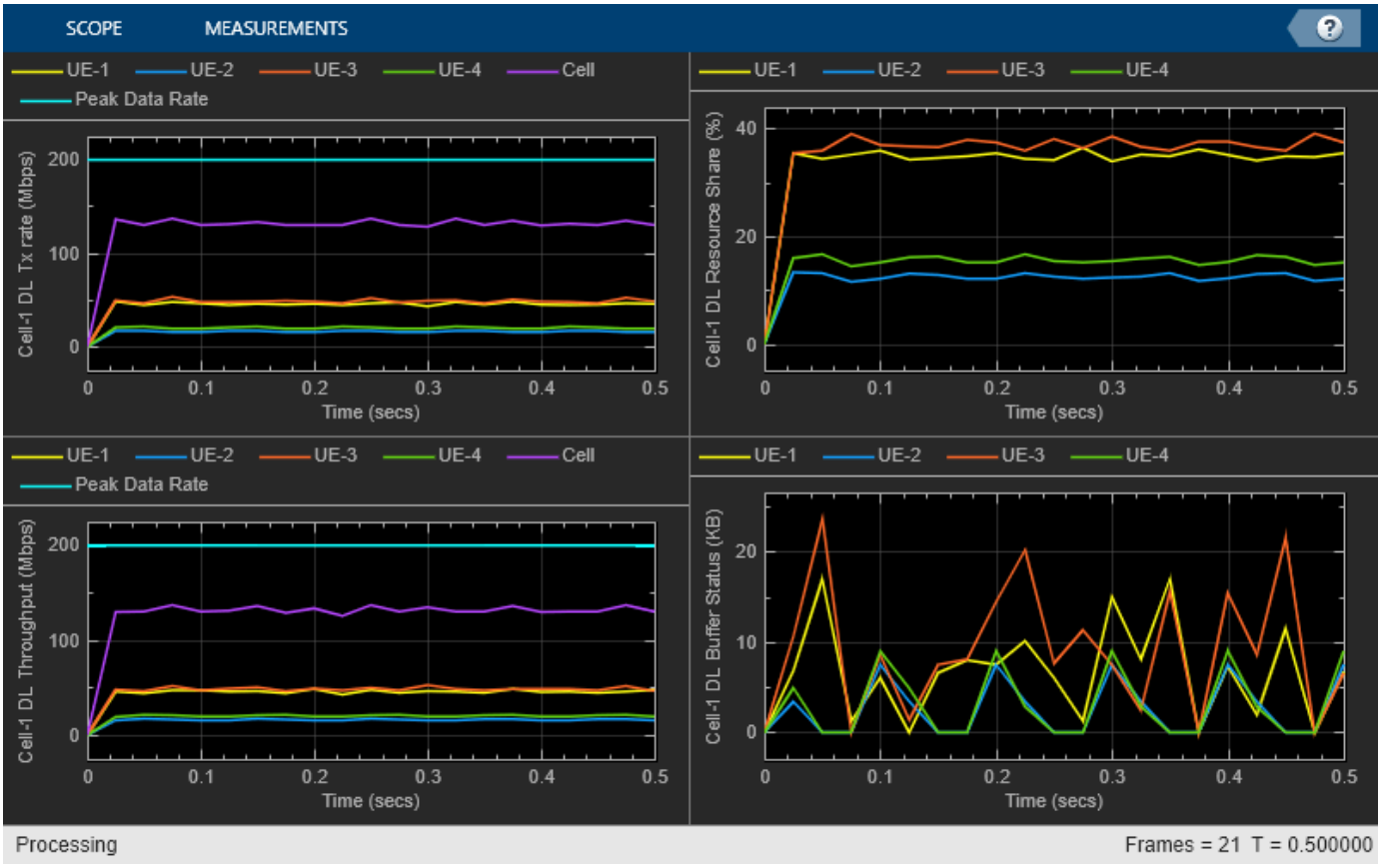
Write the logs to MAT-files. The example uses these logs for post-simulation analysis and visualization.

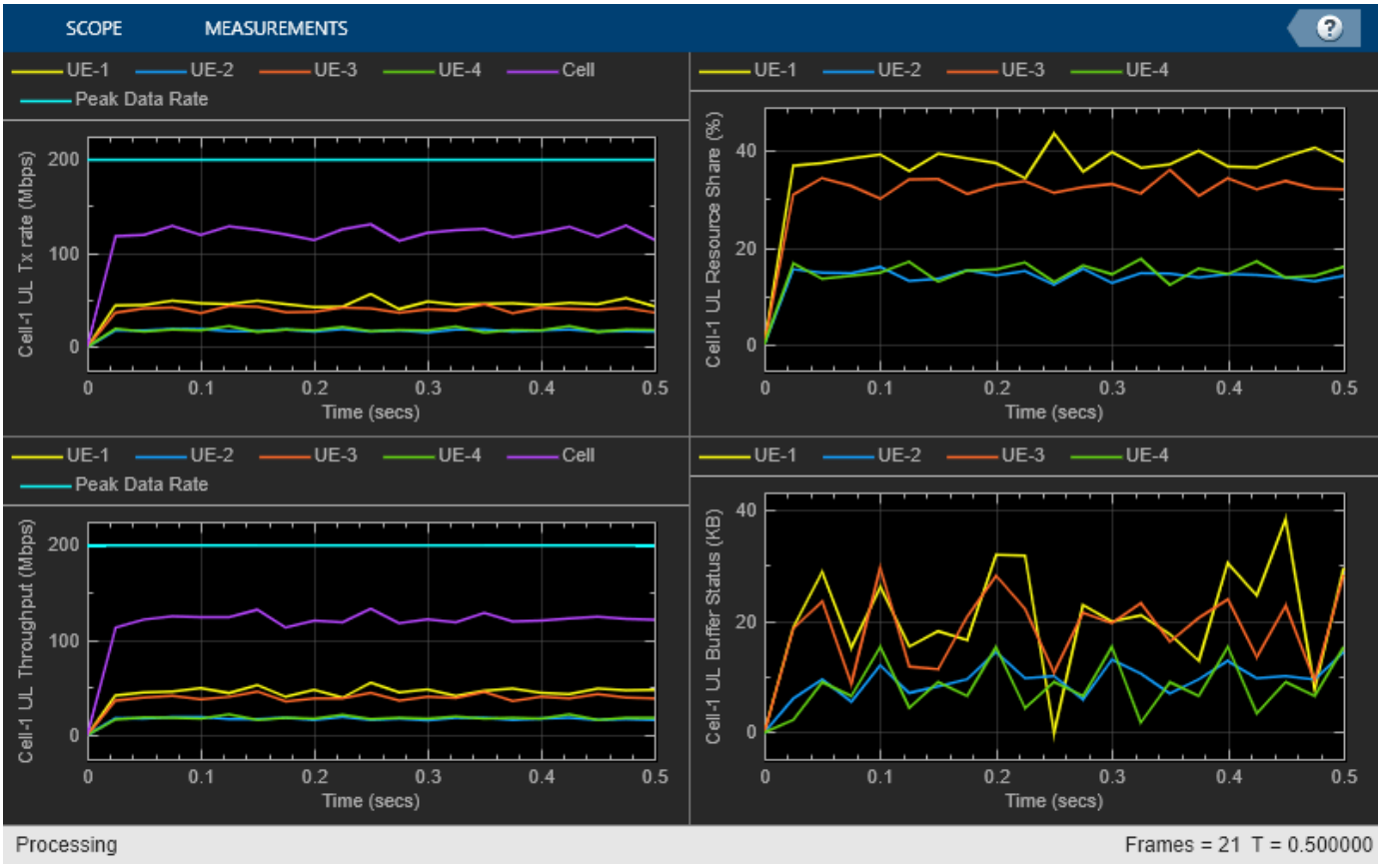
```
simulationLogFile = "simulationLogs"; % For logging the simulation traces
```

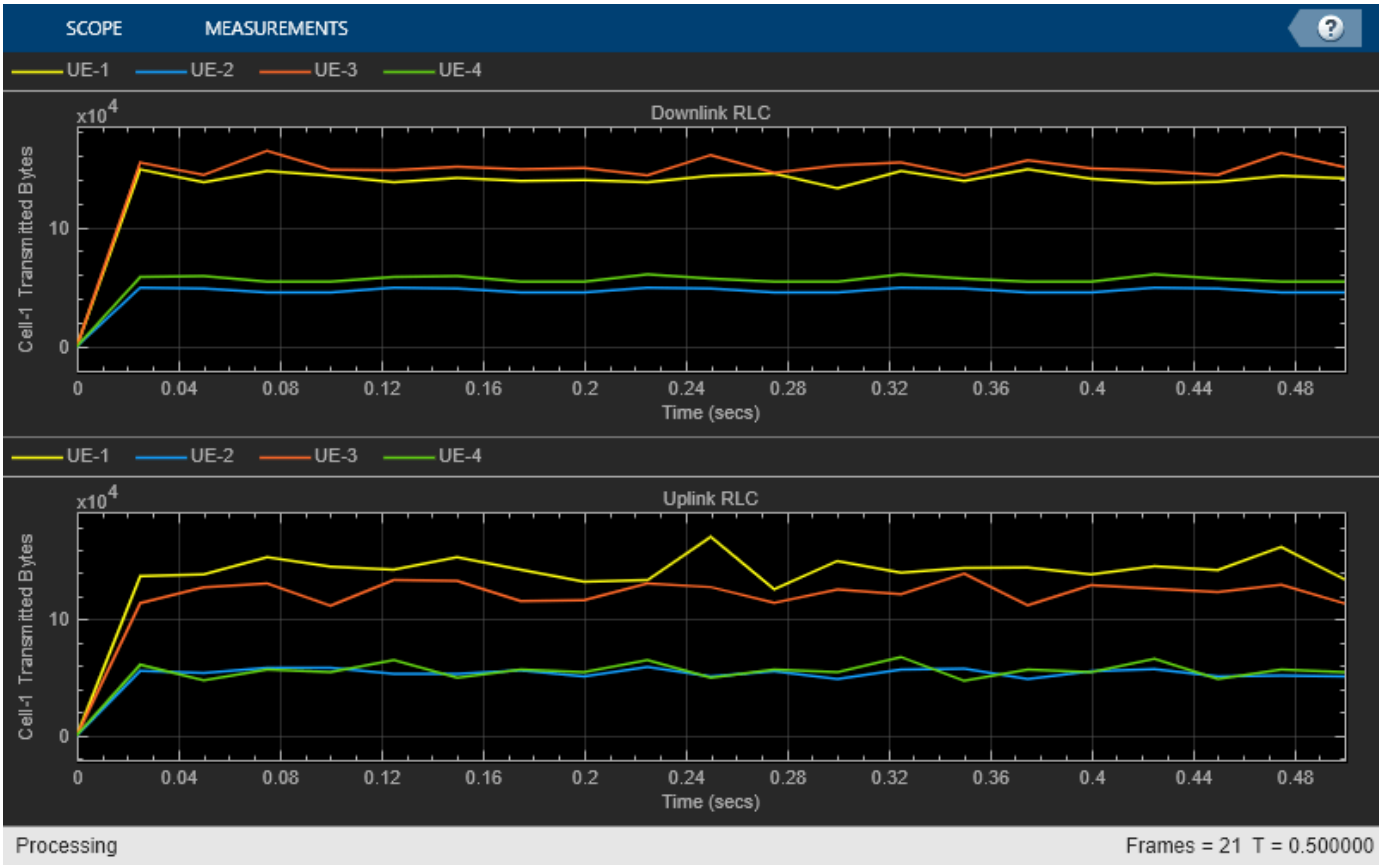
Run the simulation for the specified `numFrameSimulation` frames.

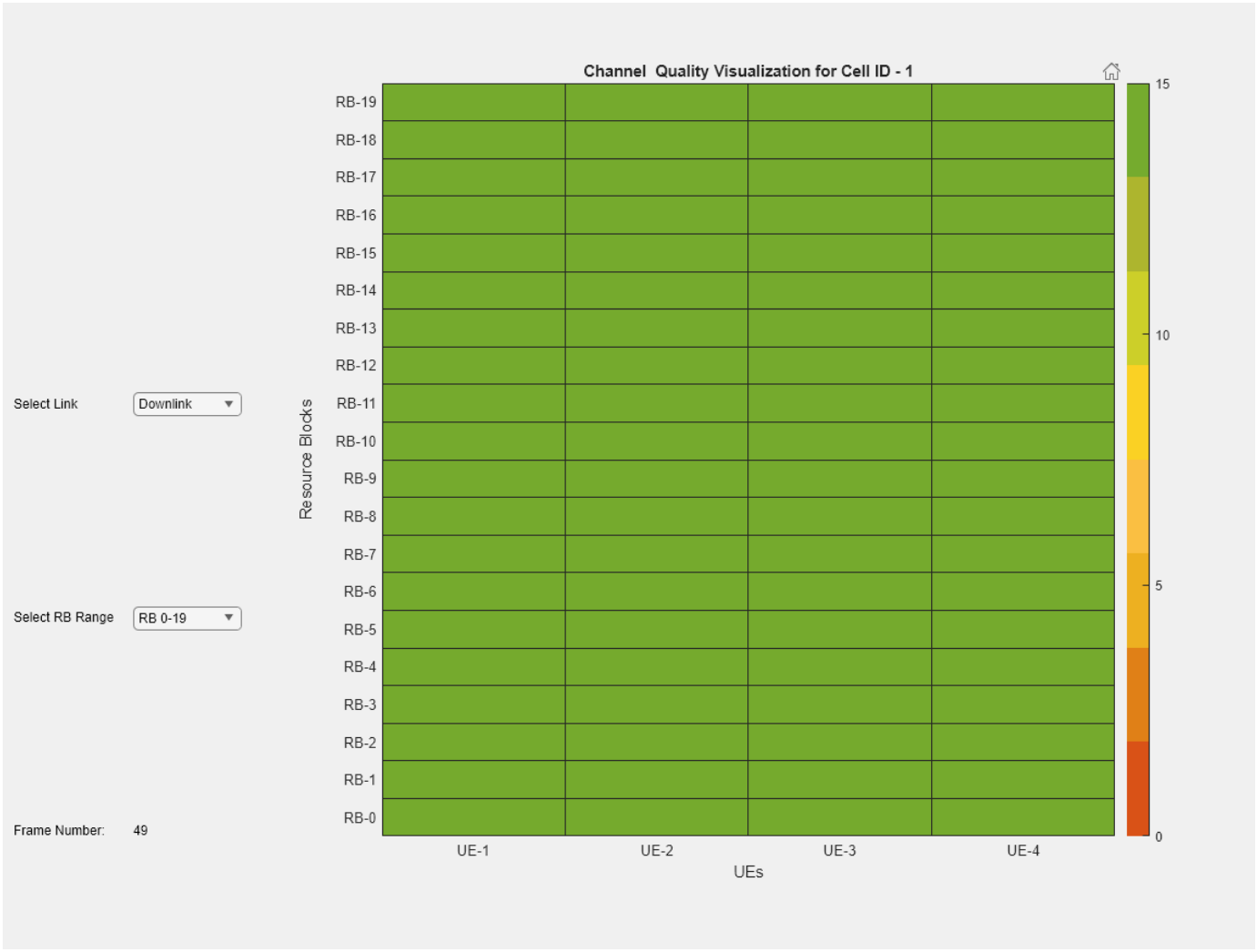
```
% Calculate the simulation duration (in seconds)
simulationTime = numFrameSimulation*1e-2;
% Run the simulation
run(networkSimulator,simulationTime)
```

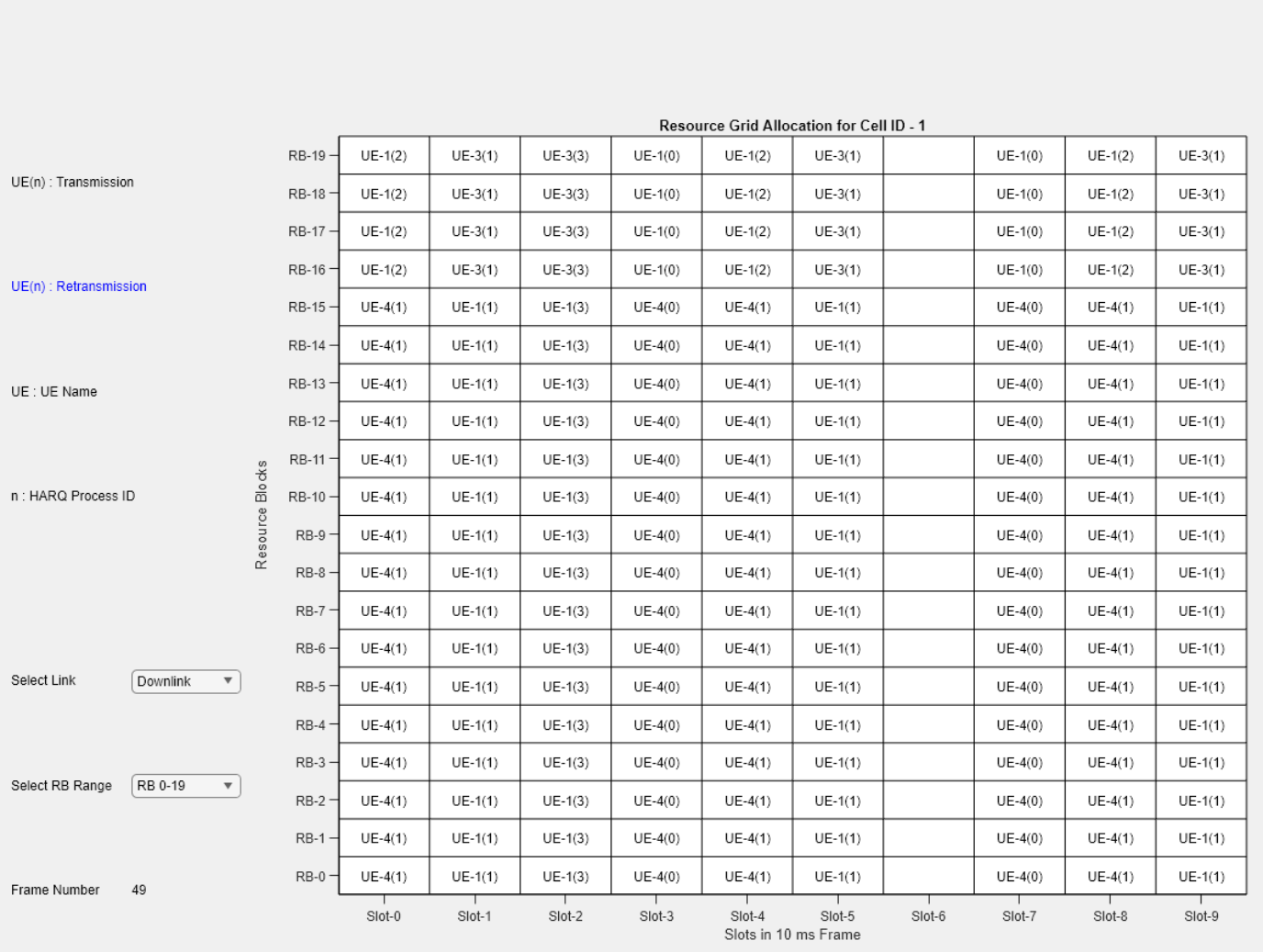
```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Cell
Peak Data Rate
Cell-1 DL Tx rate (Mbps)
Time (secs)
Cell-1 DL Resource Share (%)
Cell-1 DL Throughput (Mbps)
Cell-1 DL Buffer Status (KB)
Processing
Frames = 21 T = 0.500000


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Cell
Peak Data Rate
Cell-1 UL Tx rate (Mbps)
Cell-1 UL Resource Share (%)
Cell-1 UL Throughput (Mbps)
Cell-1 UL Buffer Status (KB)
Time (secs)
Processing
Frames = 21 T = 0.500000


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Downlink RLC
x10^4
Cell-1 Transmitted Bytes
10
0
0
0.04
0.08
0.12
0.16
0.2
0.24
0.28
0.32
0.36
0.4
0.44
0.48
Time (secs)
UE-1
UE-2
UE-3
UE-4
Uplink RLC
x10^4
Cell-1 Transmitted Bytes
10
0
0
0.04
0.08
0.12
0.16
0.2
0.24
0.28
0.32
0.36
0.4
0.44
0.48
Time (secs)
Processing
Frames = 21 T = 0.500000


Channel Quality Visualization for Cell ID - 1
Select Link
Downlink
Select RB Range
RB 0-19
Frame Number:
49
Resource Blocks
RB-19
RB-18
RB-17
RB-16
RB-15
RB-14
RB-13
RB-12
RB-11
RB-10
RB-9
RB-8
RB-7
RB-6
RB-5
RB-4
RB-3
RB-2
RB-1
RB-0
UE-1
UE-2
UE-3
UE-4
UEs
15
10
5
0



Read per-node stats.

```
gNBStats = statistics(gNB);
ueStats = statistics(UEs);
```

At the end of the simulation, the achieved value for system performance indicators is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (UL and DL), achieved spectral efficiency (UL and DL), and block error rate (BLER) observed for UEs (UL and DL). The peak values are calculated as per 3GPP TR 37.910.

```
displayPerformanceIndicators(metricsVisualizer)
```

```
Peak UL Throughput: 199.08 Mbps. Achieved Cell UL Throughput: 121.95 Mbps
Achieved UL Throughput for each UE: [46.25        17.48         39.8        18.42]
Peak UL spectral efficiency: 6.64 bits/s/Hz. Achieved UL spectral efficiency for cell: 4.07 bits/
Peak DL Throughput: 199.08 Mbps. Achieved Cell DL Throughput: 131.47 Mbps
Achieved DL Throughput for each UE: [45.96         16.4        48.64        20.47]
Peak DL spectral efficiency: 6.64 bits/s/Hz. Achieved DL spectral efficiency for cell: 4.38 bits/
```

Simulation Visualization

The run-time visualization shown are:

- *Display of UL scheduling metrics plots*: For details, see 'Uplink Scheduler Performance Metrics' figure description in “NR Cell Performance Evaluation with MIMO” on page 6-41 example.
- *Display of DL scheduling metrics plots*: For details, see 'Downlink Scheduler Performance Metrics' figure description in “NR Cell Performance Evaluation with MIMO” on page 6-41 example.
- *Display of RLC metrics plot*: The 'RLC Metrics Visualization' figure represents the number of bytes transmitted by RLC layer for each UE.

Simulation Logs

The simulation logs are saved in MAT-files for post-simulation analysis. The per time step logs, scheduling assignment logs, and RLC logs are saved in the MAT-file `simulationLogFile`. After the simulation, open the file to load `DLTimeStepLogs`, `ULTimeStepLogs`, `SchedulingAssignmentLogs`, RLC`Logs` in the workspace.

Time step logs: For more information on the time step log format, see “NR Cell Performance Evaluation with MIMO” on page 6-41.

Scheduling assignment logs: For more information on the scheduling log format, see “NR Cell Performance Evaluation with MIMO” on page 6-41.

RLC logs: Each row in the RLC logs represents a slot and contains this information:

- *Timestamp*: Timestamp (in milliseconds)
- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *UE RLC statistics*: *N*-by-P cell, where *N* is the number of UEs, and *P* is the number of statistics collected. Each row represents statistics of a UE. The last row contains the cumulative RLC statistics of the entire simulation.
- *gNB RLC statistics*: *N*-by-P cell, where *N* is the number of UEs, and *P* is the number of statistics collected. Each row represents statistics of all logical channel of a UE at gNB. The last row contains the cumulative RLC statistics of the entire simulation.

Each row of the UE and gNB RLC statistics table represents a logical channel of a UE and contains:

- *UEID*: Node ID of a UE.
- *RNTI*: Radio network temporary identifier of a UE.
- *TransmittedPackets*: Number of packets sent by RLC to MAC layer.
- *TransmittedBytes*: Number of bytes sent by RLC to MAC layer.
- *ReceivedPackets*: Number of packets received by RLC from MAC layer.
- *ReceivedBytes*: Number of bytes received by RLC from MAC layer.
- *DroppedPackets*: Number of received packets from MAC which are dropped by RLC layer.
- *DroppedBytes*: Number of received bytes from MAC which are dropped by RLC layer.

Save the simulation logs in a MAT file.

```
if enableTraces
    simulationLogs = cell(1,1);
```

```
    if gNB.DuplexMode == "FDD"
        logInfo = struct("DLTimeStepLogs",[],"ULTimeStepLogs",[],"SchedulingAssignmentLogs",[],"R
        [logInfo.DLTimeStepLogs,logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger)
    else % TDD
        logInfo = struct("TimeStepLogs",[],"SchedulingAssignmentLogs",[],"RLCLogs",[]);
        logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger);
    end
    % Get the scheduling assignments log
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger);
    % Get the RLC logs
    logInfo.RLCLogs = getRLCLogs(simRLCLogger);
    % Save simulation logs in a MAT-file
    simulationLogs{1} = logInfo;
    save(simulationLogFile,"simulationLogs")
end
```

Further Exploration

You can use this example to further explore TDD modeling.

TDD modeling

You can use this example to further explore TDD modeling by setting the DuplexMode property of the gNB object to "TDD". You can also customize the DL-UL slot pattern configuration for TDD by using the DLULConfigTDD property of the nrGNB object.

Appendix

The example uses these helper classes:

- helperNRMetricsVisualizer.m: Implements metrics visualization functionality
- helperNRSchedulingLogger.m: Implements scheduling information logging functionality
- helperNRPhyLogger.m: Implements Phy packet reception information logging functionality
- helperNRRLCLogger.m: Implements RLC packet transmission and reception information logging functionality
- helperNRGridVisualizer.m: Implements channel quality and resource grid visualization functionality

References

[1] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Objects
`wirelessNetworkSimulator` | `nrGNB` | `nrUE`

Related Examples

- “NR TDD Symbol Based Scheduling Performance Evaluation” on page 6-68

NR TDD Symbol Based Scheduling Performance Evaluation

This example models a symbol-based scheduling scheme in time division duplexing (TDD) mode and evaluates the network performance. Symbol-based scheduling of resources allows shorter transmission durations spanning a few symbols in the slot. In TDD mode, physical uplink shared channel (PUSCH) and physical downlink shared channel (PDSCH) transmissions are scheduled in the same frequency band with separation in time domain. You can customize the scheduling strategy and evaluate network performance. The performance of the scheduling strategy is evaluated in terms of achieved throughput and fairness in resource sharing.

Introduction

The example considers the following operations within the gNB and UEs that facilitate UL and DL transmissions and receptions.

	Operations
gNB	• Run the scheduling algorithm to assign uplink and downlink resources • Send the uplink and downlink assignments to the UEs • Receive the PUSCH transmissions from the UEs • Adhere to the downlink assignments for PDSCH transmission • Receive the feedback of PDSCHs from the UEs
UEs	• Send the pending buffer status report to the gNB • Receive the uplink and downlink assignments from the gNB • Adhere to the received uplink assignments from the gNB for PUSCH transmission • Receive the PDSCH transmissions from the gNB • Send feedback for the received PDSCHs

The complete PUSCH or PDSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

This example models:

- Configurable TDD DL-UL pattern.
- Slot-based and symbol-based DL and UL scheduling. UL scheduler ensures that the UEs get the required PUSCH preparation time.
- Noncontiguous allocation of frequency domain resources in terms of resource block groups (RBGs).
- Configurable subcarrier spacing resulting in different slot durations.
- Configurable demodulation reference signal (DM-RS) properties.
- Asynchronous hybrid automatic repeat request (HARQ) mechanism.
- Periodic DL and UL application traffic pattern.
- RLC operating in UM mode.
- Single bandwidth part covering the entire carrier bandwidth.

Following control packets are assumed to be sent out of band i.e. without the need of resources for transmission and assured error-free reception: UL assignment, DL assignment, buffer status report (BSR), PDSCH feedback, and CQI report. These control packets follow the TDD DL and UL timings. For example, BSR and PDSCH feedback are sent in UL time while resource assignments are sent in DL time.

TDD DL-UL Pattern Configuration

NR provides a flexible way of configuring the DL and UL resources. The parameters used to define a custom TDD configuration are:

1. DL-UL transmission periodicity in ms.
2. Reference subcarrier spacing to calculate the number of slots in the DL-UL pattern. In this example, it is assumed to be same as actual subcarrier spacing used for transmission.
3. Number of consecutive full DL slots at the beginning of each DL-UL pattern.
4. Number of consecutive DL symbols in the beginning of the slot following the last full DL slot.
5. Number of consecutive full UL slots at the end of each DL-UL pattern.
6. Number of consecutive UL symbols in the end of the slot preceding the first full UL slot.

The example does not model flexible symbols, so the symbols with unspecified symbol type are assumed for guard period. Here is an example of resulting TDD DL-UL pattern based on these parameters. This DL-UL pattern repeats itself in the timeline.

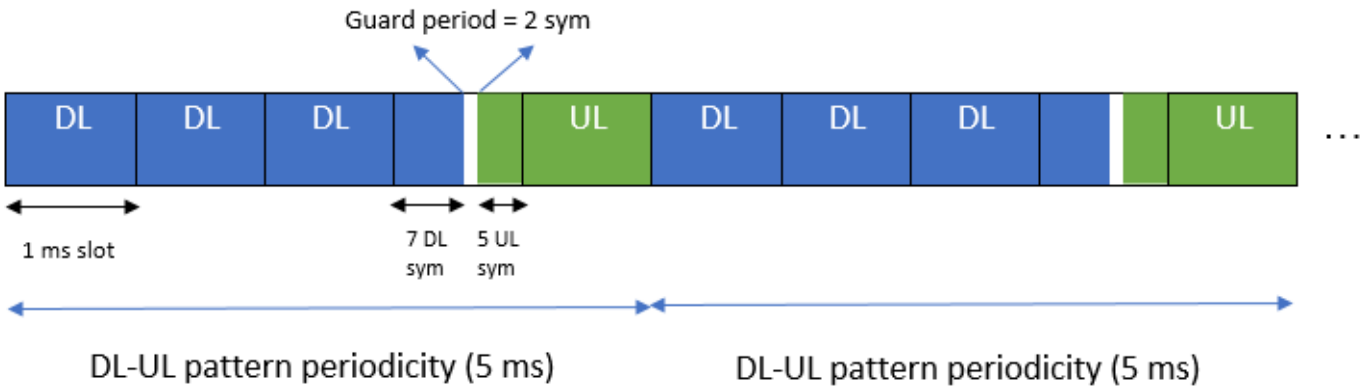
- `reference_scs` = 15 kHz (i.e. 1 ms slot), `DLULPeriodicity` = 5 ms, `numDLSlots` = 3, `numDLSyms` = 7, `numULSlots` = 1, `numULSyms` = *5*

Number of slots in DL-UL periodicity with respect to reference SCS of 15 kHz, `NumSlotsDLULPeriodicity` = 5

NumberOfGuardSymbols = TotalSymbolsInPattern - TotalSymbolsWithTypeSpecified

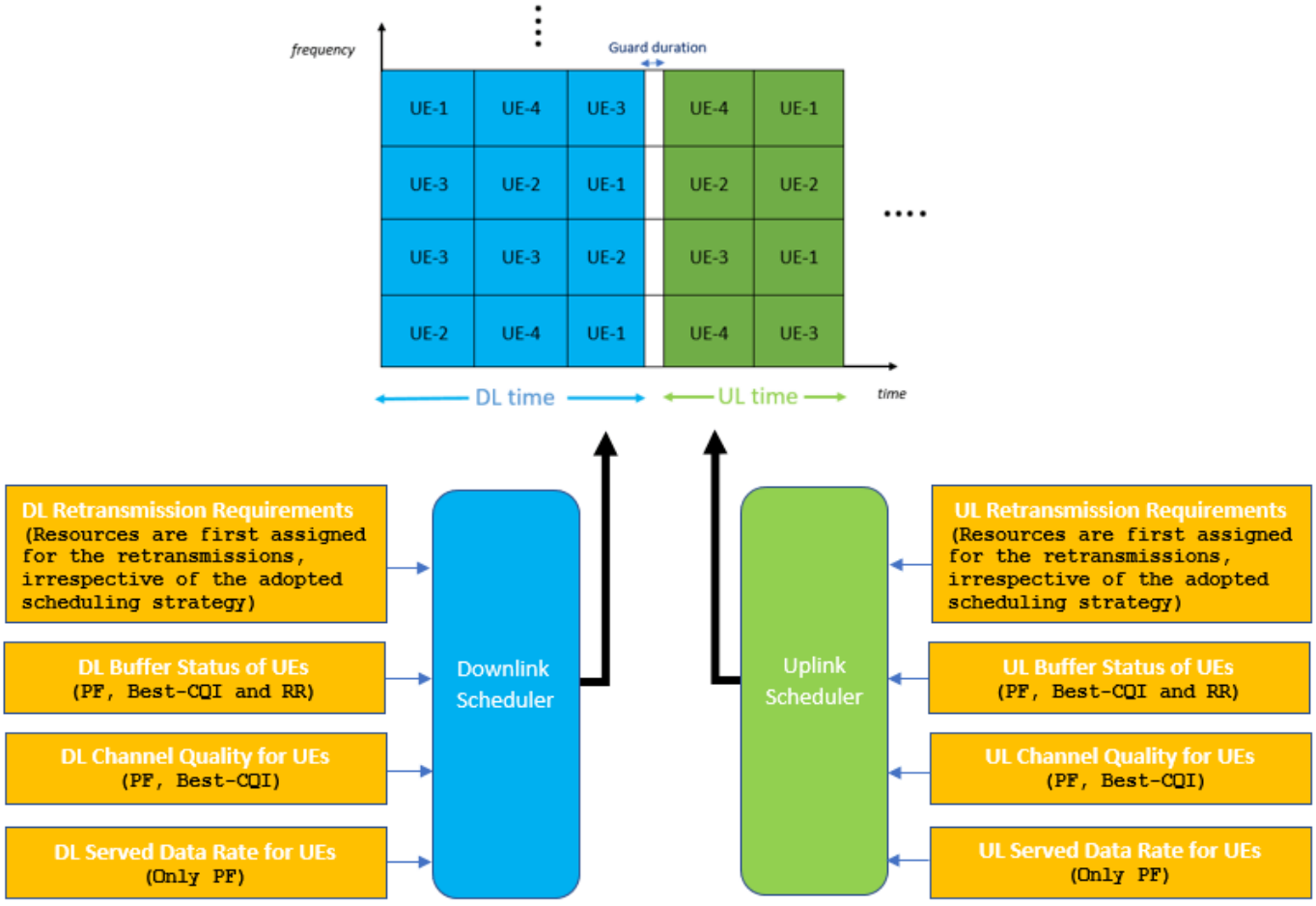
= (14 * `NumSlotsDLULPeriodicity`) - (`numDLSlots`*14 + `numDLSyms` + `numULSyms` + `numULSlots`*14)

= 2 symbols



Scheduler

UL and DL schedulers distribute the UL and DL resources respectively among the UEs. You can choose any one of the implemented scheduling strategies: proportional fair (PF), best CQI, or round-robin (RR). The various supported inputs to the schedulers are shown along with the scheduling strategies that consider them.

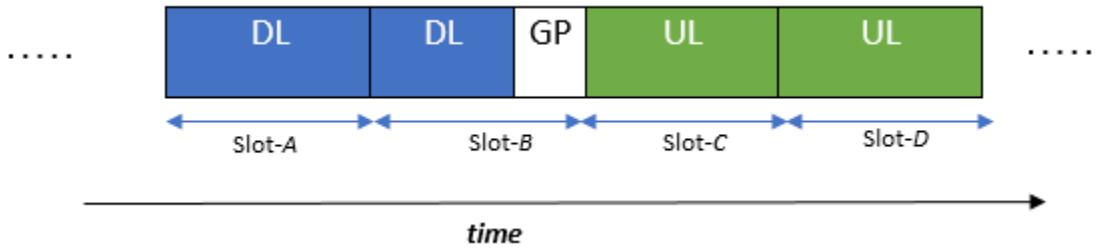


Both UL and DL schedulers run at the start of a slot when the first symbol is a DL symbol. Schedulers run in DL time so that assignments (UL and DL) can be instantly sent to UEs in the DL direction. The run time of the scheduler algorithm as well as the propagation delay is assumed to be zero. The output of scheduling operations is an array of assignments. Each assignment contains the information fields to fully define a PUSCH or PDSCH transmission.

- **UL Scheduler**

The UL scheduling operation follows these two steps.

1 *Select slots to be scheduled:* The criteria used in this example selects all the upcoming slots (including the current one) containing unscheduled UL symbols that must be scheduled now. Such slots must be scheduled now as they cannot be scheduled in the next slot with DL symbols, depending on the value of PUSCH preparation time capability of the UEs. It ensures that the UL resources are scheduled as close as possible to the actual transmission time.



Below are 2 examples to explain how UL slots are selected in this example for scheduling, based on PUSCH preparation time.

(i) Assuming that the UEs require PUSCH preparation time equivalent to 10 symbols, when the UL scheduler runs in Slot-A, it does not select any slot for scheduling. Because scheduling in the next slot (i.e. Slot-B) provides enough PUSCH preparation time (14 symbols) for the UL transmission in Slot-C. Later, when the UL scheduler runs in Slot-B, it selects both Slot-C and Slot-D for scheduling. Slot-D is scheduled in Slot-B itself (and not in Slot-C) because Slot-C is a full UL slot and hence does not have any DL symbols for sending the assignments in the DL direction.

(ii) Assuming that the UEs require PUSCH preparation time equivalent to 16 symbols, Slot-C is scheduled in Slot-A itself. Because scheduling in Slot-B would only provide 14 symbols of PUSCH preparation time to start transmission in Slot-C. Slot-D is scheduled in Slot-B.

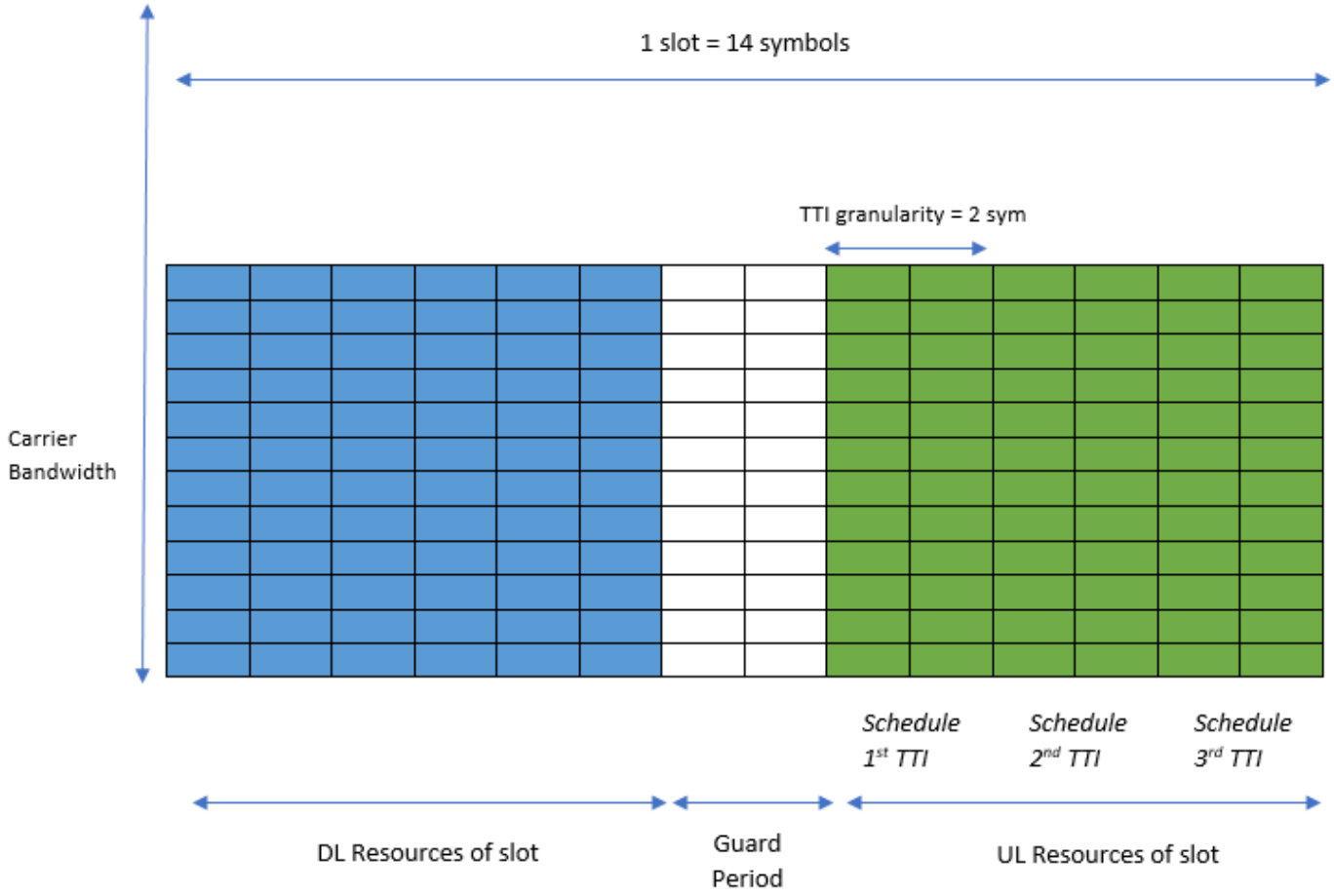
2. Resource scheduling: If any slots are selected in the first step, assign the UL resources of those slots to the UEs.

- **DL scheduler**

The DL scheduler runs at each slot beginning with a DL symbol and assign resources of the first upcoming slot containing DL symbols. So, when the DL scheduler runs at the start of Slot-A, it schedules DL resources of Slot-B.

Symbol based scheduling

NR allows the TTI to start at any symbol position in the slot and with TTI granularity in symbols. The figure shows the way UL scheduler operates in this example to schedule UL symbols of a slot with TTI granularity of two symbols. The slot shown contains six UL symbols. Scheduler completes the iteration of the UL symbols in three iterations with each iteration distributing the frequency resources of two symbols. The DL scheduler also follows a similar approach for DL scheduling.



Scenario Configuration

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Configure simulation parameters in the `simParameters` structure.

```
rng('default'); % Reset the random number generator
simParameters = []; % Clear simParameters variable
simParameters.NumFramesSim = 100; % Simulation time in terms of number of 10 ms frames
simParameters.SchedulingType = 1; % Set the value to 0 (slot-based scheduling) or 1 (symbol-based
```

Specify the number of UEs in the cell, assuming that UEs have sequential radio network temporary identifiers (RNTIs) from 1 to `simParameters.NumUEs`. If you change the number of UEs, ensure that the number of rows in `simParameters.UEPosition` parameter equals to the value of `simParameters.NumUEs`.

```
simParameters.NumUEs = 4;
% Assign position to the UEs assuming that the gNB is at (0, 0, 0). N-by-3
```

```
% matrix where 'N' is the number of UEs. Each row has (x, y, z) position of a
% UE (in meters)
simParameters.UEPosition = [100 0 0;
                            150 0 0;
                            300 0 0;
                            400 0 0];
% Validate the UE positions
validateattributes(simParameters.UEPosition,{'numeric'},{'nonempty','real','nrows',simParameters
    'finite'},'simParameters.UEPosition','UEPosition');
```

Set the channel bandwidth to 5 MHz and the subcarrier spacing (SCS) to 15 kHz as defined in 3GPP TS 38.104 Section 5.3.2. The complete bandwidth is assumed to be allotted for PUSCH or PDSCH.

```
simParameters.NumRBs = 25;
simParameters.SCS = 15; % kHz
simParameters.DLBandwidth = 5e6; % Hz
simParameters.ULBandwidth = 5e6; % Hz
simParameters.DLCarrierFreq = 2.595e9; % Hz
simParameters.ULCarrierFreq = 2.595e9; % Hz
```

Specify the TDD DL-UL pattern. The reference subcarrier spacing used for calculating slot duration for the pattern is assumed to be same as actual subcarrier spacing used for transmission as defined by `simParameters.SCS`. Keep only the symbols intended for guard period during DLULPeriodicity with type (DL or UL) unspecified.

```
simParameters.DLULPeriodicity = 5; % Duration of the DL-UL pattern in ms
simParameters.NumDLSlots = 2; % Number of consecutive full DL slots at the beginning of each DL-U
simParameters.NumDLSyms = 8; % Number of consecutive DL symbols in the beginning of the slot fol
simParameters.NumULSyms = 4; % Number of consecutive UL symbols in the end of the slot preceding
simParameters.NumULSlots = 2; % Number of consecutive full UL slots at the end of each DL-UL patt
```

Specify the scheduling strategy, the time domain resource assignment granualarity and the maximum limit on the RBs allotted for PDSCH and PUSCH. The time domain resource assignment granualarity is applicable only for symbol-based scheduling. If the number of symbols (DL or UL) are less than the configured time domain resource assignment granularity, then a smaller valid granularity is chosen. For slot-based scheduling, biggest possible granularity in a slot is chosen. The RB transmission limit applies only to new transmissions and not to the retransmissions.

```
simParameters.SchedulerStrategy = 'PF'; % Supported scheduling strategies: 'PF', 'RR' and 'BestC
simParameters.TTIGranularity = 4;
simParameters.RBAllocationLimitUL = 15; % For PUSCH
simParameters.RBAllocationLimitDL = 15; % For PDSCH
```

Set the UL scheduling related configurations - BSR periodicity and PUSCH preparation time. gNB ensures that PUSCH assignment is received at the UEs at least `PUSCHPrepTime` ahead of the transmission time.

```
simParameters.BSRPeriodicity = 1; % Buffer status report transmission periodicity (in ms)
simParameters.PUSCHPrepTime = 200; % In microseconds
```

Set the channel quality related configurations for the UEs. Channel quality is periodically improved or deteriorated by CQI Delta for all RBs of a UE. Whether channel conditions for a particular UE improve or deteriorate is randomly determined by: RB_CQI = RB_CQI +/- CQIDelta. However, the maximum allowed CQI value depends on the UE position and is determined by CQIvsDistance mapping table. This mapping is only applicable when passthrough PHY is used.

```
simParameters.ChannelUpdatePeriodicity = 0.2; % In sec
simParameters.CQIDelta = 1;
```

```
% Mapping between distance from gNB (first column in meters) and maximum achievable UL CQI value
simParameters.CQIvsDistance = [
    200  15;
    300  12;
    500  10;
    1000  8;
    1200  7];
```

Specify the PUSCH and PDSCH associated DMRS configurations.

```
simParameters.DMRSTypeAPosition = 2; % Type-A DM-RS position as 2 or 3
% PUSCH DM-RS configuration
simParameters.PUSCHDMRSAdditionalPosTypeB = 0;
simParameters.PUSCHDMRSAdditionalPosTypeA = 0;
simParameters.PUSCHDMRSConfigurationType = 1;
% PDSCH DM-RS configuration
simParameters.PDSCHDMRSAdditionalPosTypeB = 0;
simParameters.PDSCHDMRSAdditionalPosTypeA = 0;
simParameters.PDSCHDMRSConfigurationType = 1;
```

Application traffic configuration

Set the periodic DL and UL application traffic pattern for UEs.

```
% Set the periodic DL and UL application traffic pattern for UEs
dlAppDataRate = 16e4*ones(simParameters.NumUEs,1); % DL application data rate in kilo bits per se
ulAppDataRate = 16e4*ones(simParameters.NumUEs,1); % UL application data rate in kbps
% Validate the DL application data rate
validateattributes(dlAppDataRate, {'numeric'},{'nonempty','vector','numel',simParameters.NumUEs,
    'dlAppDataRate','dlAppDataRate');
% Validate the UL application data rate
validateattributes(ulAppDataRate,{'numeric'},{'nonempty','vector','numel',simParameters.NumUEs,'
    'ulAppDataRate', 'ulAppDataRate');
```

Logging and visualization configuration

The `CQIVisualization` and `RBVisualization` parameters control the display of the CQI visualization and the RB assignment visualization respectively. To enable the RB visualization plot, set the `RBVisualization` field to `true`.

```
simParameters.CQIVisualization = false;
simParameters.RBVisualization = false;
```

Set the `enableTraces` as `true` to log the traces. If the `enableTraces` is set to `false`, then `CQIVisualization` and `RBVisualization` are disabled automatically and traces are not logged in the simulation. To speed up the simulation, set the `enableTraces` to `false`.

```
enableTraces = true;
```

The example updates the metrics plots periodically. Set the number of updates during the simulation. Number of steps must be less than or equal to number of slots in simulation

```
simParameters.NumMetricsSteps = 20;
```

Write the logs to MAT-files. The example uses these logs for post-simulation analysis and visualization.

```
parametersLogFile = 'simParameters'; % For logging the simulation parameters
simulationLogFile = 'simulationLogs'; % For logging the simulation traces
simulationMetricsFile = 'simulationMetrics'; % For logging the simulation metrics
```

Derived Parameters

Compute the derived parameters based on the primary configuration parameters specified in the previous section and additionally set some example-specific constants.

```
simParameters.DuplexMode = 1; % FDD (Value as 0) or TDD (Value as 1)
simParameters.NCellID = 1; % Physical cell ID
simParameters.Position = [0 0 0]; % Position of gNB in (x,y,z) coordinates
```

Compute the number of slots in the simulation.

```
numSlotsSim = (simParameters.NumFramesSim * 10 * simParameters.SCS)/15;
```

Determine the PDSCH/PUSCH mapping type.

```
if simParameters.SchedulingType % Symbol-based scheduling
    simParameters.PUSCHMappingType = 'B';
    simParameters.PDSCHMappingType = 'B';
else % Slot-based scheduling
    simParameters.PUSCHMappingType = 'A';
    simParameters.PDSCHMappingType = 'A';
end
```

Set the interval at which the example updates metrics visualization in terms of number of slots.

```
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
```

Specify one logical channel for each UE, and set the logical channel configuration for all nodes (UEs and gNBs) in the example.

```
numLogicalChannels = 1;
simParameters.LCHConfig.LCID = 4;
```

Specify the RLC entity type in the range [0, 3]. The values 0, 1, 2, and 3 indicate RLC UM unidirectional DL entity, RLC UM unidirectional UL entity, RLC UM bidirectional entity, and RLC AM entity, respectively.

```
simParameters.RLCConfig.EntityType = 2;
```

Construct information for RLC logger.

```
lchInfo = repmat(struct('RNTI',[],'LCID',[],'EntityDir',[]), [simParameters.NumUEs 1]);
for idx = 1:simParameters.NumUEs
    lchInfo(idx).RNTI = idx;
    lchInfo(idx).LCID = simParameters.LCHConfig.LCID;
    lchInfo(idx).EntityDir = simParameters.RLCConfig.EntityType;
end
```

Create RLC channel configuration structure.

```
rlcChannelConfigStruct.LCGID = 1; % Mapping between logical channel and logical channel group ID
rlcChannelConfigStruct.Priority = 1; % Priority of each logical channel
rlcChannelConfigStruct.PBR = 8; % Prioritized bitrate (PBR), in kilobytes per second, of each log
rlcChannelConfigStruct.BSD = 10; % Bucket size duration (BSD), in ms, of each logical channel
rlcChannelConfigStruct.EntityType = simParameters.RLCConfig.EntityType;
rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID;
```

Set the maximum RLC SDU length (in bytes) as per 3GPP TS 38.323

```
simParameters.maxRLCSDULength = 9000;
```

Calculate maximum achievable CQI value for the UEs based on their distance from the gNB

```
maxUECQIs = zeros(simParameters.NumUEs, 1); % To store the maximum achievable CQI value for UEs
for ueIdx = 1:simParameters.NumUEs
    % Based on the distance of the UE from gNB, find matching row in CQIvsDistance mapping
    matchingRowIdx = find(simParameters.CQIvsDistance(:, 1) > simParameters.UEPosition(ueIdx,1))
    if isempty(matchingRowIdx)
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(end, 2);
    else
        maxUECQIs(ueIdx) = simParameters.CQIvsDistance(matchingRowIdx(1), 2);
    end
end
```

Define initial UL and DL channel quality as an N-by-P matrix, where 'N' is the number of UEs and 'P' is the number of RBs in the carrier bandwidth. The initial value of CQI for each RB, for each UE, is given randomly and is limited by the maximum achievable CQI value corresponding to the distance of the UE from gNB.

```
simParameters.InitialChannelQualityUL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To s
simParameters.InitialChannelQualityDL = zeros(simParameters.NumUEs, simParameters.NumRBs); % To s
for ueIdx = 1:simParameters.NumUEs
    % Assign random CQI values for the RBs, limited by the maximum achievable CQI value
    simParameters.InitialChannelQualityUL(ueIdx, :) = randi([1 maxUECQIs(ueIdx)], 1, simParameter
    % Initially, DL and UL CQI values are assumed to be equal
    simParameters.InitialChannelQualityDL(ueIdx, :) = simParameters.InitialChannelQualityUL(ueIdx
end
```

gNB and UEs Setup

Create the gNB and UE objects, initialize the channel quality information for UEs and set up the logical channel at gNB and UE. The helper classes hNRGNB.m and hNRUE.m create gNB and UE node respectively, containing the RLC and MAC layer. For MAC layer, hNRGNB.m uses the helper class hNRGNBMAC.m to implement the gNB MAC functionality and hNRUE.m uses hNRUEMAC.m to implement the UE MAC functionality. Schedulers are implemented in hNRSchedulerRoundRobin.m (RR), hNRSchedulerProportionalFair.m (PF), hNRSchedulerBestCQI.m (Best CQI) . All the schedulers inherit from the base class hNRScheduler.m which contains the core scheduling functionality. For RLC layer, both hNRGNB.m and hNRUE.m use hNRUMEntity.m to implement the functionality of the RLC transmitter and receiver. Passthrough PHY layer for UE and gNB is implemented in hNRUEPassThroughPhy.m and hNRGNBPassThroughPhy.m, respectively.

Create the gNB node and add scheduler

```
gNB = hNRGNB(simParameters);
switch(simParameters.SchedulerStrategy)
    case 'RR' % Round-robin scheduler
        scheduler = hNRSchedulerRoundRobin(simParameters);
    case 'PF' % Proportional fair scheduler
        scheduler = hNRSchedulerProportionalFair(simParameters);
    case 'BestCQI' % Best CQI scheduler
        scheduler = hNRSchedulerBestCQI(simParameters);
end
addScheduler(gNB, scheduler); % Add scheduler to gNB

gNB.PhyEntity = hNRGNBPassThroughPhy(simParameters); % Add passthrough PHY
configurePhy(gNB, simParameters);
setPhyInterface(gNB); % Set the interface to PHY layer
```

Create the set of UE nodes.

```
UEs = cell(simParameters.NumUEs, 1);
for ueIdx = 1:simParameters.NumUEs
    simParameters.Position = simParameters.UEPosition(ueIdx, :); % Position of the UE
    UEs{ueIdx} = hNRUE(simParameters, ueIdx);
    UEs{ueIdx}.PhyEntity = hNRUEPassThroughPhy(simParameters, ueIdx); % Add passthrough PHY
    configurePhy(UEs{ueIdx}, simParameters);
    setPhyInterface(UEs{ueIdx}); % Set the interface to PHY layer

    % Initialize the UL CQI values at gNB scheduler
    channelQualityInfoUL = struct('RNTI', ueIdx, 'CQI', simParameters.InitialChannelQualityUL(ue
    updateChannelQualityUL(gNB.MACEntity.Scheduler, channelQualityInfoUL);

    % Initialize the DL CQI values at gNB scheduler
    channelQualityInfoDL = struct('RNTI', ueIdx, 'CQI', simParameters.InitialChannelQualityDL(ue
    updateChannelQualityDL(gNB.MACEntity.Scheduler, channelQualityInfoDL);

    % Initialize the DL CQI values at UE for packet error probability estimation
    updateChannelQualityDL(UEs{ueIdx}.MACEntity, channelQualityInfoDL);

    % Setup logical channel at gNB for the UE
    configureLogicalChannel(gNB, ueIdx, rlcChannelConfigStruct);
    % Setup logical channel at UE
    configureLogicalChannel(UEs{ueIdx}, ueIdx, rlcChannelConfigStruct);

    % Create an object for On-Off network traffic pattern and add it to the
    % specified UE. This object generates the uplink (UL) data traffic on the UE
    ulApp = networkTrafficOnOff('GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', ulAppDataRate(ueIdx))
    UEs{ueIdx}.addApplication(ueIdx, simParameters.LCHConfig.LCID, ulApp);

    % Create an object for On-Off network traffic pattern for the specified
    % UE and add it to the gNB. This object generates the downlink (DL) data
    % traffic on the gNB for the UE
    dlApp = networkTrafficOnOff('GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim/100, 'OffTime', 0, 'DataRate', dlAppDataRate(ueIdx))
    gNB.addApplication(ueIdx, simParameters.LCHConfig.LCID, dlApp);
end
```

Simulation

```
% Initialize wireless network simulator
nrNodes = [{gNB}; UEs];
networkSimulator = hWirelessNetworkSimulator(nrNodes);
```

Create objects to log RLC and MAC traces.

```
if enableTraces

    % RLC metrics are logged for every 1 slot duration
    simRLCLogger = hNRRLCLogger(simParameters, lchInfo, networkSimulator, gNB, UEs);

    simSchedulingLogger = hNRSchedulingLogger(simParameters, networkSimulator, gNB, UEs);

    % Create an object for CQI and RB grid visualization
    if simParameters.CQIVisualization || simParameters.RBVisualization
        gridVisualizer = hNRGridVisualizer(simParameters, 'MACLogger', simSchedulingLogger);
```

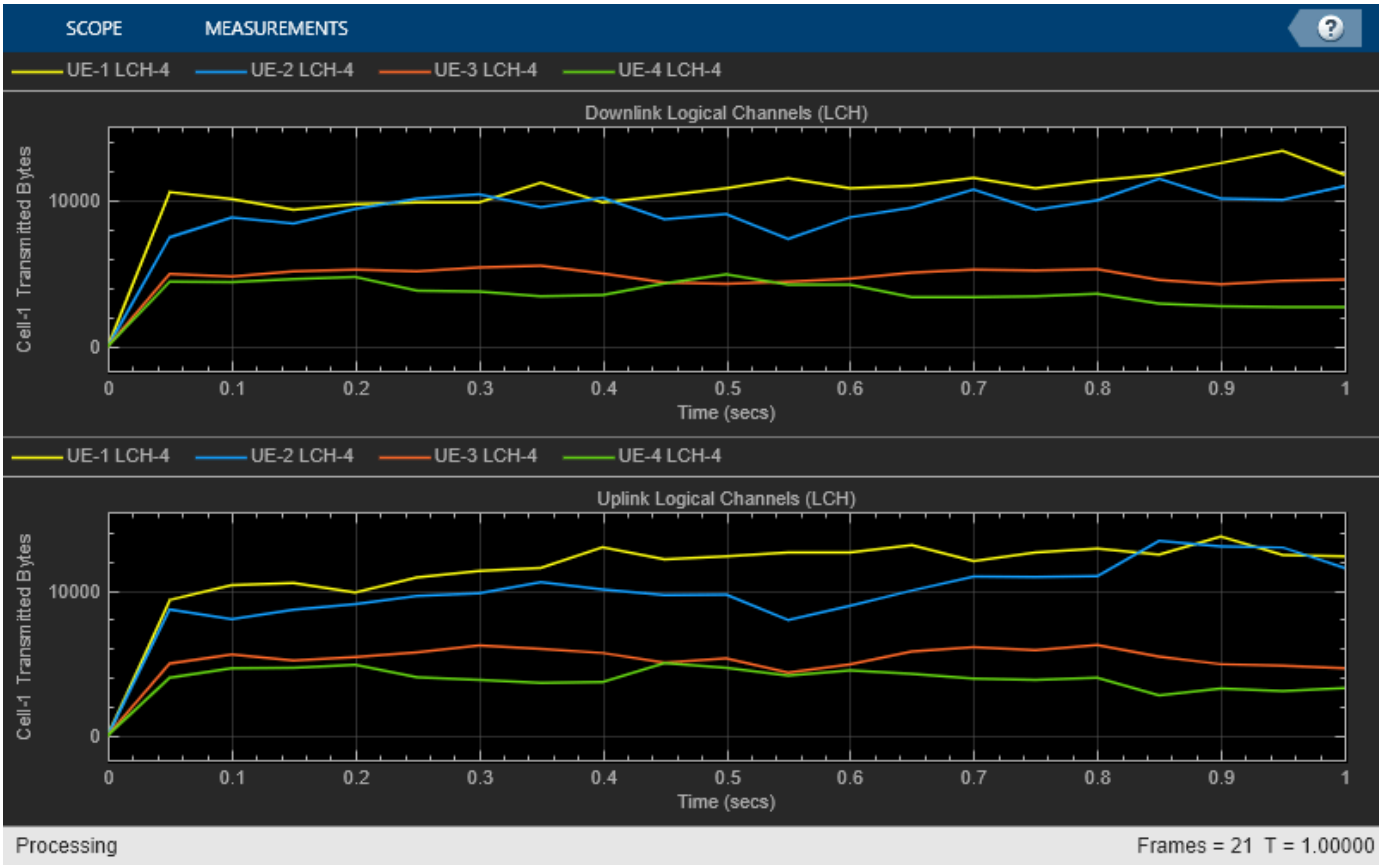
```
    end
end
```

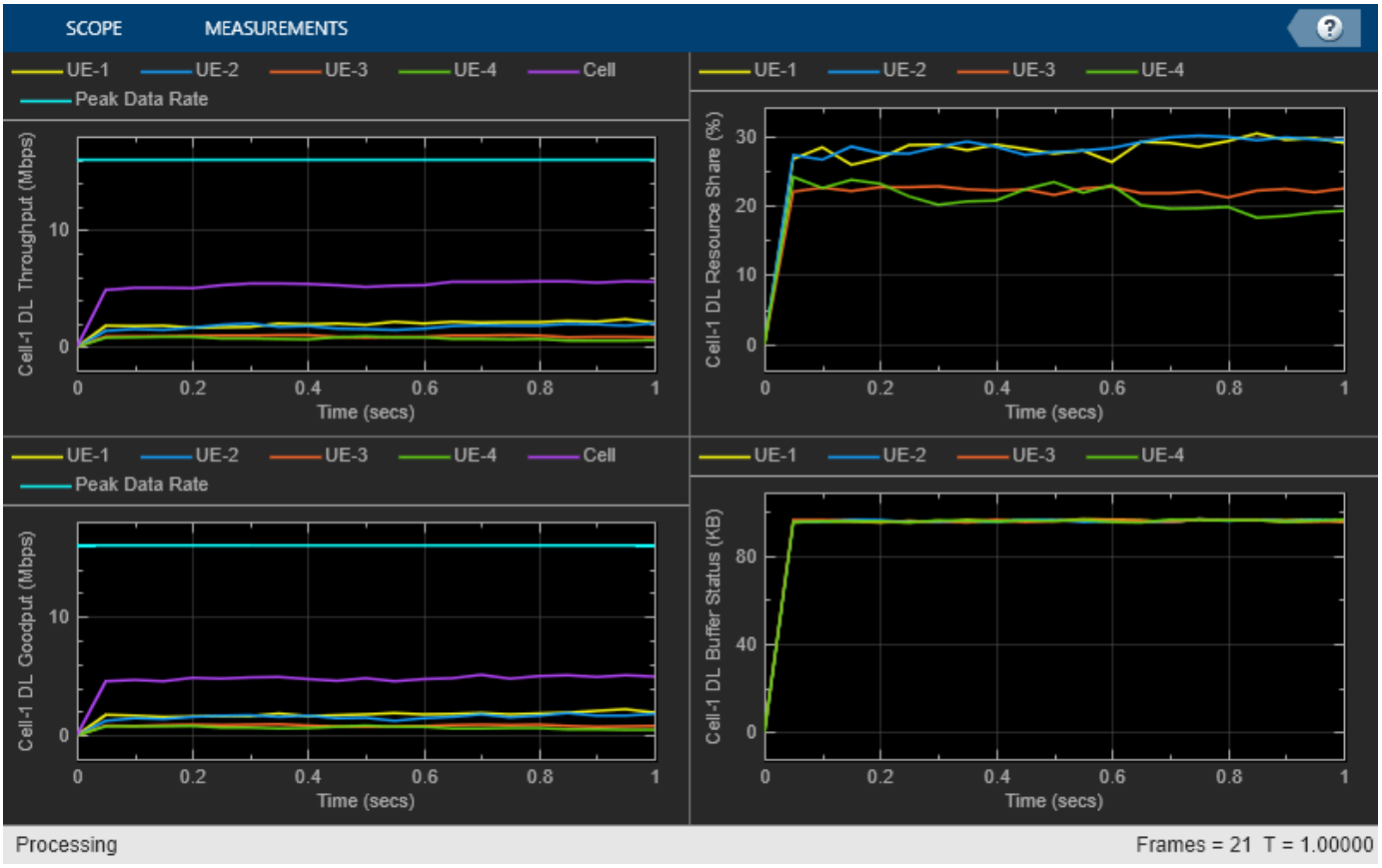
Create an object for RLC and MAC metrics visualization.

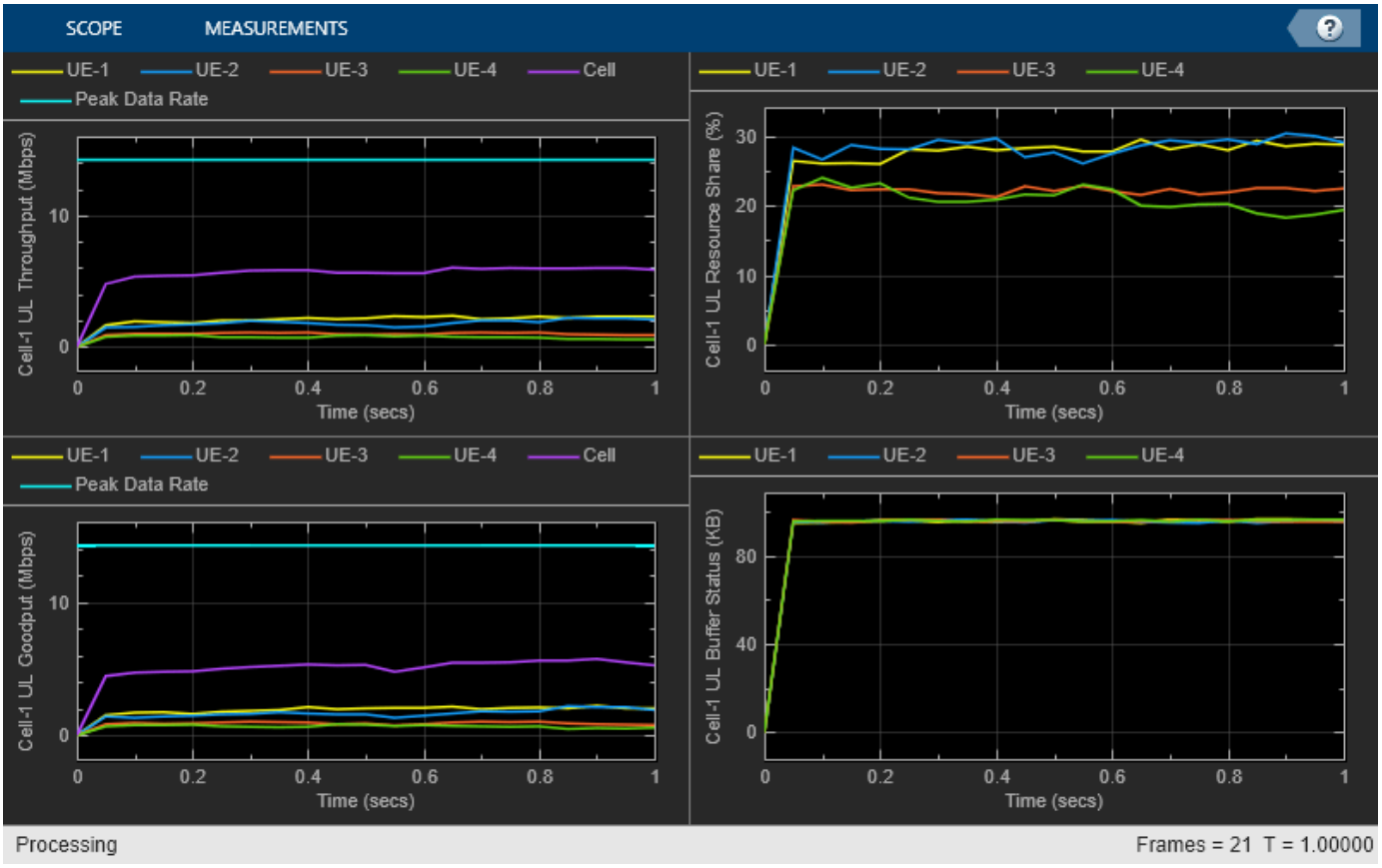
```
metricsVisualizer = hNRMetricsVisualizer(simParameters, 'EnableSchedulerMetricsPlots', true, ...
    'EnableRLCMetricsPlots', true, 'LCHInfo', lchInfo, 'NetworkSimulator', networkSimulator, 'GNB
```

Run the simulation for the specified `NumFramesSim` frames.

```
% Calculate the simulation duration (in seconds) from 'NumFramesSim'
simulationTime = simParameters.NumFramesSim * 1e-2;
% Run the simulation
run(networkSimulator, simulationTime);
```




SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Cell
Peak Data Rate
Cell-1 DL Throughput (Mbps)
Cell-1 DL Resource Share (%)
Cell-1 DL Goodput (Mbps)
Cell-1 DL Buffer Status (KB)
Time (secs)
Processing
Frames = 21 T = 1.00000



At the end of the simulation, the achieved value for system performance indicator is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (UL and DL), and achieved spectral efficiency (UL and DL). The peak values are calculated as per 3GPP TR 37.910.

```
displayPerformanceIndicators(metricsVisualizer);
```

```
Peak UL Throughput: 14.22 Mbps. Achieved Cell UL Throughput: 5.72 Mbps
Achieved UL Throughput for each UE: [2.13        1.83           1        0.75]
Achieved Cell UL Goodput: 5.18 Mbps
Achieved UL Goodput for each UE: [1.94        1.67         0.9        0.67]
Peak UL spectral efficiency: 2.84 bits/s/Hz. Achieved UL spectral efficiency for cell: 1.04 bits/

Peak DL Throughput: 16.00 Mbps. Achieved Cell DL Throughput: 5.35 Mbps
Achieved DL Throughput for each UE: [1.99        1.73        0.92        0.72]
Achieved Cell DL Goodput: 4.82 Mbps
Achieved DL Goodput for each UE: [1.78        1.56        0.82        0.64]
Peak DL spectral efficiency: 3.20 bits/s/Hz. Achieved DL spectral efficiency for cell: 0.96 bits/
```

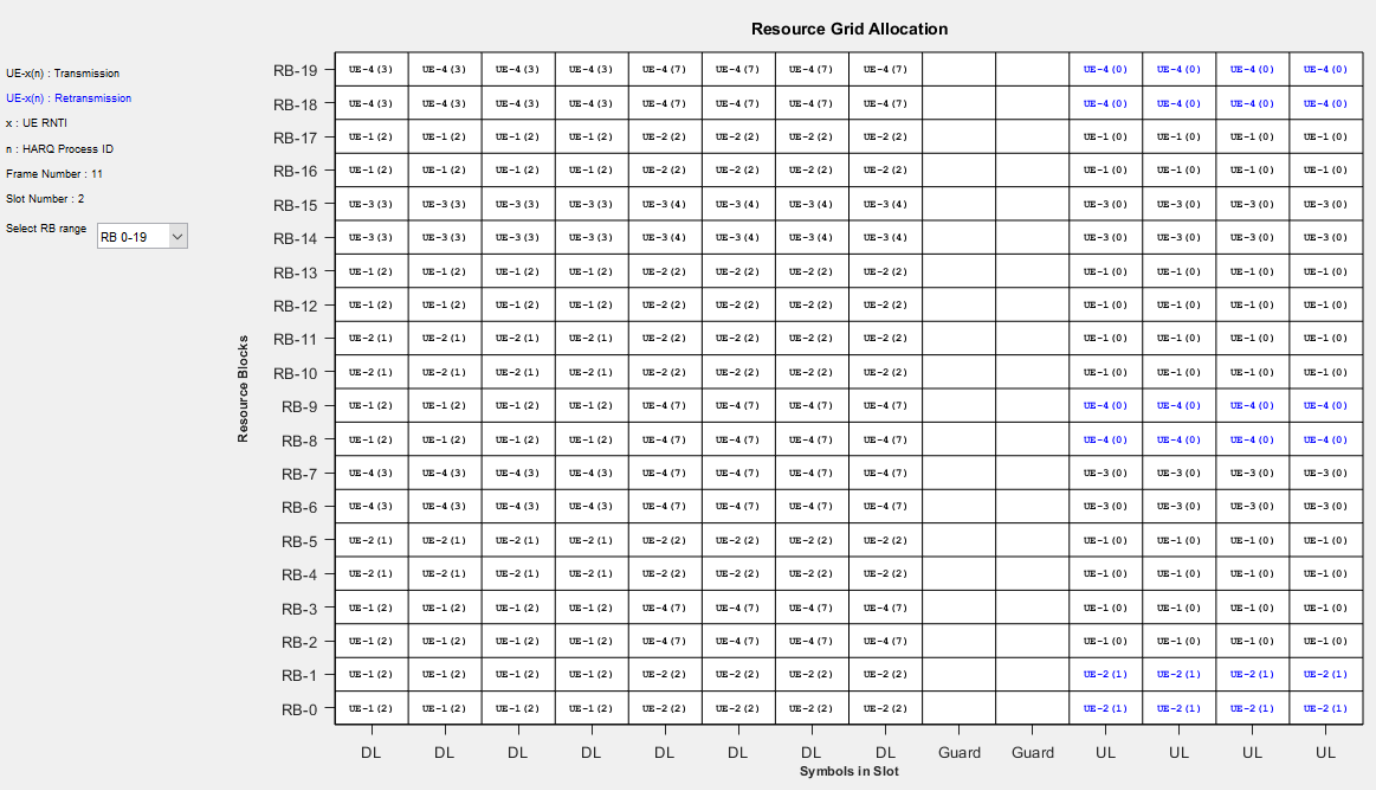
Get the simulation metrics and save it in a MAT-file. The simulation metrics are saved in a MAT-file with the file name as `simulationMetricsFile`

```
metrics = getMetrics(metricsVisualizer);
save(simulationMetricsFile, 'metrics'); % Save simulation metrics in a MAT-file
```

Simulation Visualization

The five types of runtime visualization shown are:

- *Display of CQI values for UEs over the channel bandwidth:* For details, see the 'Channel Quality Visualization' figure.
- *Display of resource grid assignment to UEs:* The 2-D time-frequency grid shows the resource allocation to the UEs. For slot-based scheduling, it updates every 10 ms (frame length), and shows the RB allocation to the UEs in the previous frame. For symbol-based scheduling, it updates every slot, and shows the RB allocation of symbols of the previous slot. For details, see the 'Resource Grid Allocation' figure.
- *Display of UL scheduling metrics plots:* The 'Uplink Scheduler Performance Metrics' figure includes plots of the: UL throughput (per UE and cell), UL goodput (per UE and cell), resource share percentage among UEs (out of the total UL resources) to convey the fairness of scheduling, and pending UL buffer status of the UEs to show whether UEs are getting sufficient resources. The maximum achievable data rate value for UL throughput is shown with a dashed line in throughput and goodput plots. The performance metrics plots update for every `metricsStepSize` slots. Here, throughput is the number of bytes transmitted (including retransmissions) by a UE during a specified period, and goodput is the number of newly transmitted bytes by a UE during a specified period.
- *Display of DL scheduling metrics plots:* Like uplink metrics plots, the 'Downlink Scheduler Performance Metrics' displays corresponding subplots for DL direction. The performance metrics plots update for every `metricsStepSize` slots. Here, throughput is the number of bytes transmitted (including retransmissions) by a gNB during a specified period, and goodput is the number of newly transmitted bytes by a gNB during a specified period to each UE.
- *Display of RLC metrics plot:* The 'RLC Metrics Visualization' figure shows the number of bytes transmitted by RLC layer (per logical channel) for each UE. The RLC metrics plot updates for every `metricsStepSize` slots.



Simulation Logs

The parameters used for simulation and the simulation logs are saved in MAT-files for post-simulation analysis and visualization. The simulation parameters are saved in MAT-file with filename as the value of configuration parameter `parametersLogFile`. The per time step logs, scheduling assignment logs, and RLC logs are saved in the MAT-file `simulationLogFile`. After the simulation, open it to load `TimeStepLogs`, `SchedulingAssignmentLogs`, and `RLCLogs` in the workspace.

Time step logs: The table shows a sample time step entry. Each row of the table represents a symbol or a slot, based on the chosen scheduling type (symbol-based or slot-based). The information in a row is for DL, if the type of symbol (or slot) is `DL`. Likewise, for `UL` symbol (or slot).

Timestamp	Frame	Slot	Symbol	Type	RBG Allocation Bitmap	MCS	HARQ Process	NDI	Tx Type	CQI for UEs	HARQ NDI Status	Throughput Bytes	Goodput Bytes	Buffer Status of UEs
50	5	0	0	....	.....	.....	.....	....	.....	.....	.....	.....	.....	.....
51.2857	5	1	4	'UL'	[0 0 1 1 0 1 0 1 0 1 0 0 0] [1 1 1 0 0 0 0 0 0 0 1 0 0] [0 0 0 0 1 0 1 0 1 0 0 1 1] [0 0 0 0 0 0 0 0 0 0 0 0 0]	[10 12 8 -1]	[0 3 6 -1]	[0 0 1 -1]	['newTx', 'newTx', 'reTx', 'noTx']	[5 8 9 8 7 8 9 8 9 11 12 . . . . .] [5 7 3 4 9 5 9 7 9 10 15 . . . . .] [3 9 9 6 9 4 9 8 9 9 13 . . . . .] [9 7 7 3 7 8 5 8 8 15 9 . . . . .]	[1 0 0 1 0 1 0 . .] [1 1 0 1 0 1 0 . .] [1 0 0 1 0 1 1 . .] [1 0 0 1 0 1 0 . .]	[46 54 38 0]	[46 54 0 0]	[299480 135671 77567 137070]
52	5	2	0	....	.....	.....	.....	....	.....	.....	.....	.....	.....	.....

Each row contains the following information:

- *Timestamp*: Timestamp (in milliseconds)
- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- `Symbol`: Symbol number in the slot (Only for symbol-based scheduling).

- Type: Symbol (or Slot) type as 'DL', 'UL', or 'Guard'. For slot-based scheduling, type can only be DL/UL. As the slot containing the guard symbols is assumed to be a DL slot with guard symbols at the end of the slot.
- *RBG Allocation Bitmap*: *N-by-P* bitmap matrix, where *N* is the number of UEs and *P* is the number of RBGs in the bandwidth. If an RBG is assigned to a particular UE, the corresponding bit is set to 1. For example, [0 0 1 1 0 1 0 1 0 1 0 0 0; 1 1 0 0 0 0 0 0 0 0 1 0 0; 0 0 0 0 1 0 1 0 1 0 0 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0] means, that the bandwidth has 13 RBGs and UE-1 is assigned the RBG indices 2, 3, 5, 7, and 9; UE-2 is assigned the RBG indices: 0, 1, and 10; UE-3 is assigned the RBG indices: 4, 6, 8, 11, and 12; and UE-4 is not assigned any RBG.
- *MCS*: Row vector of length *N* where *N* is the number of UEs. Each value corresponds to the modulation and coding scheme (MCS) index for the PUSCH or PDSCH transmission. For example, [10 12 8 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources (symbol type is 'UL') for this symbol and use MCS values 10, 12, and 8, respectively.
- *HARQ Process*: Row vector of length *N,* where *N* is the number of UEs. The value is the HARQ process ID used by UE for the PUSCH transmission or used by gNB for PDSCH transmission. For example, [0 3 6 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources (symbol type is 'UL') for this symbol and use the HARQ process IDs 0, 3, and 6, respectively.
- *NDI*: Row vector of length *N,* where *N* is the number of UEs. The value is the NDI flag value in the assignment for PUSCH or PDSCH transmission. For example, [0 0 1 -1] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this symbol and use the NDI flag values (which determine whether a new transmission or a retransmission is done) are 0, 0, and 1, respectively.
- *Tx Type*: Tx Type specifies the transmission type (new transmission or retransmission). Row vector of length *N,* where *N* is the number of UEs. Possible values are either 'newTx', 'reTx', or 'noTx'. 'noTx' means that the UE is not allotted PUSCH or PDSCH resources. For example: ['newTx' 'newTx' 'reTx' 'noTx'] means that only UE-1, UE-2, and UE-3 are assigned UL resources for this symbol. UE-1 and UE-2 transmit a new packet from the specified HARQ process, while UE-3 retransmits the packet in the buffer of the specified HARQ process.
- *CQI for UEs*: *N-by-P* matrix, where *N* is the number of UEs and *P* is the number of RBs in the bandwidth. A matrix element at position *(i, j)* corresponds to the CQI value for UE with RNTI *i* at RB *j*.
- *HARQ NDI Status*: *N-by-P* matrix, where *N* is the number of UEs and *P* is the number of HARQ processes. A matrix element at position *(i, j)* is the last received NDI flag at UE *i* for DL or UL HARQ process ID *j*. For new transmissions, this value and the NDI flag in the PUSCH or PDSCH assignment must toggle for the HARQ process in the assignment.
- *Throughput Bytes*: Row vector of length *N,* where *N* is the number of UEs. The values represent UL or DL MAC bytes transmitted by or for the UEs in this symbol. Note that the total throughput bytes for the complete PUSCH or PDSCH transmission are shown in the row corresponding to the first symbol of the transmission.
- *Goodput Bytes*: Row vector of length *N,* where *N* is the number of UEs. The values represent new UL or DL transmission MAC bytes transmitted by or for the UEs in this symbol. Like throughput, all the goodput bytes for the complete PUSCH or PDSCH are shown in the row corresponding to first symbol of the transmission.
- *Buffer Status of UEs*: Row vector of length *N,* where *N* is the number of UEs. The values represent the amount of UL direction pending buffers at UEs (or DL direction pending buffers for UEs at gNB).

Scheduling assignment logs: Information of all the scheduling assignments and related information is logged in this table. Each row is one UL or DL assignment. For details of log format, see the 'Simulation Logs' section of “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87 example.

RLC logs: Each row in the RLC logs represents a slot and contains this information:

- *Timestamp*: Timestamp (in milliseconds)
- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *UE RLC statistics*: *N*-by-P cell, where *N* is the product of the number of UEs and the number of logical channels, and *P* is the number of statistics collected. Each row represents statistics of a logical channel in a UE. The last row contains the cumulative RLC statistics of the entire simulation.
- *gNB RLC statistics*: *N*-by-P cell, where *N* is the product of the number of UEs and the number of logical channels, and *P* is the number of statistics collected. Each row represents statistics of a logical channel of a UE at gNB. The last row contains the cumulative RLC statistics of the entire simulation.

Each row of the UE and gNB RLC statistics table represents a logical channel of a UE and contains:

- *RNTI*: Radio network temporary identifier of a UE.
- *LCID*: Logical channel identifier.
- *TxDataPDU*: Number of data PDUs sent by RLC to MAC layer.
- *TxDataBytes*: Number of data bytes sent by RLC to MAC layer.
- *ReTxDataPDU*: Number of data PDUs retransmitted by RLC to MAC layer.
- *ReTxDataBytes*: Number of data bytes retransmitted by RLC to MAC layer.
- *TxControlPDU*: Number of control PDUs sent by RLC to MAC layer.
- *TxControlBytes*: Number of control bytes sent by RLC to MAC layer.
- TxPacketsDropped: Number of RLC SDUs dropped by RLC due to Tx buffer overflow.
- *TxBytesDropped*: Number of bytes dropped by RLC due to Tx buffer overflow.
- *TimerPollRetransmitTimedOut*: Number of times the poll retransmit timer expired.
- *RxDataPDU*: Number of data PDUs received by RLC from MAC layer.
- *RxDataBytes*: Number of data bytes received by RLC from MAC layer.
- *RxDataPDUDropped*: Number of received data PDUs from MAC which are dropped by RLC layer.
- *RxDataBytesDropped*: Number of received data bytes from MAC which are dropped by RLC layer.
- *RxDataPDUDuplicate*: Number of duplicate PDUs received by RLC from MAC layer.
- *RxDataBytesDuplicate*: Number of duplicate data bytes received by RLC from MAC layer.
- *RxControlPDU*: Number of control PDUs received by RLC from MAC layer.
- *RxControlBytes*: Number of control bytes received by RLC from MAC layer.
- *TimerReassemblyTimedOut*: Number of times the reassembly timer expired.
- *TimerStatusProhibitTimedOut*: Number of times the status prohibit timer expired.

You can run the script NRPostSimVisualization to get a post-simulation visualization of logs. In the post-simulation script, you are provided with variable `isLogReplay`, which provides these options to visualize 'Resource Grid Allocation' and 'Channel Quality Visualization' figures.

- Set `isLogReplay` to `true` for a replay of the simulation logs.
- Set `isLogReplay` to `false` to analyze the details of a particular frame or a particular slot of a frame. In the 'Resource Grid Allocation' window, input the frame number and slot number to

visualize the resource assignment of the particular slot, if scheduling type is symbol-based. For slot-based scheduling, enter the frame number to visualize the resource assignment for the entire frame. The frame number entered here controls the frame number for 'Channel Quality Visualization' figure too.

```
if enableTraces
    % Read the logs and write them to MAT-files
    % Get the logs
    simulationLogs = cell(1,1);
    logInfo = struct('TimeStepLogs',[], 'SchedulingAssignmentLogs',[] ,'RLCLogs', []);
    [logInfo.TimeStepLogs] = getSchedulingLogs(simSchedulingLogger);
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger); % Scheduling assignmen
    logInfo.RLCLogs = getRLCLogs(simRLCLogger); % RLC statistics logs
    simulationLogs{1} = logInfo;

    save(simulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
    save(parametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-file
end
```

Further Exploration

You can use this example to further explore these options.

Custom scheduling

You can modify the existing scheduling strategy to implement a custom one. “Plug In Custom Scheduler in System-Level Simulation” on page 6-118 example explains how to create a custom scheduling strategy and plug it into system-level simulation.

Use 5G Toolbox™ physical layer

You can also switch from passthrough PHY layer to 5G Toolbox™ physical layer processing by creating PHY objects using hNRGNBPhy.m and hNRUEPhy.m. For more details, see 'gNB and UEs Setup' section of “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87.

Use RLC AM

You can also switch the operating mode of an RLC entity from UM to acknowledged mode (AM) by modifying the input structure fields `EntityType` and `SeqNumFieldLength` in the `configureLogicalChannel` function of hNRNode.m. Set the `EntityType` to 3 and `SeqNumFieldLength` to either 12 or 18. To explore the RLC AM functionality, add these fields to the input structiure.

- `PollRetransmitTimer`: Timer used by the transmitting side of an RLC AM entity in order to retransmit a poll
- `PollPDU`: Parameter used by the transmitting side of an RLC AM entity to trigger a poll based on number of PDUs
- `PollByte`: Parameter used by the transmitting side of an RLC AM entity to trigger a poll based on number of SDU bytes
- `MaxRetransmissions`: Maximum number of retransmissions corresponding to an RLC SDU, including its segments
- `StatusProhibitTimer`: Timer used by the receiving side of an RLC AM entity in order to prohibit frequent transmission of status PDUs

Based on the chosen scheduling strategy, this example demonstrates the assignment of UL and DL resources to multiple UEs by the gNB. UL and DL scheduling performance is analyzed based on runtime plots of throughput, goodput, resource share fairness, and pending buffer status of the UEs. A more thorough post-simulation analysis by using the saved logs gives a detailed picture of the operations happening on a per symbol or per slot basis.

References

[1] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “NR FDD Scheduling Performance Evaluation” on page 6-56

NR Cell Performance Evaluation with Physical Layer Integration

This example demonstrates the integration of high fidelity 5G Toolbox™ physical layer in a 5G New Radio (NR) node. The example models a 5G NR cell consisting of a set of user equipment (UE) connected to a gNB. The NR stack on the nodes includes radio link control (RLC), medium access control (MAC), and physical (PHY) layers. The example also models channel impairments that you can customize. For faster MAC focused simulations you can switch to passthrough PHY layer or you can integrate with a custom PHY layer.

Introduction

The example considers the following operations within gNB and UEs that facilitate uplink (UL) and downlink (DL) transmissions and receptions.

	Operations
gNB	• Run the scheduling algorithm to assign uplink and downlink resources • Send the uplink and downlink assignments to the UEs • Receive the physical uplink shared channel (PUSCH) transmissions from the UEs • Adhere to the downlink assignments for physical downlink shared channel (PDSCH) transmission • Receive the feedback of the PDSCHs from the UEs • Send the channel state information reference signals (CSI-RS) to the UEs
UEs	• Send the pending buffer status report to gNB • Receive the uplink and downlink assignments from the gNB • Adhere to the received uplink assignments from the gNB for PUSCH transmission • Receive the PDSCH transmission from the gNB • Send feedback for the received PDSCHs • Measure the channel quality on received CSI-RS and report it to gNB

The complete PUSCH or PDSCH packet is transmitted in the first symbol of its allocated symbol set. Receiver processes the packet in the symbol just after the last symbol in the allocated symbol set.

This example models:

- Slot based and symbol based DL and UL scheduling.
- Configurable subcarrier spacing resulting in different slot durations.
- Noncontiguous allocation of frequency-domain resources in terms of resource block groups (RBGs).
- Asynchronous adaptive hybrid automatic repeat request (HARQ) mechanism in UL and DL.
- PUSCH demodulation reference signal (DM-RS) and PDSCH DM-RS.
- DL channel quality measurement by UEs based on the CSI-RS received from gNB. By default, the CSI-RS resource element is transmitted in each slot for each resource block (RB) in DL bandwidth for all UEs. The same CSI-RS configuration is applicable to all the UEs. The example does not

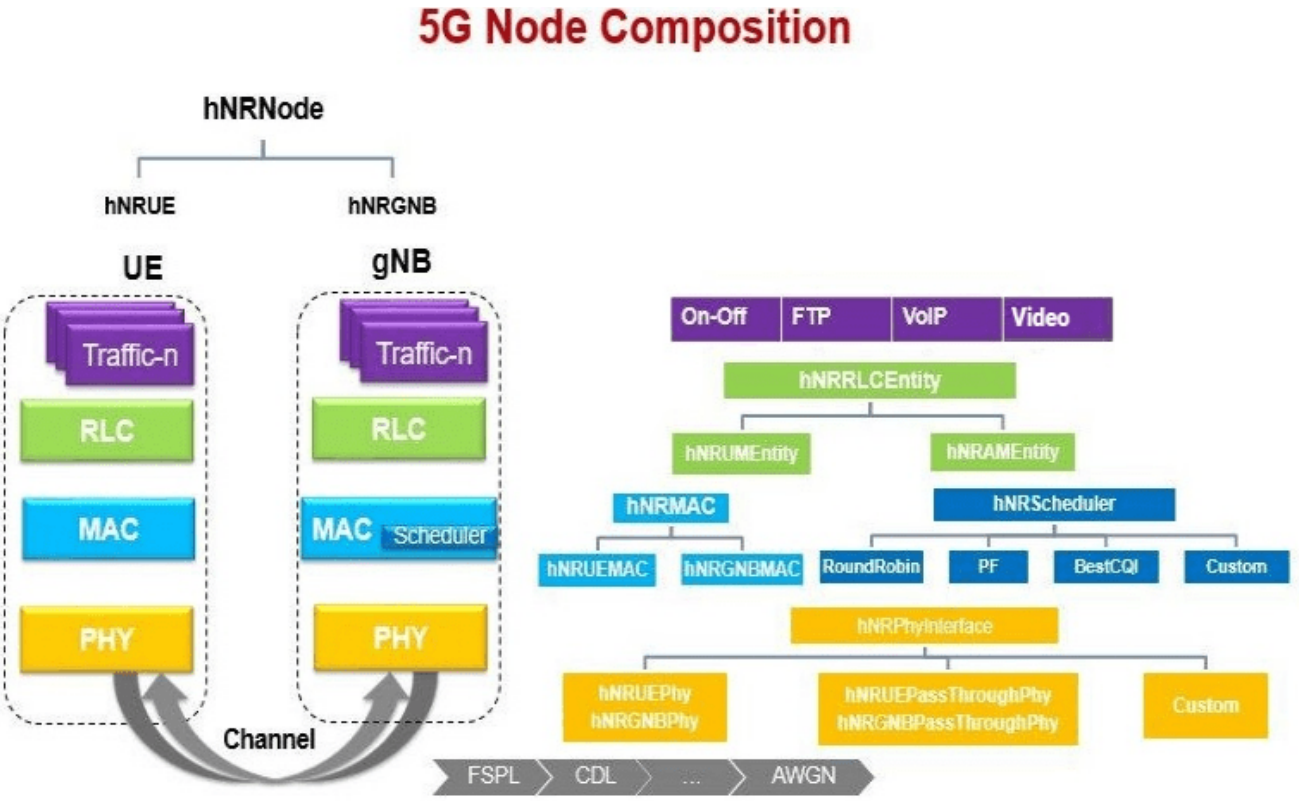
model the sounding reference signal (SRS) for measuring UL channel quality. UL channel quality is assumed to be the same as the DL channel quality measured on CSI-RS.

- Free space path loss (FSPL), additive white Gaussian noise (AWGN), and clustered delay line (CDL) propagation channel model.
- Single input single output (SISO) antenna configuration.
- Single bandwidth part across the whole carrier.

Control packets such as UL assignment, DL assignment, buffer status report (BSR), PDSCH feedback, and channel quality indicator (CQI) report, are assumed to be sent out of band, that is, without the need of resources for transmission and assured error-free reception.

NR Protocol Stack

A node (gNB or UE) is a composition of NR stack layers. The helper classes hNRGNB.m and hNRUE.m create gNB and UE nodes respectively, containing the RLC, MAC, and PHY layers.



RLC Layer

RLC operates in unacknowledged mode (UM) with a single logical channel (LCH). For the RLC layer, both hNRGNB.m and hNRUE.m use hNRUMEntity.m to implement the functionality of both the RLC transmitter and receiver.

MAC Layer

For the MAC layer, hNRGNB.m uses the helper class hNRGNBMAC.m to implement the gNB MAC functionality and hNRUE.m uses hNRUEMAC.m to implement the UE MAC functionality. gNB MAC

has UL and DL schedulers that assign UL and DL resources, respectively to the UEs. For more details about UL and DL scheduling to assign the PUSCH and the PDSCH resources, see the "NR FDD Scheduling Performance Evaluation" on page 6-56 example. Schedulers are implemented in hNRSchedulerRoundRobin.m (Round-robin strategy), hNRSchedulerProportionalFair.m (Proportional fair strategy), and hNRSchedulerBestCQI.m (Best CQI strategy) helper classes. All these schedulers are inherited from the base class hNRScheduler.m, which contains the core scheduling functionality.

PHY Layer and Channel Modeling

The example uses 5G Toolbox™ for PHY layer operations of UE and gNB. On the Tx side, the operations involve the physical layer processing of a transport block received from MAC and its transmission. On the Rx side, there is processing of received waveform and sending the decoded information to MAC. For more details on PDSCH and PUSCH processing chains, refer to the "NR PDSCH Throughput" on page 1-58 example and "NR PUSCH Throughput" on page 2-43 example, respectively. For the PHY layer, hNRGNB.m uses the helper class hNRGNBPhy.m to implement the gNB PHY layer functionality and hNRUE.m uses hNRUEPhy.m to implement the UE PHY layer functionality. For channel impairments, the example models FSPL, AWGN, and the CDL propagation channel model.

The example uses a lookup table to map the received signal-to-interference-plus-noise ratio (SINR) to CQI index for 0.1 block error rate (BLER). The lookup table corresponds to the CQI table as per 3GPP TS 38.214 Table 5.2.2.1-3. For more information about the process of generating this lookup table, refer to "5G NR Downlink CSI Reporting" on page 5-47 example.

MAC-PHY Interface

Following are the major interface calls between the MAC layer and PHY layer. For more details, refer to hNRPhyInterface.m.

- `txDataRequest:` The request from MAC to PHY to transmit either PDSCH (by gNB) or PUSCH (by UE). MAC calls this request at the start of Tx time. The PHY processing time is not modeled in this example.
- `rxDataRequest:` The request from MAC to PHY to receive either PUSCH (by gNB) or PDSCH (by UE). MAC calls this request at the start of Rx time.
- `dlControlRequest:` The request from MAC to PHY for non-data downlink transmissions or receptions. For gNB, this request is sent by gNB MAC for DL transmissions. For UE, it is sent by UE MAC for DL receptions. MAC sends the request at the start of a DL slot for all the scheduled DL transmission or receptions in the slot. This interface is used for all the DL transmission and receptions, except PDSCH. `txDataRequest` and `rxDataRequest` are used for PDSCH. In this example, gNB MAC uses this interface to send CSI-RS, and UE MAC uses it to receive CSI-RS.
- `registerMACInterfaceFcn:` The one-time setup call to register MAC callback functions at PHY. PHY uses the callbacks to send information up the stack to MAC. gNB PHY uses the callback to send decoded UL packets to MAC. UE PHY uses the callbacks to send decoded DL packets and DL channel quality measured on CSI-RS to MAC.

Pluggable PHY

You can plug and use different variations of PHY layer in your system. In a simulation run, all the nodes use the same variation of the PHY layer. The MAC is unaware of the type of PHY layer underneath, because the MAC uses the MAC-PHY interface to interact with the PHY layer. By default the example uses 5G Toolbox™ to model the PHY layer. To use a passthrough PHY layer, refer to "NR TDD Symbol Based Scheduling Performance Evaluation" on page 6-68 examples. A passthrough PHY layer does not do any physical layer processing of packets.

Scenario Configuration

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Configure simulation parameters in the `simParameters` structure.

```
rng('default'); % Reset the random number generator
simParameters = []; % Clear the simParameters variable
simParameters.NumFramesSim = 30; % Simulation time in terms of number of 10 ms frames
simParameters.SchedulingType = 0; % Set the value to 0 (slot based scheduling) or 1 (symbol based
```

Specify the number of UEs in the cell, assuming that UEs have sequential radio network temporary identifiers (RNTIs) from 1 to `simParameters.NumUEs`. If you change the number of UEs, ensure that the number of rows in `simParameters.UEPosition` parameter equals to the value of `simParameters.NumUEs`.

```
simParameters.NumUEs = 4;
% Assign position to the UEs assuming that the gNB is at (0, 0, 0). N-by-3
% matrix where 'N' is the number of UEs. Each row has (x, y, z) position of a
% UE (in meters)
simParameters.UEPosition = [100 0 0;
                            600 0 0;
                            1500 0 0;
                            2500 0 0];
% Validate the UE positions
validateattributes(simParameters.UEPosition,{'numeric'},{'nonempty','real','nrows',simParameters
```

Set the channel bandwidth to 5 MHz and subcarrier spacing (SCS) to 15 kHz as defined in 3GPP TS 38.104 Section 5.3.2. The complete bandwidth is assumed to be allotted for PUSCH or PDSCH.

```
simParameters.NumRBs = 25;
simParameters.SCS = 15; % kHz
simParameters.DLCarrierFreq = 2.635e9; % Hz
simParameters.ULCarrierFreq = 2.515e9; % Hz
% The UL and DL carriers are assumed to have symmetric channel
% bandwidth
simParameters.DLBandwidth = 5e6; % Hz
simParameters.ULBandwidth = 5e6; % Hz
```

Specify the transmit power and antenna gain.

```
simParameters.UETxPower = 23; % Tx power for all the UEs in dBm
simParameters.GNBTxPower = 29; % Tx power for gNB in dBm
simParameters.GNBRxGain = 10; % Receiver antenna gain at gNB
```

Specify the SINR to a CQI index mapping table for a BLER of 0.1.

```
simParameters.SINR90pc = [-5.46 -0.46 4.54 9.05 11.54 14.04 15.54 18.04 ...
    20.04 22.43 24.93 25.43 27.43 30.43 33.43];
```

Specify the scheduling strategy and the maximum limit on the RBs allotted for PDSCH and PUSCH. The transmission limit applies only to new transmissions and not to the retransmissions.

```
simParameters.SchedulerStrategy = 'PF'; % Supported scheduling strategies: 'PF', 'RR', and 'BestC
simParameters.RBAllocationLimitUL = 15; % For PUSCH
simParameters.RBAllocationLimitDL = 15; % For PDSCH
```

Logging and visualization configuration

The `CQIVisualization` and `RBVisualization` parameters control the display of the CQI visualization and the RB assignment visualization respectively. To enable these visualization plots, set these parameters to `true`.

```
simParameters.CQIVisualization = false;
simParameters.RBVisualization = false;
```

Set the `enableTraces` as `true` to log the traces. If the `enableTraces` is set to `false`, then `CQIVisualization` and `RBVisualization` are disabled automatically and traces are not logged in the simulation. To speed up the simulation, set the `enableTraces` to `false`.

```
enableTraces = true;
```

The example updates the metrics plots periodically. Set the number of updates during the simulation.

```
simParameters.NumMetricsSteps = 20;
```

Write the logs to MAT-files. The example uses these logs for post-simulation analysis and visualization.

```
parametersLogFile = 'simParameters'; % For logging the simulation parameters
simulationLogFile = 'simulationLogs'; % For logging the simulation traces
simulationMetricsFile = 'simulationMetrics'; % For logging the simulation metrics

% Enable packet capture (PCAP)
simParameters.PCAPLogging = false; % Set the value to true to enable packet capture for UEofInter
simParameters.UEofInterest = 1; % Log the packets of UE with this RNTI
```

Application traffic configuration

Set the DL and UL application traffic pattern for UEs.

```
dlAppDataRate = 16e4*ones(simParameters.NumUEs,1); % DL application data rate in kilo bits per se
ulAppDataRate = 16e4*ones(simParameters.NumUEs,1); % UL application data rate in kbps
% Validate the DL application data rate
validateattributes(dlAppDataRate,{'numeric'},{'nonempty','vector','numel',simParameters.NumUEs,'
% Validate the UL application data rate
validateattributes(ulAppDataRate, {'numeric'}, {'nonempty','vector','numel',simParameters.NumUEs
```

Derived Parameters

Compute the derived parameters based on the primary configuration parameters specified in the previous section and set some example-specific constants.

```
simParameters.DuplexMode = 0; % FDD
simParameters.NCellID = 1; % Physical cell ID
simParameters.Position = [0 0 0]; % Position of gNB in (x,y,z) coordinates
```

Specify the CSI-RS resource configuration, assuming that all UEs measure channel quality on the same CSI-RS resource.

```
csirsConfig = nrCSIRSConfig('NID',simParameters.NCellID,'NumRB',simParameters.NumRBs,'RowNumber'
simParameters.CSIRSConfig = {csirsConfig};
```

Specify the CSI report configuration.

```
csiReportConfig = struct('SubbandSize',8,'CQIMode','Subband');
simParameters.CSIReportConfig = {csiReportConfig};
```

Configure the channel model.

```
channelModelUL = cell(1, simParameters.NumUEs);
channelModelDL = cell(1, simParameters.NumUEs);
waveformInfo = nrOFDMInfo(simParameters.NumRBs,simParameters.SCS);
for ueIdx = 1:simParameters.NumUEs
    % Configure the uplink channel model
    channel = nrCDLChannel;
    channel.DelayProfile = 'CDL-C';
    channel.DelaySpread = 300e-9;
    channel.Seed = 73 + (ueIdx - 1);
    channel.CarrierFrequency = simParameters.ULCarrierFreq;
    channel.TransmitAntennaArray.Size = [1 1 1 1 1];
    channel.ReceiveAntennaArray.Size = [1 1 1 1 1];
    channel.SampleRate = waveformInfo.SampleRate;
    channelModelUL{ueIdx} = channel;

    % Configure the downlink channel model
    channel = nrCDLChannel;
    channel.DelayProfile = 'CDL-C';
    channel.DelaySpread = 300e-9;
    channel.Seed = 73 + (ueIdx - 1);
    channel.CarrierFrequency = simParameters.DLCarrierFreq;
    channel.TransmitAntennaArray.Size = [1 1 1 1 1];
    channel.ReceiveAntennaArray.Size = [1 1 1 1 1];
    channel.SampleRate = waveformInfo.SampleRate;
    channelModelDL{ueIdx} = channel;
end
```

Set the PUSCH preparation time for UEs. The gNB ensures that PUSCH assignment is received at UEs `PUSCHPrepTime` ahead of the transmission time.

```
simParameters.PUSCHPrepTime = 200; % In microseconds
```

Compute the number of slots in the simulation.

```
numSlotsSim = (simParameters.NumFramesSim * 10 * simParameters.SCS)/15;
```

Set the interval at which the example updates metrics visualization in terms of number of slots. Because this example uses a time granularity of one slot, the `MetricsStepSize` field must be an integer.

```
simParameters.MetricsStepSize = ceil(numSlotsSim / simParameters.NumMetricsSteps);
```

Specify one logical channel for each UE, and set the logical channel configuration for all nodes (UEs and gNBs) in the example.

```
numLogicalChannels = 1; % Only 1 logical channel is assumed in each UE in this example
% Logical channel id (logical channel ID of data radio bearers starts from 4)
simParameters.LCHConfig.LCID = 4;
```

Specify the RLC entity type in the range [0, 3]. The values 0, 1, 2, and 3 indicate RLC UM unidirectional DL entity, RLC UM unidirectional UL entity, RLC UM bidirectional entity, and RLC AM entity, respectively.

```
simParameters.RLCConfig.EntityType = 2;
```

Create RLC channel configuration structure.

```
rlcChannelConfigStruct.LCGID = 1; % Mapping between logical channel and logical channel group ID
rlcChannelConfigStruct.Priority = 1; % Priority of each logical channel
rlcChannelConfigStruct.PBR = 8; % Prioritized bitrate (PBR), in kilobytes per second, of each log
rlcChannelConfigStruct.BSD = 10; % Bucket size duration (BSD), in ms, of each logical channel
rlcChannelConfigStruct.EntityType = simParameters.RLCConfig.EntityType;
rlcChannelConfigStruct.LogicalChannelID = simParameters.LCHConfig.LCID;
```

Set the mapping type as per the configured scheduling type.

```
if ~isfield(simParameters, 'SchedulingType') || simParameters.SchedulingType == 0 % If no schedu
    simParameters.PUSCHMappingType = 'A';
    simParameters.PDSCHMappingType = 'A';
else % Symbol based scheduling
    simParameters.PUSCHMappingType = 'B';
    simParameters.PDSCHMappingType = 'B';
end
```

gNB and UEs Setup

Create the gNB and UE objects, initialize the channel quality information for UEs, and set up the logical channel at gNB and UE. The helper classes hNRGNB.m and hNRUE.m create gNB node and UE node respectively, containing the RLC, MAC and PHY layers.

```
gNB = hNRGNB(simParameters); % Create gNB node
% Create scheduler
switch(simParameters.SchedulerStrategy)
    case 'RR' % Round-robin scheduler
        scheduler = hNRSchedulerRoundRobin(simParameters);
    case 'PF' % Proportional fair scheduler
        scheduler = hNRSchedulerProportionalFair(simParameters);
    case 'BestCQI' % Best CQI scheduler
        scheduler = hNRSchedulerBestCQI(simParameters);
end
addScheduler(gNB, scheduler); % Add scheduler to gNB
simParameters.ChannelModel = channelModelUL;
gNB.PhyEntity = hNRGNBPhy(simParameters); % Create the PHY layer instance
configurePhy(gNB, simParameters); % Configure the PHY layer
setPhyInterface(gNB); % Set the interface to PHY layer

% Create the set of UE nodes
UEs = cell(simParameters.NumUEs, 1);
for ueIdx=1:simParameters.NumUEs
    ueParam = simParameters;
    ueParam.Position = simParameters.UEPosition(ueIdx, :); % Position of the UE
    ueParam.ChannelModel = channelModelDL{ueIdx};
    ueParam.CSIReportConfig = csiReportConfig;
    UEs{ueIdx} = hNRUE(ueParam,ueIdx);
    UEs{ueIdx}.PhyEntity = hNRUEPhy(ueParam,ueIdx); % Create the PHY layer instance
    configurePhy(UEs{ueIdx},ueParam); % Configure the PHY layer
    setPhyInterface(UEs{ueIdx}); % Set up the interface to PHY layer

    % Setup logical channel at gNB for the UE
    configureLogicalChannel(gNB,ueIdx,rlcChannelConfigStruct);
    % Setup logical channel at UE
```

```
    configureLogicalChannel(UEs{ueIdx},ueIdx,rlcChannelConfigStruct);

    % Create an object for On-Off network traffic pattern and add it to the
    % specified UE. This object generates the uplink data traffic on the UE
    ulApp = networkTrafficOnOff('GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim*10e-3,'OffTime', 0,'DataRate',ulAppDataRate(ueIdx));
    UEs{ueIdx}.addApplication(ueIdx,simParameters.LCHConfig.LCID,ulApp);

    % Create an object for On-Off network traffic pattern for the specified
    % UE and add it to the gNB. This object generates the downlink data
    % traffic on the gNB for the UE
    dlApp = networkTrafficOnOff('GeneratePacket', true, ...
        'OnTime', simParameters.NumFramesSim*10e-3,'OffTime',0,'DataRate',dlAppDataRate(ueIdx));
    gNB.addApplication(ueIdx,simParameters.LCHConfig.LCID,dlApp);
end
```

Enable PCAP logging.

```
if simParameters.PCAPLogging
    % To generate unique file name for every simulation run
    ueCapturefileName = strcat('CellID-',num2str(simParameters.NCellID),'_ue-',num2str(simParame
    enablePacketLogging(UEs{simParameters.UEofInterest}.PhyEntity,ueCapturefileName);

    % Uncomment the below code to enable packet capture at gNB
    % gnbCapturefileName = strcat('CellID-',num2str(simParameters.NCellID),'_gNB-',num2str(now))
    % enablePacketLogging(gNB.PhyEntity,gnbCapturefileName);
end
```

Simulation

```
% Initialize wireless network simulator
nrNodes = [{gNB}; UEs];
networkSimulator = hWirelessNetworkSimulator(nrNodes);
```

Create objects to log MAC and PHY traces.

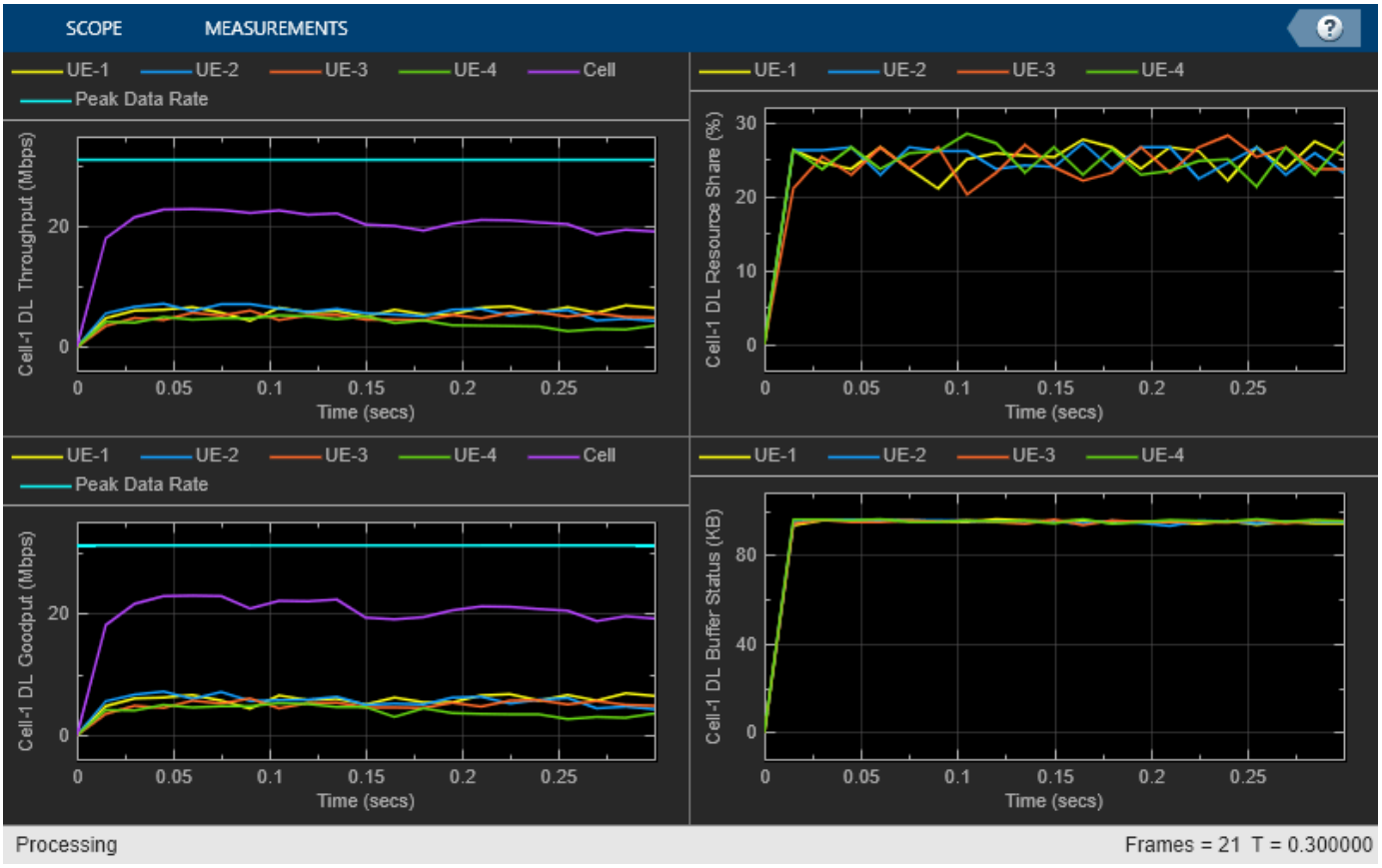
```
if enableTraces
    % Create an object for MAC traces logging
    simSchedulingLogger = hNRSchedulingLogger(simParameters,networkSimulator,gNB,UEs);

    % Create an object for PHY traces logging
    simPhyLogger = hNRPhyLogger(simParameters,networkSimulator,gNB,UEs);

    % Create an object for CQI and RB grid visualization
    if simParameters.CQIVisualization || simParameters.RBVisualization
        gridVisualizer = hNRGridVisualizer(simParameters,'MACLogger',simSchedulingLogger);
    end
end
```

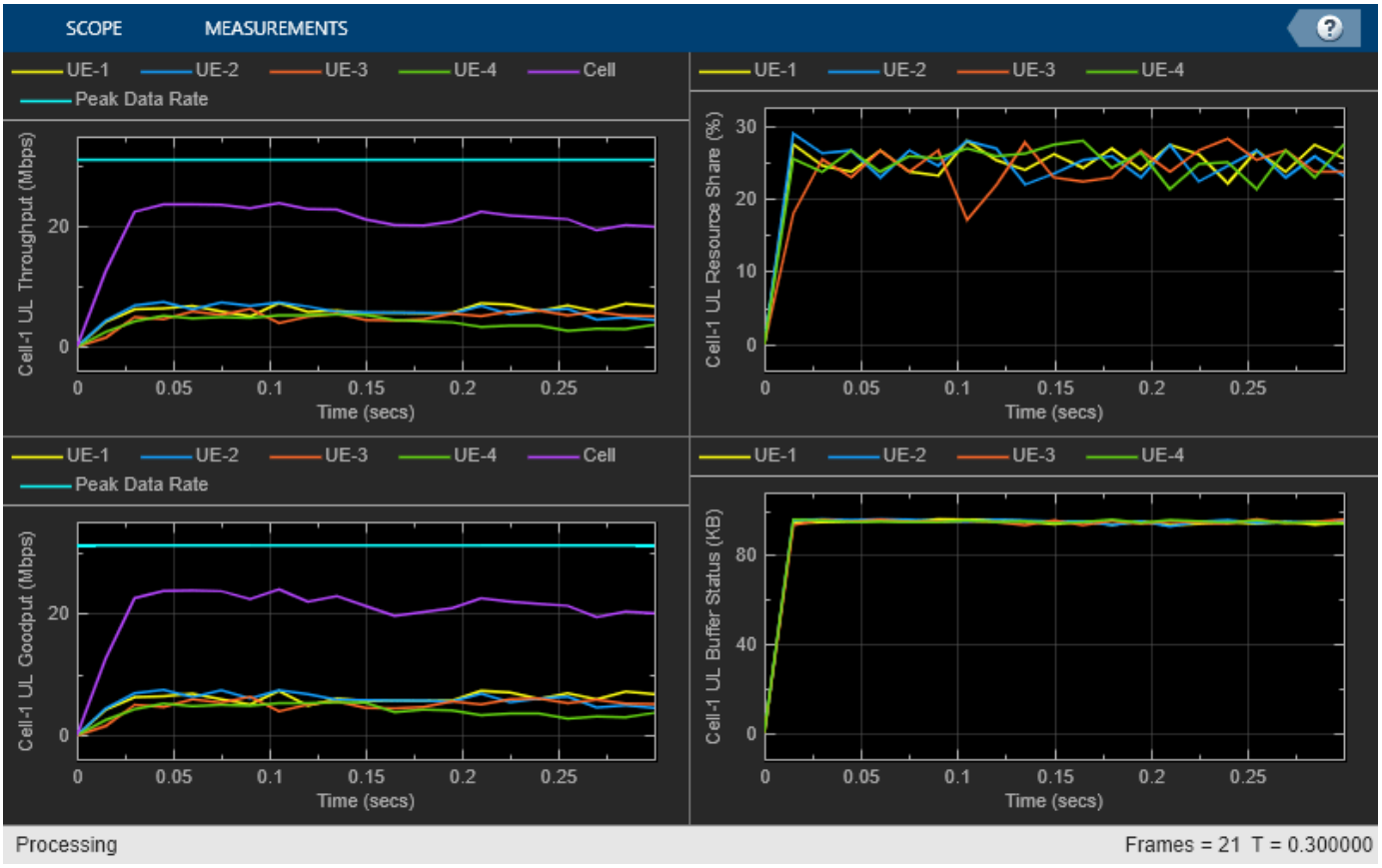
Create an object for MAC and PHY metrics visualization.

```
metricsVisualizer = hNRMetricsVisualizer(simParameters,'EnableSchedulerMetricsPlots',true,'Enable
```

Run the simulation for the specified `NumFramesSim` frames.

```
% Calculate the simulation duration (in seconds) from 'NumFramesSim'
simulationTime = simParameters.NumFramesSim * 1e-2;
% Run the simulation
run(networkSimulator,simulationTime);
```


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Cell
Peak Data Rate
Cell-1 DL Throughput (Mbps)
Cell-1 DL Resource Share (%)
Cell-1 DL Goodput (Mbps)
Cell-1 DL Buffer Status (KB)
Time (secs)
Processing
Frames = 21 T = 0.300000


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Cell
Peak Data Rate
Cell-1 UL Throughput (Mbps)
Cell-1 UL Resource Share (%)
Cell-1 UL Goodput (Mbps)
Cell-1 UL Buffer Status (KB)
Time (secs)
Processing
Frames = 21 T = 0.300000

Get the simulation metrics and save it in a MAT-file. The simulation metrics are saved in a MAT-file with the file name as `simulationMetricsFile`.

```
metrics = getMetrics(metricsVisualizer);
save(simulationMetricsFile,'metrics');
```

At the end of the simulation, the achieved value for system performance indicators is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (UL and DL), achieved spectral efficiency (UL and DL), and BLER observed for UEs (DL and UL). The peak values are calculated as per 3GPP TR 37.910.

```
displayPerformanceIndicators(metricsVisualizer)
```

```
Peak UL Throughput: 31.11 Mbps. Achieved Cell UL Throughput: 21.40 Mbps
Achieved UL Throughput for each UE: [6.18        6.02        5.04        4.17]
Achieved Cell UL Goodput: 21.28 Mbps
Achieved UL Goodput for each UE: [6.13        5.98        5.04        4.13]
Peak UL spectral efficiency: 6.22 bits/s/Hz. Achieved UL spectral efficiency for cell: 4.26 bits/

Peak DL Throughput: 31.11 Mbps. Achieved Cell DL Throughput: 20.92 Mbps
Achieved DL Throughput for each UE: [5.94        5.87        5.02        4.09]
Achieved Cell DL Goodput: 20.70 Mbps
Achieved DL Goodput for each UE: [5.94        5.72        5.02        4.03]
Peak DL spectral efficiency: 6.22 bits/s/Hz. Achieved DL spectral efficiency for cell: 4.14 bits/

Block error rate for each UE in the uplink direction: [0.006       0.007          0       0.006]
Block error rate for each UE in the downlink direction: [0       0.032          0       0.019]
```

Simulation Visualization

The five types of run-time visualization shown are:

- *Display of CQI values for UEs over the channel bandwidth*: For details, see the 'Channel Quality Visualization' figure.
- *Display of resource grid assignment to UEs*: The 2-D time-frequency grid updates every 10 ms (frame length) and shows the RB allocation to the UEs in the previous frame. The HARQ process for the PUSCH and PDSCH assignments is also shown alongside with the RNTI of the UEs. New transmissions are shown in black and retransmissions are shown in blue using the HARQ process ID of each UE, a retransmission assignment can be mapped to its previously failed transmission. You can enable this visualization in the 'Scenario Configuration' section. For details, see the 'Resource Grid Allocation' figure.
- *Display of UL scheduling metrics plots*: The four plots displayed in 'Uplink Scheduler Performance Metrics' figure represent: UL throughput (per UE and cell), UL goodput (per UE and cell), resource share percentage among UEs (out of the total UL resources) to convey the fairness of scheduling, and pending UL buffer-status of UEs to show whether UEs are getting sufficient resources. The maximum achievable data rate value for UL throughput is shown with a dashed line in throughput and goodput plots. The performance metrics plots update for every `metricsStepSize` slots. The UL throughput plot shows the rate of MAC data transmission for each UE and across the cell. The calculation includes transport block size (TBS) of new transmissions as well as the TBS of retransmissions. The units are in megabits per second (Mbps). The goodput (in Mbps) calculation only considers new transmission MAC data.
- *Display of DL scheduling metrics plots*: Like UL metrics plots, 'Downlink Scheduler Performance Metrics' displays corresponding subplots for DL direction. The performance metrics plots update for every `metricsStepSize` slots.
- *Display of DL and UL Block Error Rates:* The two sub-plots displayed in 'Block Error Rate (BLER) Visualization' shows the BLER (for each UE) observed in the uplink and downlink directions, as the simulation progresses. The plot is updated every `metricsStepSize` slots.

Simulation Logs

The parameters used for simulation and the simulation logs are saved in MAT-files for post-simulation analysis and visualization. The simulation parameters are saved in a MAT-file with the file name as the value of configuration parameter `parametersLogFile`. The per time step logs, scheduling assignment logs, and BLER logs are saved in the MAT-file `simulationLogFile`. After the simulation, open the file to load `DLTimeStepLogs`, `ULTimeStepLogs` `SchedulingAssignmentLogs`, and `RLCLogs` in the workspace.

Time step logs: Both the DL and UL time step logs follow the same format. The table shows sample time step entries.

Timestamp	Frame	Slot	RBG Allocation Bitmap	MCS	HARQ Process	NDI	Tx Type	CQI for UEs	HARQ NDI Status	Throughput Bytes	Goodput Bytes	Buffer Status of UEs
50	5	0	.....	.....	.....	....	.....	.....	.....	.....	.....	.....
51	5	1	[0 0 1 1 0 1 0 1 0 1 0 0 0] [1 1 1 0 0 0 0 0 0 0 1 0 0] [0 0 0 0 1 0 1 0 1 0 0 1 1] [0 0 0 0 0 0 0 0 0 0 0 0 0]	[10 12 8 -1]	[0 3 6 -1]	[0 0 1 -1]	['newTx', 'newTx', 'reTx', 'noTx']	[5 8 9 8 7 8 9 8 9 11 12 . . . . .] [5 7 3 4 9 5 9 7 9 10 15 . . . . .] [3 9 9 6 9 4 9 8 9 9 13 . . . . .] [9 7 7 3 7 8 5 8 8 15 9 . . . . .]	[1 0 0 1 0 1 0 . .] [1 1 0 1 0 1 0 . .] [1 0 0 1 0 1 1 . .] [1 0 0 1 0 1 0 . .]	[46 54 38 0]	[46 54 0 0]	[299480 135671 77567 137070]
52	5	2	.....	.....	.....	....	.....	.....	.....	.....	.....	.....

Each row of the table represents a slot and contains the following information:

- *Timestamp*: Time (in milliseconds) since the start of simulation.

- *Frame*: Frame number.
- *Slot*: Slot number in the frame.
- *RBG Allocation Bitmap*: *N-by-P* bitmap matrix, where *N* is the number of UEs and *P* is the number of RBGs. If an RBG is assigned to a particular UE, the corresponding bit is set to 1. For example, [0 0 1 1 0 1 0 1 0 1 0 0 0; 1 1 0 0 0 0 0 0 0 0 1 0 0; 0 0 0 0 1 0 1 0 1 0 0 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0] means that the bandwidth has 13 RBGs and UE-1 is assigned RBG indices: 2, 3, 5, 7 and 9; UE-2 is assigned the RBG indices 0, 1 and 10; UE-3 is assigned the RBG indices 4, 6, 8, 11 and 12 and UE-4 is not assigned any RBG.
- *MCS*: Row vector of length *N,* where *N* is the number of UEs. Each value corresponds to the modulation and coding scheme (MCS) index for the transmission. For example, [10 12 8 -1] means that only UE-1, UE-2, and UE-3 are assigned resources for this slot and use MCS values 10, 12, and 8, respectively.
- *HARQ Process*: Row vector of length *N,* where *N* is the number of UEs. The value is the HARQ process ID used by UE for the transmission. For example, [0 3 6 -1] means that only UE-1, UE-2, and UE-3 are assigned resources for this slot and use the HARQ process IDs 0, 3, and 6, respectively.
- *NDI*: Row vector of length *N,* where *N* is the number of UEs. The value is the NDI flag value in the assignment for transmission. For example, [0 0 1 -1] means that only UE-1, UE-2, and UE-3 are assigned resources for this slot and use the NDI flag values (which determine whether a new transmission or a retransmission is used) 0, 0, and 1, respectively.
- *Tx Type*: Tx Type specifies the transmission type (new transmission or retransmission). Row vector of length *N,* where *N* is the number of UEs. Possible values are either 'newTx', 'reTx', or 'noTx'. 'noTx' means that the UE is not allocated resources. For example, ['newTx' 'newTx' 'reTx' 'noTx'] means that only UE-1, UE-2, and UE-3 are assigned resources for this slot. The assigned resources correspond to a new transmission for UE-1 and UE-2 and a retransmission for UE-3.
- *CQI for UEs*: *N-by-P* matrix, where *N* is the number of UEs and *P* is the number of RBs in the bandwidth. A matrix element at position *(i, j)* corresponds to the CQI value for UE with RNTI *i* at RB *j*.
- *HARQ NDI Status*: *N-by-P* matrix, where *N* is the number of UEs and *P* is the number of HARQ processes on UEs. A matrix element at position *(i, j)* is the last received NDI flag at UE *i* for HARQ process ID *j*. For new transmissions, this value and the NDI flag in the assignment must toggle. For example, in slot 1 of frame 5 described in the scheduling log, UE-1 uses the HARQ ID 0 and the last NDI flag value for HARQ ID 0 at UE-1 is 1. To indicate a new transmission, the NDI flag values changes to 0 in the assignment.
- *Throughput Bytes*: Row vector of length *N,* where *N* is the number of UEs. The values represent MAC bytes transmitted per UE in this slot.
- *Goodput Bytes*: Row vector of length *N,* where *N* is the number of UEs. The values represent new transmission MAC bytes transmitted per UE in this slot.
- *Buffer Status of UEs*: Row vector of length *N,* where *N* is the number of UEs. The values represent the amount of pending buffers per UE.

Scheduling assignment logs: Information of all the scheduling assignments and the related information is logged in this file. The table shows sample log entries.

'RNTI'	'Frame'	'Slot'	'Grant type'	'RBG Allocation Map'	'Start Sym'	'Num Sym'	'MCS'	'HARQ ID'	'NDI Flag'	'RV'	'Tx Type'	'Feedback Slot Offset (DL grants only)'	'CQI on RBs'
3	0	8	'DL'	'[1 0 0 0 0 0 0 1 1 0 1 1 1]'	0	14	10	2	1	0	'newTx'	2	'[3 7 7 2 2 5 10 4 6 3 8 3 6 7 9 10 6 2...
4	0	8	'DL'	'[0 1 1 1 1 1 1 0 0 1 0 0 0]'	0	14	9	3	1	0	'newTx'	2	'[4 2 3 7 5 4 9 6 6 10 3 8 8 4 6 1 1 6 ...
1	0	9	'UL'	'[1 1 1 1 1 1 1 0 0 0 0 0 0]'	0	14	16	0	1	0	'newTx'	'NA'	'[13 14 2 14 10 2 5 9 15 15 3 15 15 8...
2	0	9	'UL'	'[0 0 0 0 0 0 0 0 1 0 1 1 1]'	0	14	12	0	1	0	'newTx'	'NA'	'[10 9 5 8 3 9 1 4 1 2 10 9 4 12 1 6 5 ...
3	0	9	'UL'	'[0 0 0 0 0 0 0 1 0 0 0 0 0]'	0	14	15	0	1	0	'newTx'	'NA'	'[3 7 7 2 2 5 10 4 6 3 8 3 6 7 9 10 6 2...
4	0	9	'UL'	'[0 0 0 0 0 0 0 0 0 1 0 0 0]'	0	14	15	0	1	0	'newTx'	'NA'	'[4 2 3 7 5 4 9 6 6 10 3 8 8 4 6 1 1 6 ...

BLER logs: Block error information observed in the uplink and downlink directions are logged in this file. This table shows the sample log entries.

'Timestamp'	'Frame Number'	'Slot Number'	'Number of Erroneous Packets(DL)'	'Number of Packets(DL)'	'Number of Erroneous Packets(UL)'	'Number of Packets(UL)'
0	0	0	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
1	0	1	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
2	0	2	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]	[0;0;0;0]
3	0	3	[0;0;0;0]	[1;1;0;0]	[0;0;0;0]	[0;0;0;0]
4	0	4	[0;0;1;1]	[0;0;1;1]	[0;0;0;0]	[0;0;0;0]
5	0	5	[0;0;0;0]	[1;1;0;0]	[0;0;0;0]	[0;0;0;0]
6	0	6	[0;0;0;0]	[1;1;0;0]	[0;0;0;0]	[0;0;0;0]
7	0	7	[0;0;0;0]	[1;1;0;0]	[0;0;0;0]	[0;0;0;0]
8	0	8	[0;0;0;0]	[1;1;0;1]	[0;0;0;0]	[1;1;0;0]
9	0	9	[0;0;0;0]	[0;1;1;0]	[0;0;0;0]	[0;0;1;1]

Each row of the log represents one slot. The column contains the information vector of length equal to the number of UEs. Information about a UE is at an index equal to its RNTI.

```
if enableTraces
    simulationLogs = cell(1,1);
    % Read the logs and save them in MAT-files
    if simParameters.DuplexMode == 0 % FDD
        logInfo = struct('DLTimeStepLogs', [], 'ULTimeStepLogs', [], 'SchedulingAssignmentLogs',
        [logInfo.DLTimeStepLogs, logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger)
    else % TDD
        logInfo = struct('TimeStepLogs', [], 'SchedulingAssignmentLogs', [], 'BLERLogs', [], 'Avg
        logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger);
    end
    [logInfo.BLERLogs, logInfo.AvgBLERLogs] = getBLERLogs(simPhyLogger); % BLER logs
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger); % Scheduling assignment
    simulationLogs{1} = logInfo;
    save(parametersLogFile, 'simParameters'); % Save simulation parameters in a MAT-file
    save(simulationLogFile, 'simulationLogs'); % Save simulation logs in a MAT-file
end
```

To obtain a post-simulation visualization of logs, you can run the script NRPostSimVisualization. The variable `isLogReplay` in the post-simulation script offers options to visualize the Resource Grid Allocation and Channel Quality Visualization figures. Specify `isLogReplay` as one of these options.

- true — Use this option to replay the Resource Grid Allocation and Channel Quality Visualization figures.
- false — Use this option to analyze the details of a particular simulation frame. In the Resource Grid Allocation window, input the frame number to visualize the resource assignment for the entire frame. The Channel Quality Visualization figure also uses this frame number.

Further Exploration

You can use this example to further explore these options.

Custom scheduling

You can modify the existing scheduling strategy to implement a custom one. “Plug In Custom Scheduler in System-Level Simulation” on page 6-118 example explains how to create a custom scheduling strategy and plug it into system-level simulation.

Use passthrough physical layer

For MAC focused simulations, you can use the passthrough PHY layer by installing passthrough PHY layer object on nodes. For gNB create an object of type hNRGNBPassthroughPhy, and for UE create an object of type hNRUEPassthroughPhy. For details, see 'gNB and UEs setup' section of “NR TDD Symbol Based Scheduling Performance Evaluation” on page 6-68 example.

Based on described simulation parameters, the example evaluates the performance of the system measured in terms of various metrics. Different visualizations show the run time performance of the system. A more thorough post-simulation analysis by using the saved logs gives a detailed picture of the operations on a per slot basis.

Use RLC AM

You can also switch the operating mode of an RLC entity from UM to acknowledged mode (AM) by modifying the input structure fields `EntityType` and `SeqNumFieldLength` in the `configureLogicalChannel` function of hNRNode.m. For more details, see the 'Further Exploration' section of “NR TDD Symbol Based Scheduling Performance Evaluation” on page 6-68.

References

[1] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 3GPP TS 38.323. “NR; Packet Data Convergence Protocol (PDCP) specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[6] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[7] 3GPP TR 37.910. “Study on self evaluation towards IMT-2020 submission.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “NR FDD Scheduling Performance Evaluation” on page 6-56
- “NR Intercell Interference Modeling” on page 6-102

NR Intercell Interference Modeling

This example shows the impact on network performance due to downlink (DL) intercell interference caused by nearby cells. The example models a 5G New Radio (NR) network of multiple cells operating in the same frequency band. Each cell has a gNB placed at the center of the cell that serves a set of user equipment (UE). The NR stack on the nodes includes radio link control (RLC), medium access control (MAC), and physical layers (PHY).

Introduction

This example models:

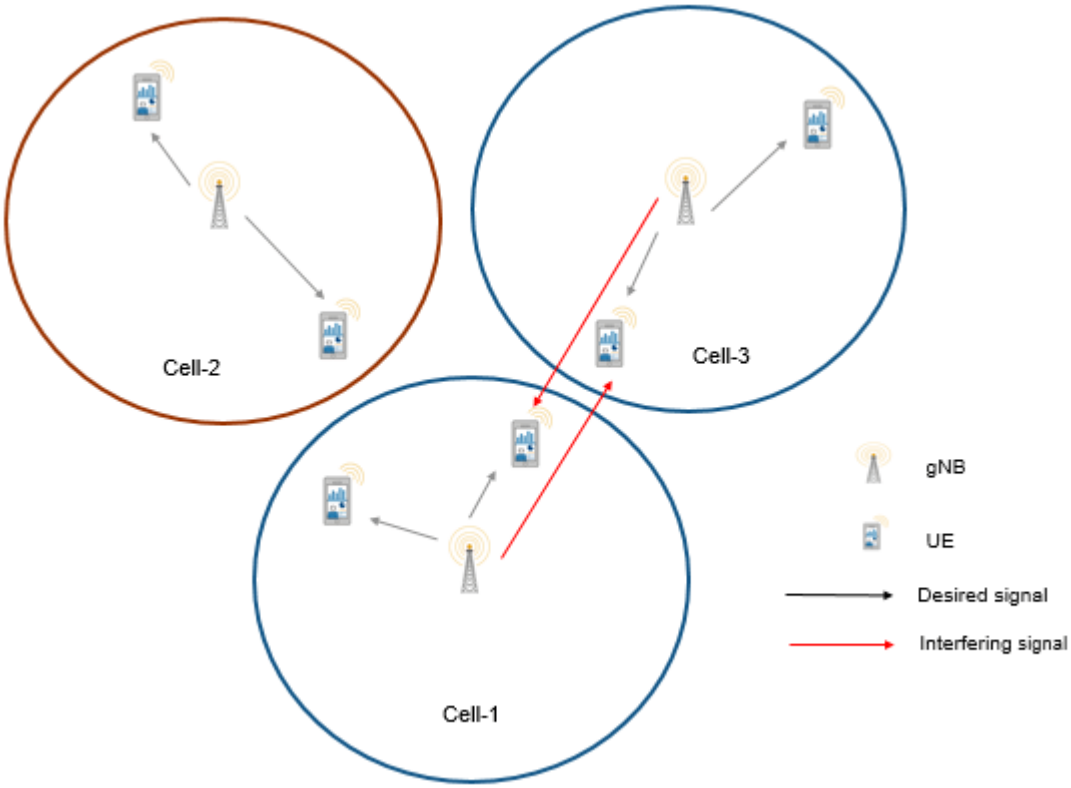
- Co-channel intercell interference.
- Slot-based round robin scheduling of physical downlink shared channel (PDSCH) resources.
- Free space path loss (FSPL) model.
- Single input single output (SISO) antenna configuration.
- Link-to-system-mapping-based abstract PHY.

In this example, you consider the control packets, such as DL assignments, PDSCH feedback, and channel quality indicator (CQI) report, transmitted as out-of-band. Out-of-band refers to the transmission without the need for resources. It also refers to assured error-free reception.

Co-Channel Interference

Co-channel cells are the NR cells operating on the same frequency. They can interfere with each other.

Consider this sample network topology consisting of 3 cells. Cell-1 and Cell-3 operate on the same frequency band. Cell-2 operates on a different frequency band and does not interfere with Cell-1 or Cell-3.



Scenario setup

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Create a wireless network simulator.

```
rng("default") % Reset the random number generator
numFrameSimulation = 20; % Simulation time in terms of number of 10 ms frames
networkSimulator = wirelessNetworkSimulator.init;
```

Specify the positions of 3 gNBs and the gNB of interest for visualizing metrics.

```
gNBPositions = [1700 600 0; 3000 600 0; 2500 2000 0];
gNBOfInterestIdx = 3; % Specify a value between 1 and number of gNBs
```

Create the gNBs. Specify the position, carrier frequency, channel bandwidth, subcarrier spacing, transmit power, and receive gain of each gNB. Each gNB operates one NR cell.

```
gNBs = nrGNB(Position=gNBPositions,CarrierFrequency=2.5e9, ...
    ChannelBandwidth=10e6,SubcarrierSpacing=30e3,TransmitPower=32,ReceiveGain=11);
```

Set the scheduler parameter `ResourceAllocationType` by using the `configureScheduler` function.

```
for gNBIdx = 1:length(gNBs)
    % Resource allocation type value 0 indicate noncontiguous allocation of
    % frequency-domain resources in terms of RBGs
    configureScheduler(gNBs(gNBIdx),ResourceAllocationType=0)
end
```

Generate the positions of UEs in each cell.

```
numCells = length(gNBs);
cellRadius = 500; % Radius of each cell (in meters)
numUEsPerCell = 4;
uePositions = generateUEPositions(cellRadius,gNBPositions,numUEsPerCell);
```

Create the UEs and connect them to a gNB. Configure full buffer traffic in the DL direction.

```
UEs = cell(numCells,1);
for cellIdx = 1:numCells
    ueNames = "UE-" + (1:size(uePositions{cellIdx},1));
    UEs{cellIdx} = nrUE(Name=ueNames,Position=uePositions{cellIdx},ReceiveGain=11);
    connectUE(gNBs(cellIdx),UEs{cellIdx},FullBufferTraffic="DL")
end
```

Add gNBs and UEs to the network simulator.

```
addNodes(networkSimulator,gNBs);
for cellIdx = 1:numCells
    addNodes(networkSimulator,UEs{cellIdx})
end
```

Get the cell ID for the gNB of interest. All the visualizations and metrics are shown for this cell.

```
cellOfInterest = gNBs(gNBOfInterestIdx).ID;
```

Set the `enableTraces` as `true` to log the traces. If the `enableTraces` is set to `false`, then traces are not logged in the simulation. To speed up the simulation, set the `enableTraces` to `false`.

```
enableTraces = true;
```

Create objects to log MAC and PHY traces.

```
linkDir = 0; % Indicates DL
if enableTraces
    simSchedulingLogger = cell(numCells,1);
    simPhyLogger = cell(numCells,1);

    for cellIdx = 1:numCells
        % Create an object for MAC DL scheduling traces logging
        simSchedulingLogger{cellIdx} = helperNRSchedulingLogger(numFrameSimulation, ...
            gNBs(cellOfInterest),UEs{cellOfInterest},linkDir);

         % Create an object for PHY layer traces logging
        simPhyLogger{cellIdx} = helperNRPhyLogger(numFrameSimulation,gNBs(cellOfInterest), ...
            UEs{cellOfInterest});
    end
end
```

Update the output metrics plots periodically, specifying `numMetricsSteps` updates within the simulation. `numMetricsSteps` must be less than or equal to number of slots in simulation.

```
numMetricsSteps = numFrameSimulation;
```

Create an object for MAC and PHY metrics visualization.

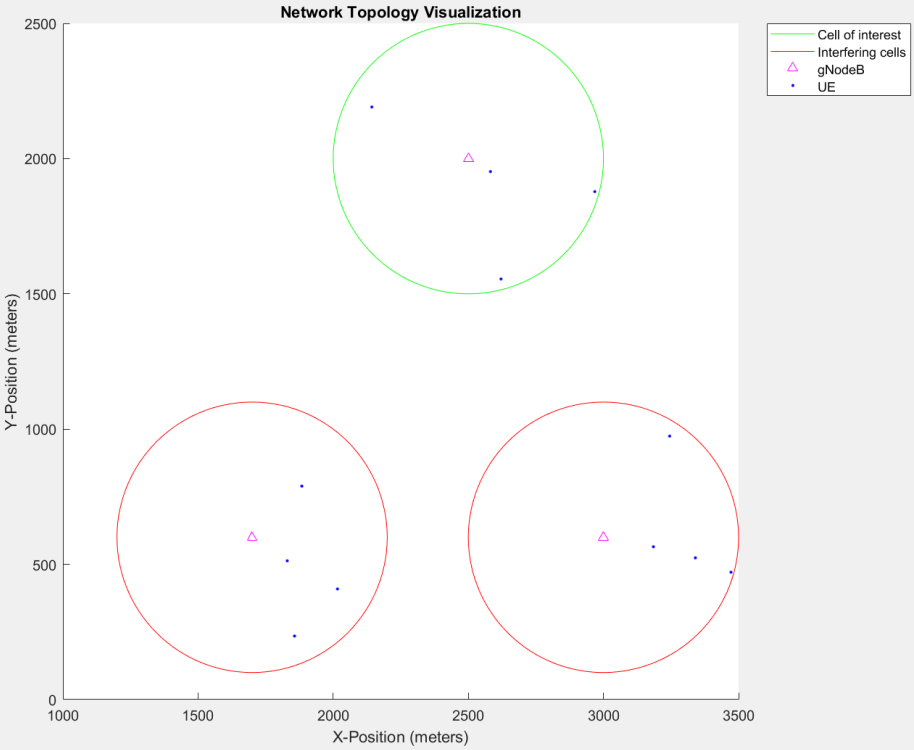
```
metricsVisualizer = helperNRMetricsVisualizer(gNBs(cellOfInterest),UEs{cellOfInterest}, ...
    CellOfInterest=cellOfInterest,NumMetricsSteps=numMetricsSteps, ...
    PlotSchedulerMetrics=true,PlotPhyMetrics=true,LinkDirection=linkDir);
```

Write the logs to MAT-files. You can use these logs for post-simulation analysis.

```
simulationLogFile = "simulationLogs"; % For logging the simulation traces
```

Display the network topology.

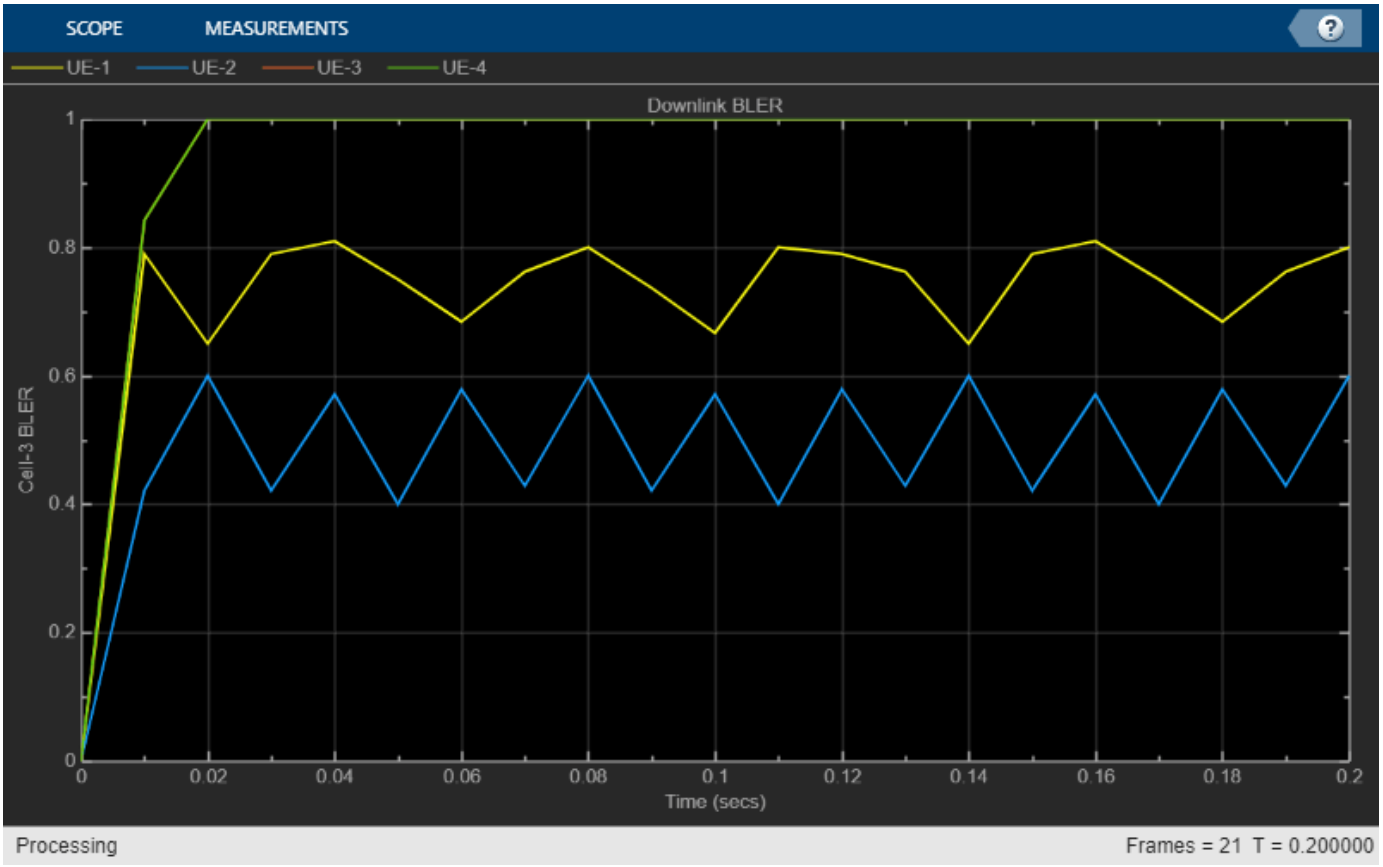
```
plotNetwork(cellOfInterest,cellRadius,gNBs,UEs);
```

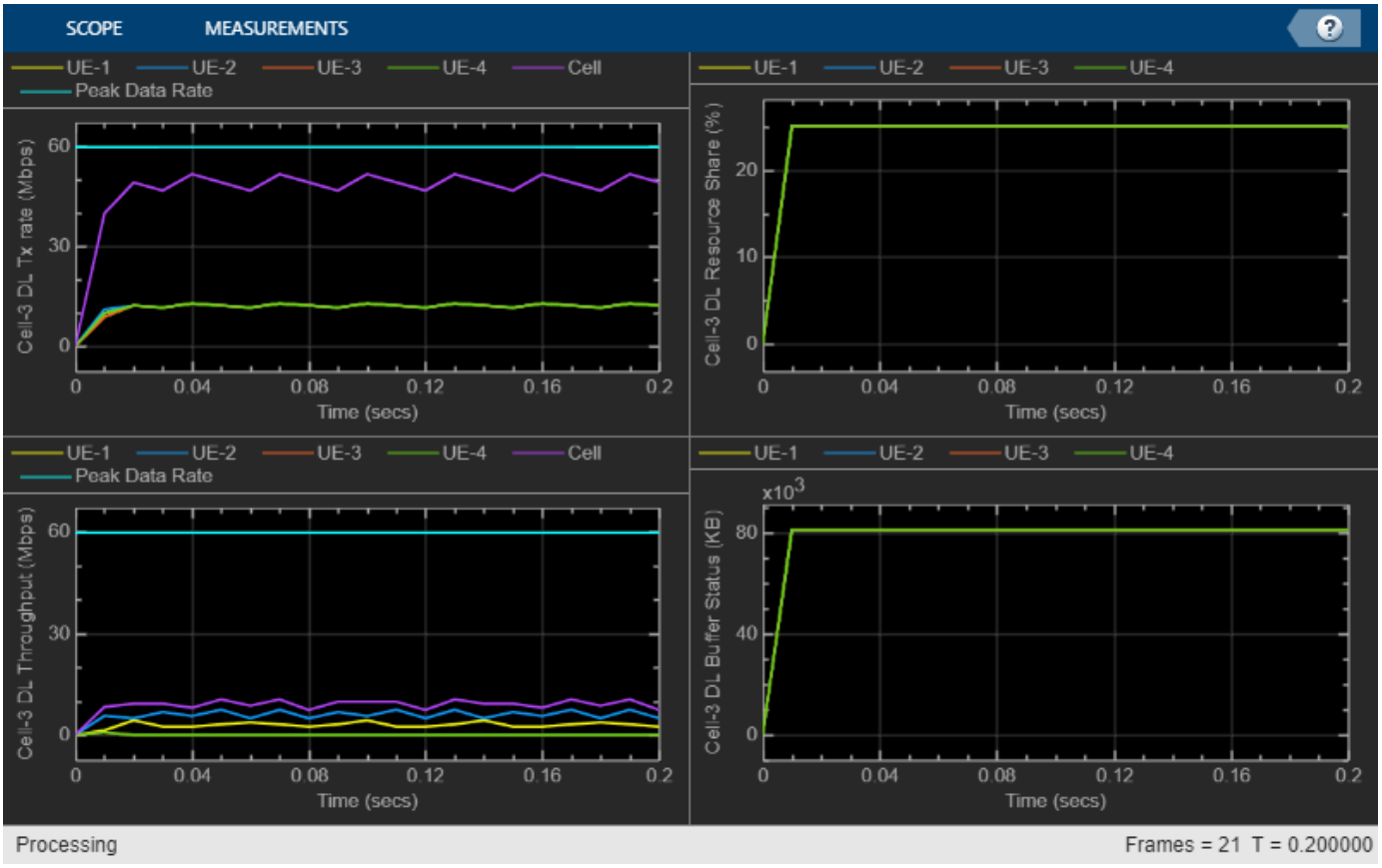


Run the simulation for the specified `numFrameSimulation` frames.

```
% Calculate the simulation duration (in seconds)
simulationTime = numFrameSimulation*1e-2;
% Run the simulation
run(networkSimulator,simulationTime);
```

```
Custom channel model is not added. Using free space path loss (fspl) model as the default channe
```


SCOPE
MEASUREMENTS
UE-1
UE-2
UE-3
UE-4
Downlink BLER
Cell-3 BLER
Time (secs)
0
0.2
0.4
0.6
0.8
1
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
Processing
Frames = 21 T = 0.200000



Read per-node stats.

```
gNBStats = statistics(gNBs);
ueStats = cell(numCells, 1);
for cellIdx = 1:numCells
    ueStats{cellIdx} = statistics(UEs{cellIdx});
end
```

At the end of the simulation, the achieved value for system performance indicators is compared to their theoretical peak values (considering zero overheads). Performance indicators displayed are achieved data rate (DL), achieved spectral efficiency (DL), and BLER observed for UEs (DL). The peak values are calculated as per 3GPP TR 37.910. In the specified cell of interest, UE-1, UE-3, and UE-4 are almost equidistant from their gNB. However, you can observe lower throughput for UE-3 and UE-4 as compared to UE-1. This is because UE-3 and UE-4 experience higher intercell interference.

```
displayPerformanceIndicators(metricsVisualizer)
```

```
Peak DL Throughput: 59.72 Mbps. Achieved Cell DL Throughput: 9.11 Mbps
Achieved DL Throughput for each UE: [2.98        6.06        0.03        0.03]
Peak DL spectral efficiency: 5.97 bits/s/Hz. Achieved DL spectral efficiency for cell: 0.91 bits/
Block error rate for each UE in the downlink direction: [0.752       0.501       0.992       0.99
```

Simulation Visualization

The run-time visualization shown are:

- *Display of Network Topology*: The figure shows the configured cell topology. For each cell, it shows the position of the gNB and the connected UEs.
- *Display of DL scheduling metrics plots*: For details, see 'Downlink Scheduler Performance Metrics ' figure description in “NR Cell Performance Evaluation with MIMO” on page 6-41 example.
- *Display of DL Block Error Rates*: The plot displayed in 'Block Error Rate (BLER) Visualization' shows the BLER (for each UE) observed in the downlink direction, as the simulation progresses.

Simulation Logs

The simulation logs are saved in MAT-files for post-simulation analysis and visualization. The per time step logs, scheduling assignment logs, and Phy reception logs are captured for each cell in the simulation and saved in the MAT-file `simulationLogFile`. After the simulation, open the file to load `NCellID`, `DLTimeStepLogs`, `SchedulingAssignmentLogs`, and PhyReception`Logs` in the workspace.

NCellID: This stores the cell ID and represents cell to which the simulation logs belong.

DL time step logs: Stores the per slot logs of the simulation with each slot as one row in the simulation. For details of log format, see the 'Time step logs' section of the “NR Cell Performance Evaluation with MIMO” on page 6-41 example.

Scheduling assignment logs: Information of all the scheduling assignments and related information is logged in this file. For details of log format, see the 'Scheduling assignment logs' section in the “NR Cell Performance Evaluation with MIMO” on page 6-41 example.

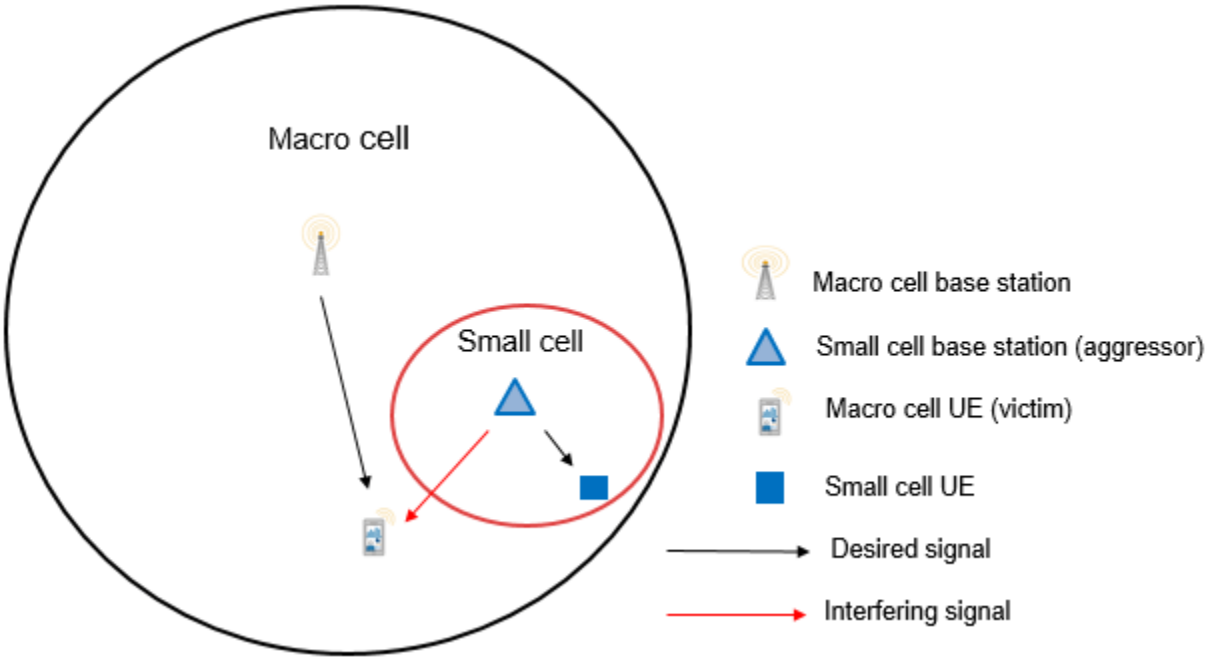
Phy reception logs: This file logs the packet reception information observed in the simulation. For details of log format, see the 'Phy reception logs' section in the “NR Cell Performance Evaluation with MIMO” on page 6-41 example.

```
if enableTraces
    % Get the logs
    simulationLogs = cell(numCells, 1);
    for cellIdx = 1:numCells
        if gNBs(cellIdx).DuplexMode == "FDD"
            logInfo = struct("NCellID",[],"DLTimeStepLogs",[], ...
                "SchedulingAssignmentLogs",[],"PhyReceptionLogs",[]);
            logInfo.DLTimeStepLogs = getSchedulingLogs(simSchedulingLogger{cellIdx});
        else % TDD
            logInfo = struct("NCellID",[],"TimeStepLogs",[], ...
                "SchedulingAssignmentLogs",[],"PhyReceptionLogs",[]);
            logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger{cellIdx});
        end
        logInfo.NCellID = gNBs(cellIdx).ID;
        % Get the scheduling assignments log
        logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger{cellIdx});
        % Get the Phy reception logs
        logInfo.PhyReceptionLogs = getReceptionLogs(simPhyLogger{cellIdx});
        simulationLogs{cellIdx} = logInfo;
    end
    % Save simulation logs in a MAT-file
    save(simulationLogFile,"simulationLogs")
end
```

Further Exploration

You can use this example to further explore these options:

- Model the uplink interference between the nodes by specifying the uplink-related configuration parameters. For details, see the “NR Cell Performance Evaluation with MIMO” on page 6-41 example.
- Model the Aggressor-Victim scenarios: The aggressor is the source of interference and the victim suffers due to the interference. Consider the DL scenario in the following figure. The macro cell UE is far away from the macro base station (BS) and near to the small cell. In DL direction, macro cell UE suffers from interference by the small cell BS. The small cell BS is called the aggressor and macro UE is called the victim.



- Model multiple clusters, where each cluster consists of cells operating on different frequencies, and analyze the impact of interference on the cell edge users.

Local functions

```
function plotNetwork(cellOfInterest,cellRadius,gNBs,UEs)
%plotNetwork Create the network figure

figure(Name="Network Topology Visualization",units="normalized", ...
    outerposition=[0 0 1 1],Visible="on");
title("Network Topology Visualization");
hold on

numCells = numel(gNBs);
for cellIdx = 1:numCells

    gNBPosition = gNBs(cellIdx).Position;
    % Plot the circle
    th = 0:pi/60:2*pi;
    xunit = cellRadius * cos(th) + gNBPosition(1);
    yunit = cellRadius * sin(th) + gNBPosition(2);
    if cellOfInterest == cellIdx
        h1 =  plot(xunit,yunit,Color="green"); % Cell of interest
```

```
    else
        h2 =  plot(xunit,yunit,Color="red");
    end
    xlabel("X-Position (meters)")
    ylabel("Y-Position (meters)")
    % Add tool tip data for gNBs
    s1 = scatter(gNBPosition(1),gNBPosition(2),"^", ...
        MarkerEdgeColor="magenta");
    cellIdRow = dataTipTextRow("Cell - ",{num2str(cellIdx)});
    s1.DataTipTemplate.DataTipRows(1) = cellIdRow;
    posRow = dataTipTextRow('Position[X, Y]: ',{['[' num2str(gNBPosition) ']']});
    s1.DataTipTemplate.DataTipRows(2) = posRow;

    % Add tool tip data for UEs
    uesPerCell = UEs{cellIdx};
    for ueIdx = 1:numel(uesPerCell)
        uePosition = uesPerCell(ueIdx).Position;
        s2 = scatter(uePosition(1),uePosition(2),".",MarkerEdgeColor="blue");
        s2.DataTipTemplate.DataTipRows(1) = uesPerCell(ueIdx).Name;
        posRow = dataTipTextRow('Position[X, Y]: ',{['[' num2str(uePosition) ']']});
        s2.DataTipTemplate.DataTipRows(2) = posRow;
    end
end
% Create the legend
if numCells > 1
    legend([h1 h2 s1 s2],"Cell of interest","Interfering cells","gNodeB", ...
        "UE","Location", "northeastoutside")
else
    legend([h1 s1 s2],"Cell of interest","gNodeB","UE","Location","northeastoutside")
end
axis auto
hold off
daspect([1000,1000,1]); % Set data aspect ratio
end

function uePositions = generateUEPositions(cellRadius,gNBPositions,numUEsPerCell)
%generateUEPositions Return the position of UEs in each cell

numCells = size(gNBPositions,1);
uePositions = cell(numCells,1);
for cellIdx=1:numCells
    gnbXCo = gNBPositions(cellIdx,1); % gNB X-coordinate
    gnbYCo = gNBPositions(cellIdx,2); % gNB Y-coordinate
    gnbZCo = gNBPositions(cellIdx,3); % gNB Z-coordinate
    theta = rand(numUEsPerCell,1)*(2*pi);
    % Expression to calculate position of UEs with in the cell. By default,
    % it will place the UEs randomly with in the cell
    r = sqrt(rand(numUEsPerCell,1))*cellRadius;
    x = round(gnbXCo + r.*cos(theta));
    y = round(gnbYCo + r.*sin(theta));
    z = ones(numUEsPerCell,1) * gnbZCo;
    uePositions{cellIdx} = [x y z];
end
end
```

Appendix

The example uses these helper classes:

- helperNRMetricsVisualizer.m: Implements metrics visualization functionality
- helperNRSchedulingLogger.m: Implements scheduling information logging functionality
- helperNRPhyLogger.m: Implements PHY packet reception information logging functionality

References

[1] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.322. “NR; Radio Link Control (RLC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 3GPP TS 38.323. “NR; Packet Data Convergence Protocol (PDCP) specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[6] 3GPP TS 38.331. “NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[7] 3GPP TR 37.910. “Study on self evaluation towards IMT-2020 submission.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Objects
`wirelessNetworkSimulator` | `nrGNB` | `nrUE`

Related Examples

- “NR Interference Modeling with Toroidal Wrap-Around” on page 6-29
- “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87
- “NR FDD Scheduling Performance Evaluation” on page 6-56

Generate and Visualize FTP Application Traffic Pattern

This example shows how to generate a file transfer protocol (FTP) application traffic pattern based on the IEEE® 802.11ax™ Evaluation Methodology [1 on page 6-117] and the 3GPP TR 36.814 specification [2 on page 6-117].

FTP Application Traffic Model

Multinode communication systems involve modeling of different application traffic models. Each application is characterized by parameters such as the data rate, packet inter arrival time, and packet size. To evaluate various algorithms and protocols, standardization bodies such as IEEE and 3GPP define certain application traffic patterns such as Voice over Internet Protocol (VoIP), video conferencing, and FTP. This example generates and visualizes an FTP application traffic pattern.

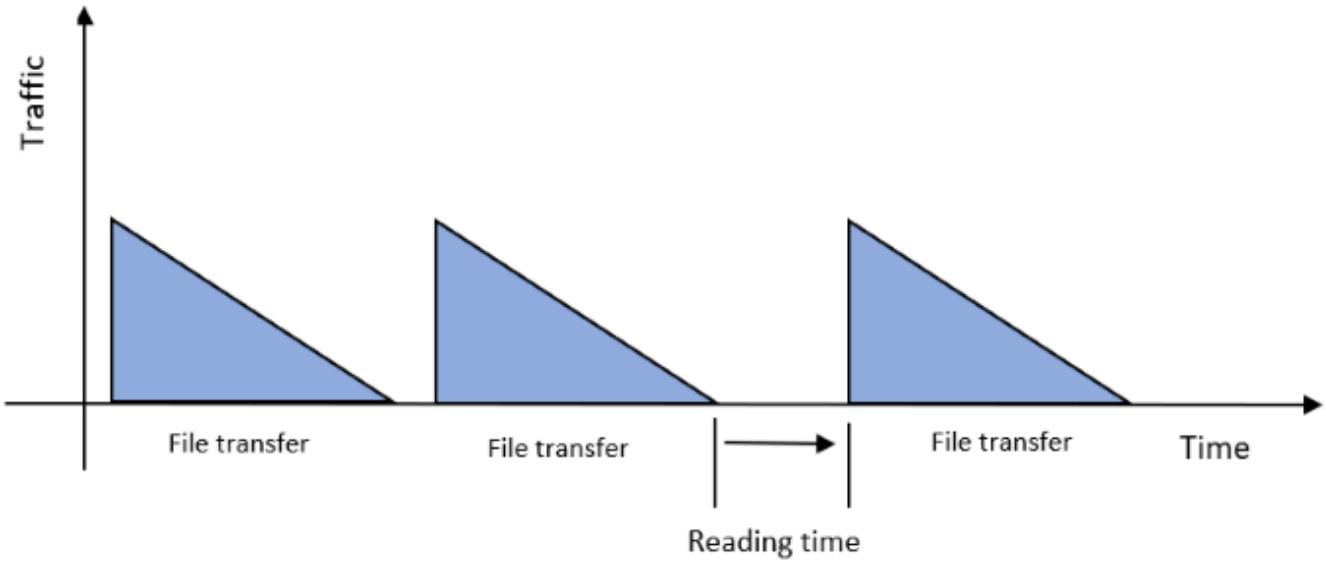
The FTP application traffic pattern is modeled as a sequence of file transfers separated by reading time. The reading time specifies the time interval between two successive file transfers. The file is generated as multiple packets separated by packet inter arrival time. The packet inter arrival time specifies the time interval between two successive packet transfers.

The 11ax Evaluation Methodology [1 on page 6-117] specifies this FTP application traffic model:

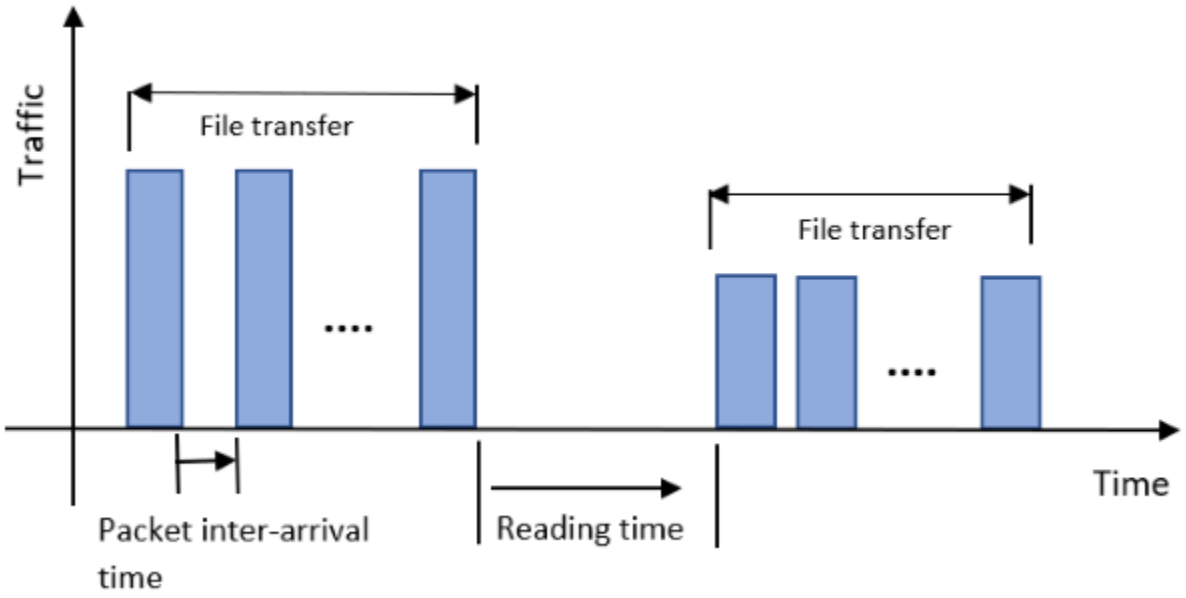
- **Local FTP traffic model -** This model is characterized by truncated Lognormal file size and exponential reading time.

The 3GPP TR 36.814 specification [2 on page 6-117] specifies these FTP application traffic models:

- **FTP traffic model 2 -** This model is characterized by 2/0.5 megabytes file size and exponential reading time. This figure shows the traffic pattern of this FTP model.



- **FTP traffic model 3 -** This model is characterized by a 0.5 megabytes file, exponential reading time, and Poisson packet arrival rate. This figure shows the traffic pattern of this FTP model.



This example demonstrates the local FTP traffic model specified in 11-ax Evaluation Methodology [1 on page 6-117]. Similarly, you can use the FTP traffic models 2 and 3 specified in 3GPP TR 36.814 specification [2 on page 6-117] using the file size and packet arrival rate properties.

Configure FTP Application Traffic Pattern Object

Check if the 'Communications Toolbox Wireless Network Simulation Library' support package is installed.

```
wirelessnetworkSupportPackageCheck
```

Create a configuration object to generate the FTP application traffic pattern.

```
% Reset the random number generator
rng('default');

% Create FTP application traffic pattern object with default properties
ftpObj = networkTrafficFTP;

% Set exponential distribution mean value for reading time in milliseconds
ftpObj.ExponentialMean = 50;

% Set truncated Lognormal distribution mu value for file size calculation
ftpObj.LogNormalMu = 10;

% Set truncated Lognormal distribution sigma value for file size calculation
ftpObj.LogNormalSigma = 1;

% Set truncated Lognormal distribution upper limit in Megabytes
ftpObj.UpperLimit = 5;

% Display object
disp(ftpObj);
```

```
  networkTrafficFTP with properties:

              LogNormalMu: 10
           LogNormalSigma: 1
               UpperLimit: 5
          ExponentialMean: 50
    PacketInterArrivalTime: 0
            GeneratePacket: 0
```

Generate and Visualize FTP Application Traffic Pattern

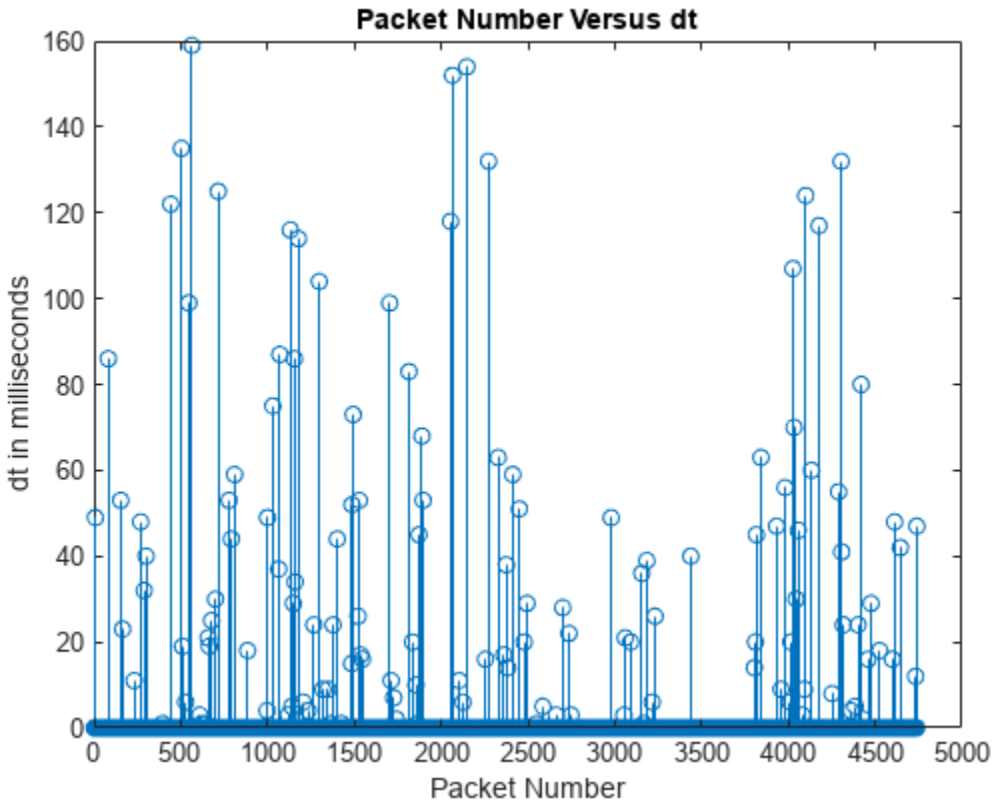
Generate FTP application traffic pattern using the `generate` object function of the `networkTrafficFTP` object.

```
% Set simulation time in milliseconds
simTime = 10000;

% Set step time in milliseconds
stepTime = 1;

% Validate simTime, simTime must be greater than or equal to stepTime
validateattributes(simTime,{'numeric'}, ...
    {'real','scalar','finite','>=',stepTime});

% Time after which the generate method must be invoked again
nextInvokeTime = 0;

% Generated packet count
packetCount = 0;

% Initialize arrays to store outputs for visualization
% Packet generation times in milliseconds
generationTime = zeros(5000,1);

% Time interval between two consecutive packet transfers in milliseconds
packetIntervals = zeros(5000,1);

% Packet sizes in bytes
packetSizes = zeros(5000,1);

% Loop over the simulation time, generating FTP application traffic
% pattern and saving the dt and packet size values for visualization.
while simTime
    if nextInvokeTime <= 0           % Time to generate the packet
        packetCount = packetCount+1; % Increment packet count
        % Call generate method and store outputs for visualization
        [packetIntervals(packetCount), packetSizes(packetCount)] = ...
            generate(ftpObj);
        % Set next invoke time
        nextInvokeTime = packetIntervals(packetCount);
        % Store packet generation time for visualization
        generationTime(packetCount+1) = ...
            generationTime(packetCount) + packetIntervals(packetCount);
    end

    % Update next invoke time
    nextInvokeTime = nextInvokeTime - stepTime;
```

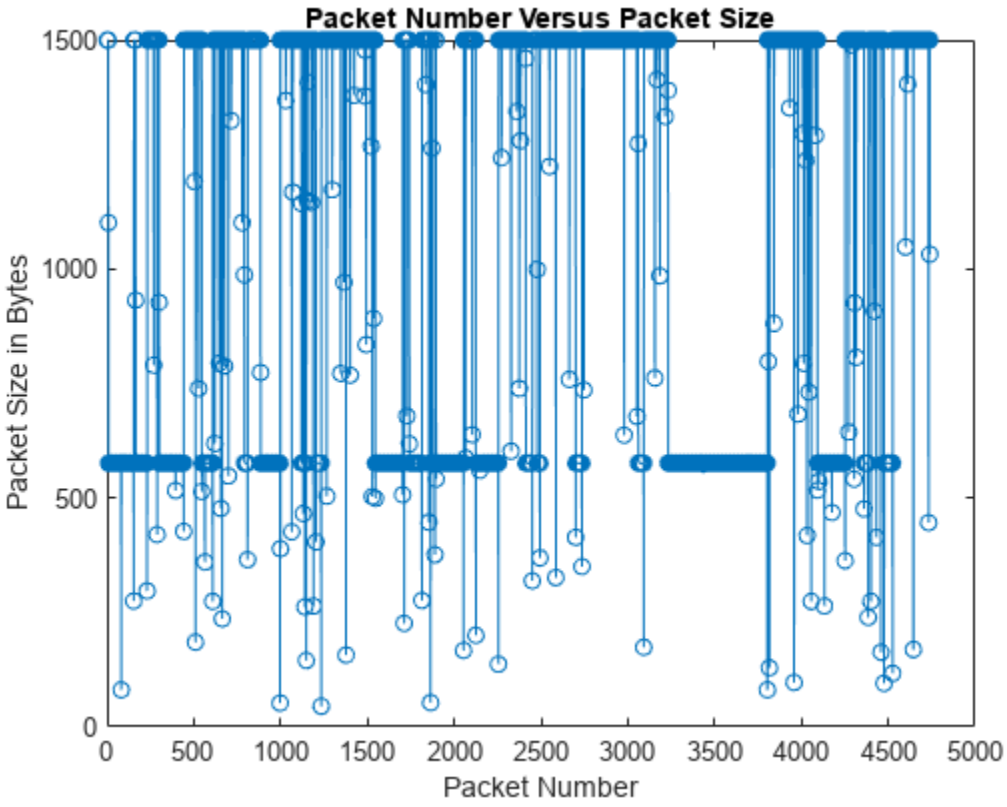
```
    % Update simulation time
    simTime = simTime - stepTime;
end
```

Visualize the generated FTP application traffic pattern. In this plot, `dt` is the time interval between two successive FTP application packets.

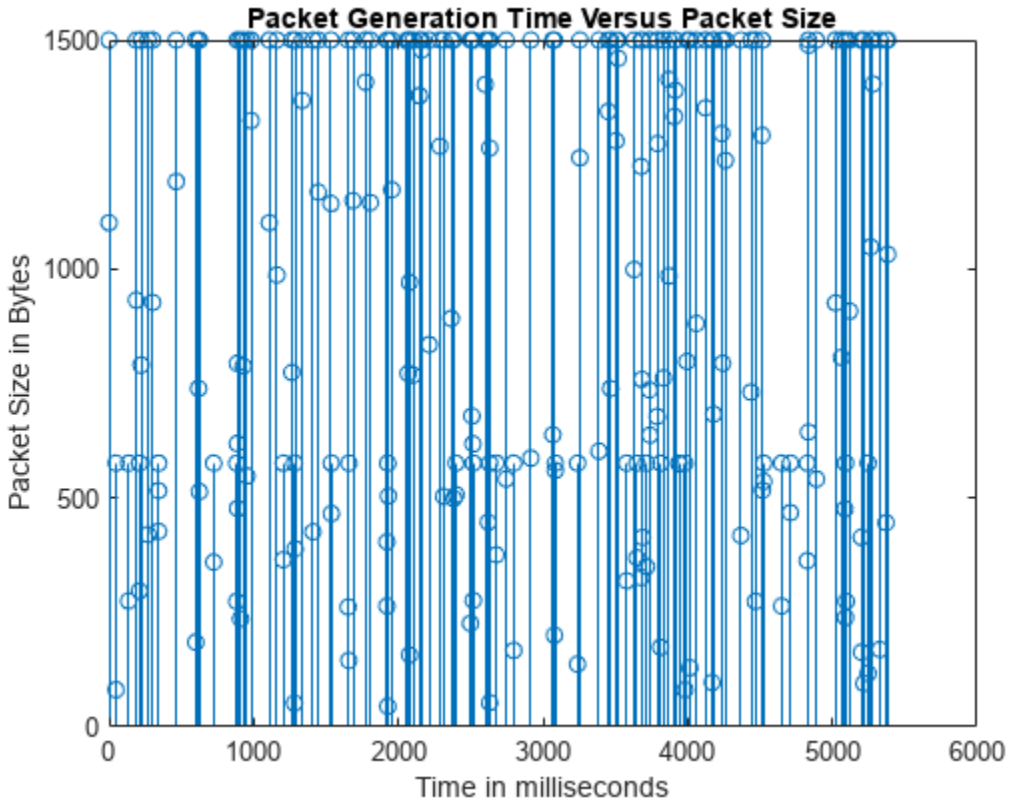
```
% Packet Number Versus Packet Intervals (dt)
% Stem graph to see packet intervals
pktIntervalsFig = figure(Name='Packet intervals',NumberTitle='off');
pktIntervalsAxes = axes(pktIntervalsFig);
stem(pktIntervalsAxes,packetIntervals(1:packetCount));
title(pktIntervalsAxes,'Packet Number Versus dt');
xlabel(pktIntervalsAxes,'Packet Number');
ylabel(pktIntervalsAxes,'dt in milliseconds');
```




```
% Plot to see different packet sizes
pktSizesFig = figure(Name='Packet sizes',NumberTitle='off');
pktSizesAxes = axes(pktSizesFig);
plot(pktSizesAxes,packetSizes(1:packetCount),Marker='o');
title(pktSizesAxes,'Packet Number Versus Packet Size');
xlabel(pktSizesAxes,'Packet Number');
ylabel(pktSizesAxes,'Packet Size in Bytes');
```




```
% Stem graph of FTP application traffic pattern (Packet sizes of
% different files at different packet generation times)
ftpPatternFig = figure(Name='FTP application traffic pattern', ...
    NumberTitle='off');
ftpPatternAxes = axes(ftpPatternFig);
stem(ftpPatternAxes,generationTime(1:packetCount), ...
    packetSizes(1:packetCount),Marker='o');
title(ftpPatternAxes,'Packet Generation Time Versus Packet Size');
ylabel(ftpPatternAxes,'Packet Size in Bytes');
xlabel(ftpPatternAxes,'Time in milliseconds');
```



Further Exploration

This example generates an FTP traffic pattern as per the 11ax Evaluation Methodology [1 on page 6-117] and 3GPP specification [2 on page 6-117]. Similarly, you can use `networkTrafficVoIP`, `networkTrafficOnOff`, and `networkTrafficVideoConference` objects to generate VoIP, On-Off, video conferencing application traffic patterns, respectively. You can use these different application traffic patterns in system-level simulations to model the real-world data traffic.

References

[1] IEEE 802.11-14/0571r12 . "11ax Evaluation Methodology". IEEE P802.11. Wireless LANs.

[2] 3GPP TR 36.814. "Evolved Universal Terrestrial Radio Access (E-UTRA). Further advancements for E-UTRA physical layer aspects". *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Objects

`networkTrafficFTP` | `networkTrafficOnOff` | `networkTrafficVideoConference` | `networkTrafficVoIP`

Plug In Custom Scheduler in System-Level Simulation

This example shows how to create a custom scheduling strategy, and drive it using the interfaces offered by the medium access control (MAC) scheduler. The example also shows how to integrate the custom scheduling strategy into the 5G system-level simulation framework and evaluate the network performance.

Scheduler Interface

The scheduler offers a function-based interface that enables the MAC layer to drive the scheduler. The function calls fall in any of the two categories as described below. Each call has a unique role.

Interface to Update Scheduler Context

A set of function calls update the scheduler context. This context contains the information used by the scheduler to make scheduling decisions.

Interface	Purpose
Update LC Buffer Status DL	Update downlink (DL) buffer status for a logical channel (LC) of a user equipment (UE)
Process MAC Control Element	Process a MAC control element (CE)
Update Channel Quality UL	Update uplink (UL) channel quality information for a UE
Update Channel Quality DL	Update DL channel quality information for a UE
Handle DL Rx Result	Update the hybrid automatic repeat request (HARQ) process context based on the Rx success/failure for DL packets
Handle UL Rx Result	Update the HARQ process context based on the Rx success/failure for UL packets

Interface to Run DL and UL Scheduler to Obtain Resource Assignments

These interface calls (to run the DL and UL scheduler) provide the scheduling decisions, which consists of the DL and UL resource assignments.

Interface	Purpose
Run DL Scheduler	Run the DL scheduler to assign the resources to the UEs and return the scheduling decisions as output
Run UL Scheduler	Run the UL scheduler to assign the resources to the UEs and return the scheduling decisions as output

For a detailed description of the input and output format of these functions, see hNRScheduler.m.

Custom Scheduling Strategy

This section describes the sample scheduling strategy that this example uses.

The scheduling strategy considered in this example is a round-robin strategy, without support for retransmissions. Each UE gets every *N*th resource block group (RBG) in the bandwidth, where *N* is the number of UEs in the cell. UL and DL schedulers assign RBGs to a UE if there is a nonzero buffer amount in the corresponding direction. The hCustomScheduler.m helper class is used to implement this scheduling strategy.

Follow these steps to create and implement this scheduling strategy.

Create Custom Scheduler Class

Create a new class `hCustomScheduler`, and inherit it from hNRScheduler.m super class. Implement the constructor of the class to call the super class constructor.

```
function obj = hCustomScheduler(simParameters)

% Invoke the super class constructor to initialize the properties

obj = obj@hNRScheduler(simParameters);

% Initialize the properties that are specific to this custom scheduling
strategy

end
```

Implement Custom UL Scheduling

Customizing UL scheduling involves overriding the super class functions in `hCustomScheduler` to select the UL slots to be scheduled, and to schedule the selected slots.

Override the function `selectULSlotsToBeScheduled` with your custom logic to select the slots to be scheduled in the current run.

```
function slotNum = selectULSlotsToBeScheduled(obj)

% Select the set of slots to be scheduled in this UL scheduler run

end
```

If you do not override this function, the default slot selection strategy is implemented in the super class (hNRScheduler.m).

Override the `scheduleULResourcesSlot` function with your custom logic to schedule the selected slots. If `selectULSlotsToBeScheduled` selects multiple slots, then the scheduler engine calls this function for each of the selected slots. One slot is scheduled per call with input parameter `slotNum` as the scheduled slot. Your custom logic must output an array of valid UL assignments for the scheduled slot. Each UL assignment is a structure that contains fields to define a physical uplink shared channel (PUSCH) transmission. For a detailed description of the structure fields, see the `scheduleULResourcesSlot` function in hNRScheduler.m class.

```
function uplinkGrants = scheduleULResourcesSlot(obj,slotNum)

% Implement custom UL scheduling to populate the output uplink grants

end
```

To learn how the UL scheduling is customized by overriding the `scheduleULResourcesSlot` as per the described strategy, see hCustomScheduler.m. The custom strategy does not override the

`selectULSlotsToBeScheduled` function and uses default slot selection strategy in the hNRScheduler.m super class.

Implement Custom DL Scheduling

Follow the steps (similar to UL scheduling) to customize DL scheduling. The process involves overriding the `selectDLSlotsToBeScheduled` and `scheduleDLResourcesSlot` functions in `hCustomScheduler` to select the DL slots to be scheduled, and to schedule the selected slots.

```
function slotNum = selectDLSlotsToBeScheduled(obj)

% Select the set of slots to be scheduled in this DL scheduler run

end

function downlinkGrants = scheduleDLResourcesSlot(obj,slotNum)

% Implement custom DL scheduling to populate the output downlink grants

end
```

To learn how the DL scheduling is customized by overriding the `scheduleDLResourcesSlot` as per the described strategy, see hCustomScheduler.m. The custom strategy does not override the `selectDLSlotsToBeScheduled` function and uses default slot selection strategy in the hNRScheduler.m super class.

Create Custom Scheduler Object

```
param.NumUEs = 4;                    % Number of UEs in the cell
param.NumRBs = 160;                  % Number of resource blocks in the bandwidth
scheduler = hCustomScheduler(param); % Scheduler object
```

Drive Custom Scheduler

This section updates the scheduler context and runs the scheduler using scheduler interface.

Run Scheduler (Without Updating Context)

Run the DL and UL schedulers. These calls take information about the current time as an input structure.

```
% Current system frame number
currentTimeInfo.SFN = 0;
% Current slot number
currentTimeInfo.CurrSlot = 0;
% Current symbol number
currentTimeInfo.CurrSymbol = 0;
% Assign UL resources to UEs for the slot to be scheduled
ulResourceAssignments = runULScheduler(scheduler,currentTimeInfo);
disp(length(ulResourceAssignments));
```

```
     0
```

```
% Assign DL resources to UEs for the slot to be scheduled
dlResourceAssignments = runDLScheduler(scheduler,currentTimeInfo);
disp(length(dlResourceAssignments));
```

```
     0
```

Because the scheduler does not have any information about the UL and DL buffer status, it assumes the buffer status as zero and does not schedule any resource.

Update DL Buffer Status

Run the DL scheduler after updating the DL buffer status of three UEs.

```
% Update DL buffer status for UE-1
lcBufferStatus.RNTI = 1;
lcBufferStatus.LogicalChannelID = 20;
lcBufferStatus.BufferStatus = 2000;                                    % In bytes
updateLCBufferStatusDL(scheduler,lcBufferStatus);

% Update DL buffer status for UE-2
lcBufferStatus.RNTI = 2;
lcBufferStatus.LogicalChannelID = 25;
lcBufferStatus.BufferStatus = 3000;                                    % In bytes
updateLCBufferStatusDL(scheduler,lcBufferStatus);

% Update DL buffer status for UE-3
lcBufferStatus.RNTI = 3;
lcBufferStatus.LogicalChannelID = 28;
lcBufferStatus.BufferStatus = 8000;                                    % In bytes
updateLCBufferStatusDL(scheduler,lcBufferStatus);

% Run DL scheduler to see the scheduled assignments
dlResourceAssignments = runDLScheduler(scheduler,currentTimeInfo);
for i = 1:length(dlResourceAssignments)
    disp(dlResourceAssignments{i});
end
```

```
                    RNTI: 1
                    Type: 'newTx'
        RBGAllocationBitmap: [1 0 0 0 1 0 0 0 1 0]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
            PrecodingMatrix: 1
                        MCS: 9
         FeedbackSlotOffset: 2
                         RV: 0
                    HARQID: 0
                        NDI: 1

                    RNTI: 2
                    Type: 'newTx'
        RBGAllocationBitmap: [0 1 0 0 0 1 0 0 0 1]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
```

```
           PrecodingMatrix: 1
                       MCS: 9
         FeedbackSlotOffset: 2
                        RV: 0
                    HARQID: 0
                       NDI: 1

                      RNTI: 3
                      Type: 'newTx'
        RBGAllocationBitmap: [0 0 1 0 0 0 1 0 0 0]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
           PrecodingMatrix: 1
                       MCS: 9
         FeedbackSlotOffset: 2
                        RV: 0
                    HARQID: 0
                       NDI: 1
```

UE-4 did not get any resources because the DL buffer status is updated only for UE-1, UE-2, and UE-3.

Update UL Buffer Status

Run the UL scheduler after updating the UL buffer status of all of the UEs. This information comes in the form of a buffer status report (BSR) CE from the UEs.

```
% Update UL buffer status for UE-1
macCEInfo.RNTI = 1;
macCEInfo.LCID = 60;                                                % Long Truncated BSR
macCEInfo.Packet = [15;21;88;108];                                  % BSR packet content
priority = [2, 3, 4, 1, 6, 7, 9, 5];                                % Priority of each logical ch
processMACControlElement(scheduler,macCEInfo,priority);

% Update UL buffer status for UE-2
macCEInfo.RNTI = 2;
macCEInfo.LCID = 62;                                                % Long BSR
macCEInfo.Packet = [53;34;78;101;103];                              % BSR packet content
processMACControlElement(scheduler,macCEInfo);

% Update UL buffer status for UE-3
macCEInfo.RNTI = 3;
macCEInfo.LCID = 60;                                                % Long Truncated BSR
macCEInfo.Packet = [38;48;55];                                      % BSR packet content
priority = [6, 7, 9, 5, 2, 3, 4, 1];                                % Priority of each LCG
processMACControlElement(scheduler,macCEInfo,priority);

% Update UL buffer status for UE-4
macCEInfo.RNTI = 4;
macCEInfo.LCID = 62;                                                % Long BSR
macCEInfo.Packet = [65;21;55];                                      % BSR packet content
processMACControlElement(scheduler,macCEInfo);
```

```
% Run UL scheduler to see the scheduled assignments
ulResourceAssignments = runULScheduler(scheduler,currentTimeInfo);
for i = 1:length(ulResourceAssignments)
    disp(ulResourceAssignments{i});
end
```

```
                    RNTI: 1
                    Type: 'newTx'
        RBGAllocationBitmap: [1 0 0 0 1 0 0 0 1 0]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 9
                       TPMI: 0
                         RV: 0
                    HARQID: 0
                        NDI: 1

                    RNTI: 2
                    Type: 'newTx'
        RBGAllocationBitmap: [0 1 0 0 0 1 0 0 0 1]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 9
                       TPMI: 0
                         RV: 0
                    HARQID: 0
                        NDI: 1

                    RNTI: 3
                    Type: 'newTx'
        RBGAllocationBitmap: [0 0 1 0 0 0 1 0 0 0]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 9
                       TPMI: 0
                         RV: 0
                    HARQID: 0
                        NDI: 1

                    RNTI: 4
```

```
                    Type: 'newTx'
        RBGAllocationBitmap: [0 0 0 1 0 0 0 1 0 0]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 9
                       TPMI: 0
                         RV: 0
                     HARQID: 0
                        NDI: 1
```

Update UL Channel Quality

Run the UL scheduler after updating the UL channel quality, and then observe the modulation and coding scheme (MCS) index in the resource assignments.

```
% Update the UL channel quality for UE-2
channelQualityInfo.RNTI = 2;
channelQualityInfo.CQI = 8*ones(1,param.NumRBs);
updateChannelQualityUL(scheduler,channelQualityInfo);

% Update the UL channel quality for UE-3
channelQualityInfo.RNTI = 3;
channelQualityInfo.CQI = 15*ones(1,param.NumRBs);
updateChannelQualityUL(scheduler,channelQualityInfo);

% Run UL scheduler to see the scheduled assignments
ulResourceAssignments = runULScheduler(scheduler,currentTimeInfo);
for i = 1:length(ulResourceAssignments)
    disp(ulResourceAssignments{i});
end
```

```
                    RNTI: 1
                    Type: 'newTx'
        RBGAllocationBitmap: [1 0 0 0 1 0 0 0 1 0]
                StartSymbol: 0
                 NumSymbols: 14
                 SlotOffset: 1
                MappingType: 'A'
                 DMRSLength: 1
                  NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 9
                       TPMI: 0
                         RV: 0
                     HARQID: 1
                        NDI: 1

                    RNTI: 2
                    Type: 'newTx'
        RBGAllocationBitmap: [0 1 0 0 0 1 0 0 0 1]
                StartSymbol: 0
```

```
              NumSymbols: 14
              SlotOffset: 1
             MappingType: 'A'
              DMRSLength: 1
               NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 11
                       TPMI: 0
                         RV: 0
                    HARQID: 1
                        NDI: 1

                       RNTI: 3
                       Type: 'newTx'
        RBGAllocationBitmap: [0 0 1 0 0 0 1 0 0 0]
                StartSymbol: 0
                NumSymbols: 14
                SlotOffset: 1
                MappingType: 'A'
                DMRSLength: 1
                 NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 25
                       TPMI: 0
                         RV: 0
                    HARQID: 1
                        NDI: 1

                       RNTI: 4
                       Type: 'newTx'
        RBGAllocationBitmap: [0 0 0 1 0 0 0 1 0 0]
                StartSymbol: 0
                NumSymbols: 14
                SlotOffset: 1
                MappingType: 'A'
                DMRSLength: 1
                 NumLayers: 1
    NumCDMGroupsWithoutData: 2
            NumAntennaPorts: 1
                        MCS: 9
                       TPMI: 0
                         RV: 0
                    HARQID: 1
                        NDI: 1
```

The assignments show that the modulation coding scheme (MCS) of the UL resource assignment to UE-3 is greater than that of UE-2. Similarly, you can observe the effect of channel quality on the MCS for DL resource assignments by using the `updateChannelQualityDL` function to update the DL channel quality.

Plug In Scheduler

Plug in the scheduler to 5G Toolbox™ system-level simulation example “NR FDD Scheduling Performance Evaluation” on page 6-56.

Copy the hCustomScheduler.m file to the example folder.

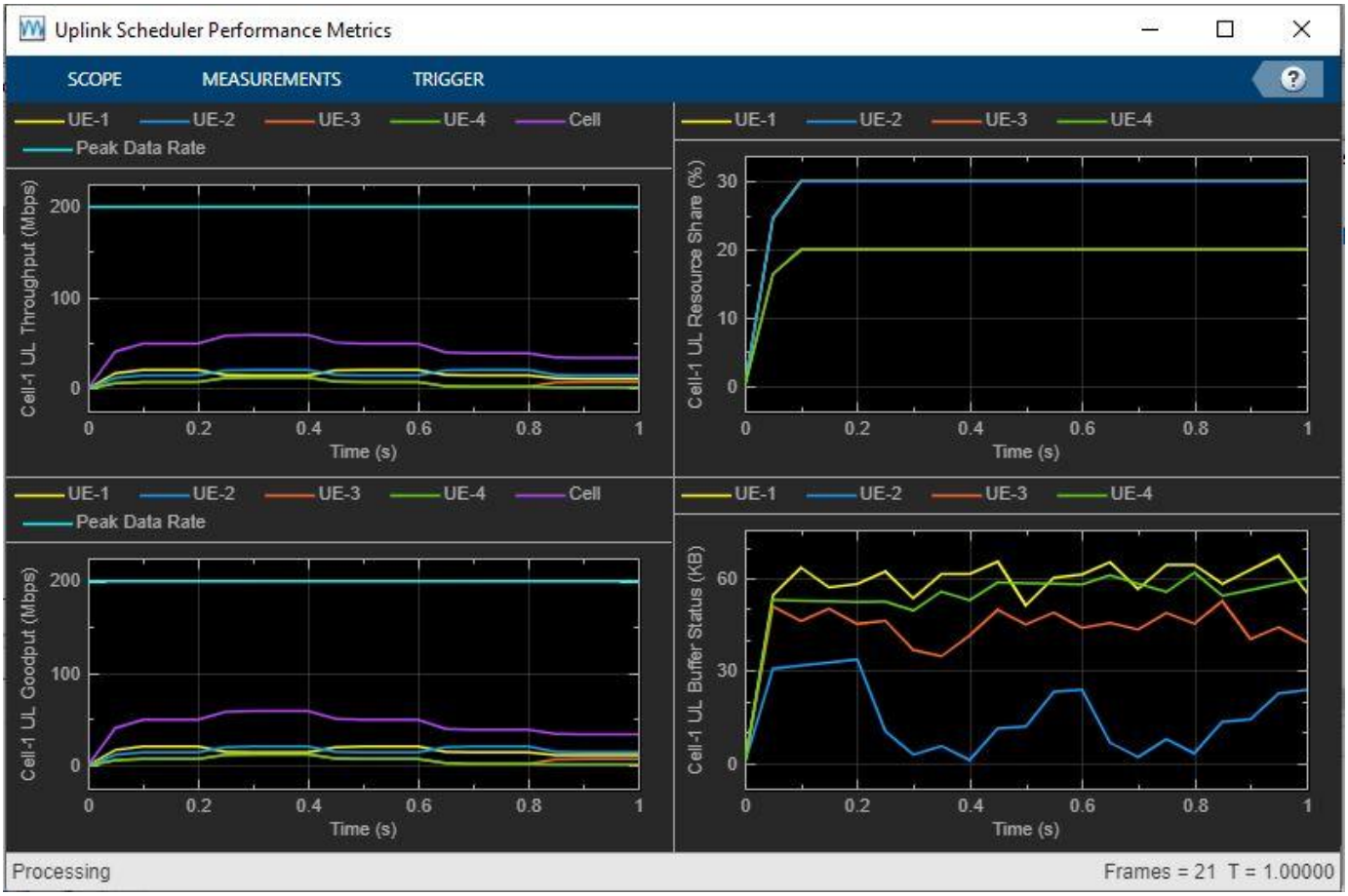
Create and Install Custom Scheduler

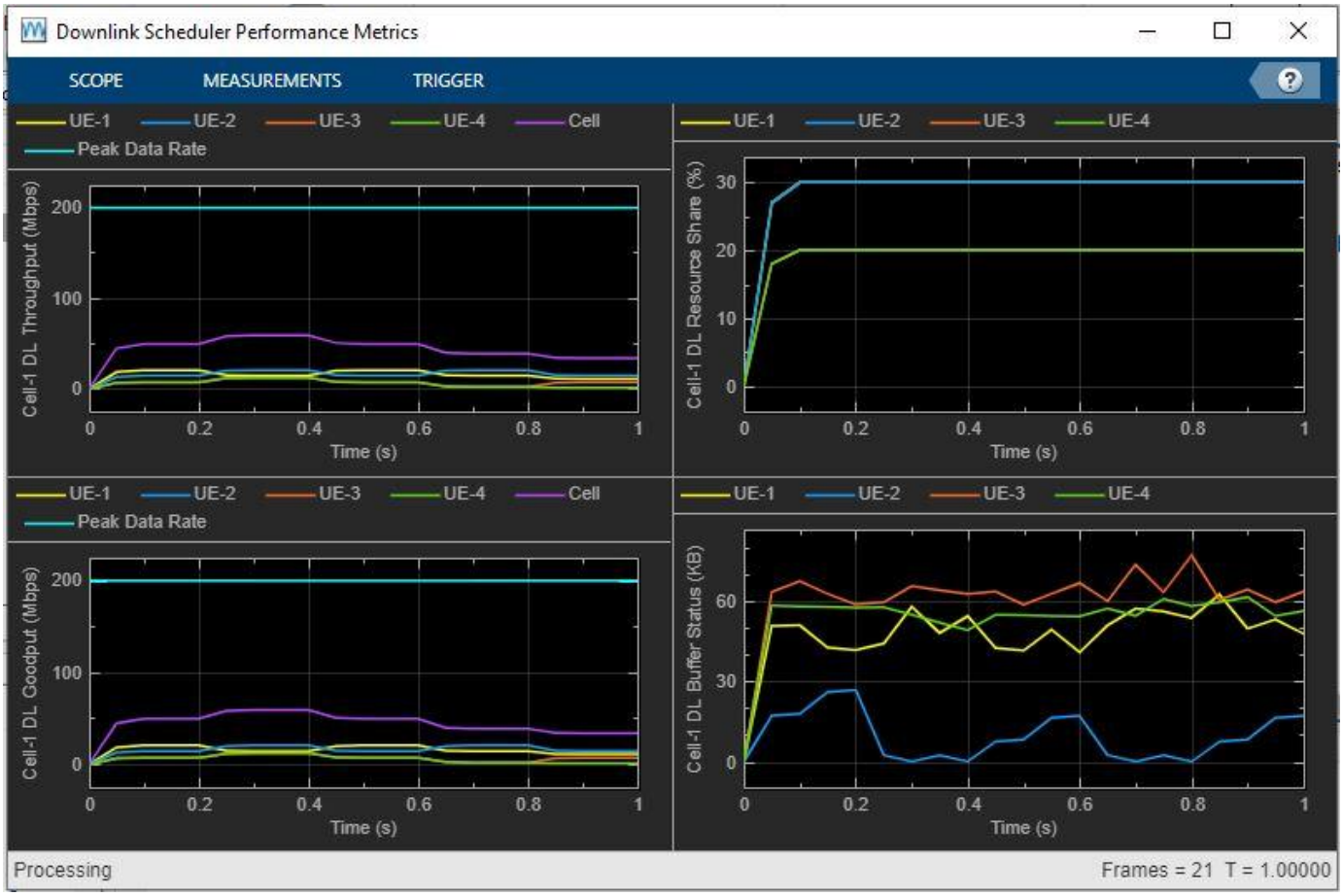
Create a custom scheduler object and install it on the gNB (as described in the "gNB and UEs Setup" section of the “NR FDD Scheduling Performance Evaluation” on page 6-56 example).

```
scheduler = hCustomScheduler(simParameters);

addScheduler(gNB,scheduler); % Add scheduler to gNB
```

Run Example with Custom Scheduler

Run the main script of the simulation example “NR FDD Scheduling Performance Evaluation” on page 6-56, and then observe the scheduler performance.





The resulting plots show that the throughput and goodput (which accounts for only new transmissions) is the same for all of the UEs because the custom scheduling strategy does not support retransmissions.

References

[1] 3GPP TS 38.321. “NR; Medium Access Control (MAC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “NR FDD Scheduling Performance Evaluation” on page 6-56
- “NR Cell Performance Evaluation with Physical Layer Integration” on page 6-87

NR Cell Performance with Downlink MU-MIMO

This example shows how to evaluate the system performance of a codebook-based downlink (DL) multi-user (MU) multiple-input multiple-output (MIMO) by using 5G Toolbox™ and the Communications Toolbox™ Wireless Network Simulation Library. The example uses link-to-system mapping-based abstract physical layer (PHY).

Using this example, you can

- Model downlink MU-MIMO.
- Modify the MU-MIMO related scheduler configuration, and observe the impact on cell performance.
- Analyze cell performance for various channel delay profiles.

The simulation results show these key performance indicators (KPIs).

- Cell throughput
- Spectral efficiency
- Block error rate (BLER)

Introduction

MU-MIMO is an important capability for increasing the capacity of a cellular network. Digital precoding facilitates MU-MIMO by spatially separating user transmissions over common time and frequency resources.

For DL MU-MIMO, a 5G NR base station (gNB) determines the precoder using one of these techniques:

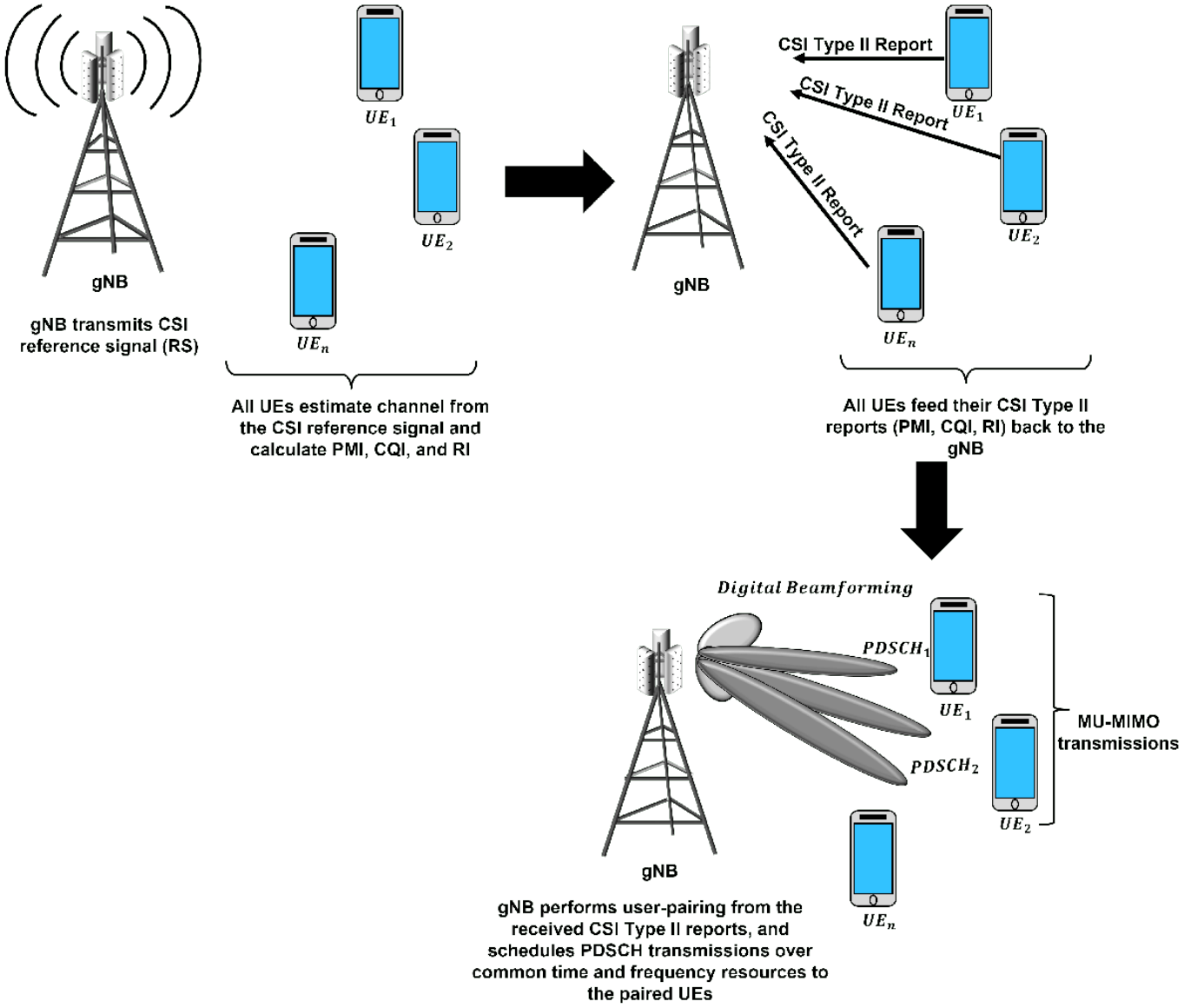
- The gNB uses reported channel state information (CSI) Type II precoders.
- In a time division duplex (TDD) system, the gNB computes the DL precoder from the sounding reference signal (SRS) using channel reciprocity. For more information about this technique, see the “TDD Reciprocity-Based PDSCH MU-MIMO Using SRS” on page 5-32 example.

In this example, the gNB facilitates DL MU-MIMO by using CSI Type II precoding.

This is the communication flow between the gNB and user equipments (UEs).

- The gNB transmits the CSI reference signal (CSI-RS).
- UEs in the same cell estimate the channel from the CSI-RS.
- Each UE determines a rank indicator (RI), precoder matrix indicator (PMI), and channel quality indicator (CQI) from the estimated channel.
- UEs report RI, PMI, and CQI to the gNB.
- The CSI reported by the UE forms the basis for the downlink user-pairing algorithm.
- For the paired UEs, the gNB transmits precoded DL transmissions over common time and frequency resources.

For more information about CSI Type II reports, see the “5G NR Downlink CSI Reporting” on page 5-47 example.



Simulation Assumptions

This example makes these assumptions.

- The number of CSI-RS ports equals the number of transmit antennas on the gNB.
- The round-robin scheduler pairs only UEs that report the same rank. Therefore, the scheduler does not pair two UEs if one of the UEs report a rank of 1 and the other UE reports a rank of 2.
- The gNB transmits retransmission packets as a single-user (SU) MIMO.
- In this example, the scheduler does not support effective Modulation and Coding Scheme (MCS) calculation for paired users. The scheduler uses the MCS corresponding to the CQI reported by the UEs, which can result in higher BLER, especially when you set the `SemiOrthogonalityFactor` to a value less than 1.

- The scheduler uses `SemiOrthogonalityFactor` and the reported PMI indices (i1) for user-pairing.

Simulate Scenario

Check if the Communications Toolbox Wireless Network Simulation Library support package is installed. If the support package is not installed, MATLAB® returns an error with a link to download and install the support package.

```
wirelessnetworkSupportPackageCheck
```

Create a wireless network simulator.

```
rng("default")          % Reset the random number generator
numFrameSimulation = 3; % Simulation time in terms of number of 10 ms frames
networkSimulator = wirelessNetworkSimulator.init;
```

Create the gNB. Specify the transmit power, subcarrier spacing, carrier frequency, channel bandwidth, number of transmit antennas, and receive gain of the gNB.

```
gNBPosition = [0 0 0]; % [x y z] meters position in Cartesian coordinates
gNB = nrGNB(Position=gNBPosition,TransmitPower=34,SubcarrierSpacing=15000, ...
    CarrierFrequency=2.6e9,ChannelBandwidth=5e6,NumTransmitAntennas=32,ReceiveGain=11);
numUEs = 10; % Set the number of UEs
```

Configure the DL MU-MIMO structure mu`MIMOConfigDL` by setting these parameters.

- `MinNumRBs` – Minimum number of resource blocks (RBs) that should be allocated to a UE to be considered for MU-MIMO.
- `MinCQI` – Minimum CQI for a UE to be considered as a MU-MIMO candidate. This examples uses the reported CQI indices, as defined in TS 38.214, Table 5.2.2.1-3 [1 on page 6-139].
- `SemiOrthogonalityFactor` – Orthogonality between the reported PMI of the UEs, specified as a scalar in the range [0, 1]. `SemiOrthogonalityFactor` value of `1` indicates that the scheduler pairs the users with only perfectly orthogonal precoders.
- `MaxNumUsersPaired` – Maximum number of users that can be paired for a MU-MIMO transmission.

```
muMIMODLConfig = struct(MaxNumUsersPaired=4,MinNumRBs=2,SemiOrthogonalityFactor=0.9, ...
    MinCQI=10);
```

Set these scheduler parameters by using the `configureScheduler` function.

- `ResourceAllocationType` – Physical downlink shared channel (PDSCH) resource allocation type, specified as `0` or `1`.
- `MaxNumUsersPerTTI` – Maximum number of users allocated per transmission time interval (TTI).

```
configureScheduler(gNB,ResourceAllocationType=0,MaxNumUsersPerTTI=10, ...
    MUMIMOConfigDL=muMIMODLConfig);
```

Calculate position of all UEs using these specifications.

- Relative distance of the UE from the gNB — `500` (Units are in meters)
- Sweep range of the azimuth angle in the horizontal plane — `[-60 60]` (Units are in degrees)

```
% For all UEs, specify position in spherical coordinates (r,azimuth,elevation)
% relative to gNB.
ueRelPosition = [ones(numUEs,1)*500 (rand(numUEs,1)-0.5)*120 zeros(numUEs,1)];
% Convert spherical to Cartesian coordinates considering gNB position as origin
[xPos,yPos,zPos] = sph2cart(deg2rad(ueRelPosition(:,2)),deg2rad(ueRelPosition(:,3)), ...
    ueRelPosition(:,1));
% Convert to absolute Cartesian coordinates
uePositions = [xPos yPos zPos] + gNBPosition;
ueNames = "UE-" + (1:size(uePositions,1));
```

Create the UEs. Specify the name, position, receiver gain, and the number of receive antennas for each UE.

```
UEs = nrUE(Name=ueNames,Position=uePositions,ReceiveGain=0,NumReceiveAntennas=1);
```

Connect the UEs to the gNB, and configure full-buffer traffic in the DL direction.

```
connectUE(gNB,UEs,FullBufferTraffic="DL")
```

Add the gNB and UEs to the network simulator.

```
addNodes(networkSimulator,gNB)
addNodes(networkSimulator,UEs)
```

Create an *N*-by-*N* matrix of link-level channels. *N* is the number of nodes in the cell. An element at index (*i*, *j*) contains the channel instance from node *i* to node *j*. An empty element at index (*i*, *j*) indicates that channel from node *i* to node *j* does not exist. *i* and *j* denote the node IDs.

```
channelConfig = struct("DelayProfile","CDL-D","DelaySpread",100e-9);
channels = createMultiUserCDLChannels(channelConfig,gNB,UEs);
```

Create a custom channel model using `channels`. Install the custom channel on the simulator. The network simulator applies the channel for a packet in transit before passing the packet to the receiver.

```
customChannelModel = hNRCustomChannelModel(channels);
addChannelModel(networkSimulator,@customChannelModel.applyChannelModel)
```

Set `enableTraces` to `true` to log the traces. If you set `enableTraces` to `false`, the simulator does not log traces. Setting `enableTraces` to `false` can speed up your simulation.

```
enableTraces = true;
```

Set up the scheduling logger and PHY logger.

```
if enableTraces
    % Create an object for scheduler trace logging
    simSchedulingLogger = helperNRSchedulingLogger(numFrameSimulation,gNB,UEs);
    % Create an object for PHY trace logging
    simPhyLogger = helperNRPhyLogger(numFrameSimulation,gNB,UEs);
end
```

The example periodically updates the plotted metrics. Set the number of updates to perform during simulation.

```
% This parameter has an impact on simulation time.
numMetricsSteps = 10*numFrameSimulation;
```

Set up a metric visualizer.

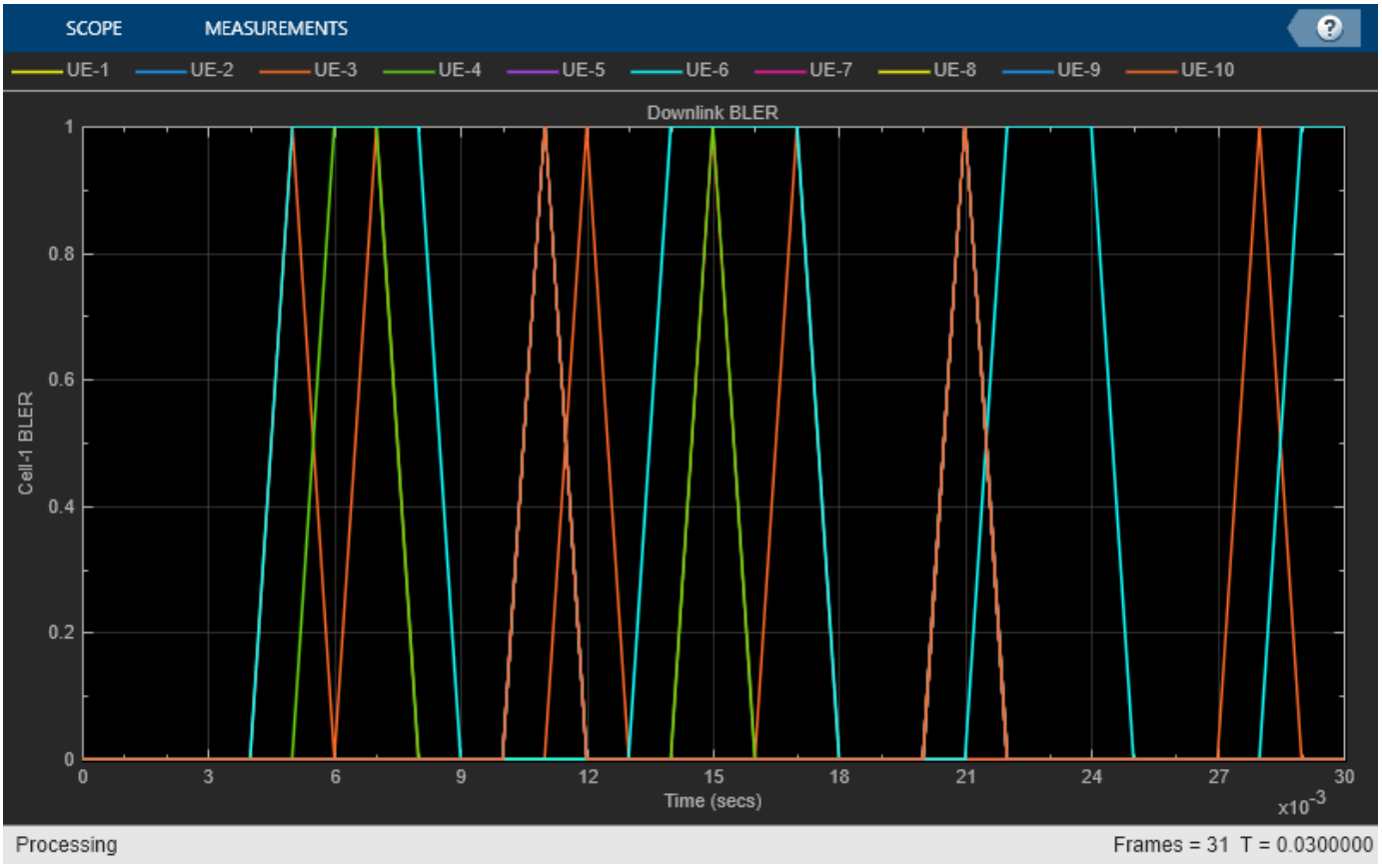
```
metricsVisualizer = helperNRMetricsVisualizer(gNB,UEs,NumMetricsSteps=numMetricsSteps, ...
    PlotSchedulerMetrics=true,PlotPhyMetrics=true,PlotCDFMetrics=true,LinkDirection=0);
```

Write the logs to MAT file. You can use these logs for post-simulation analysis.

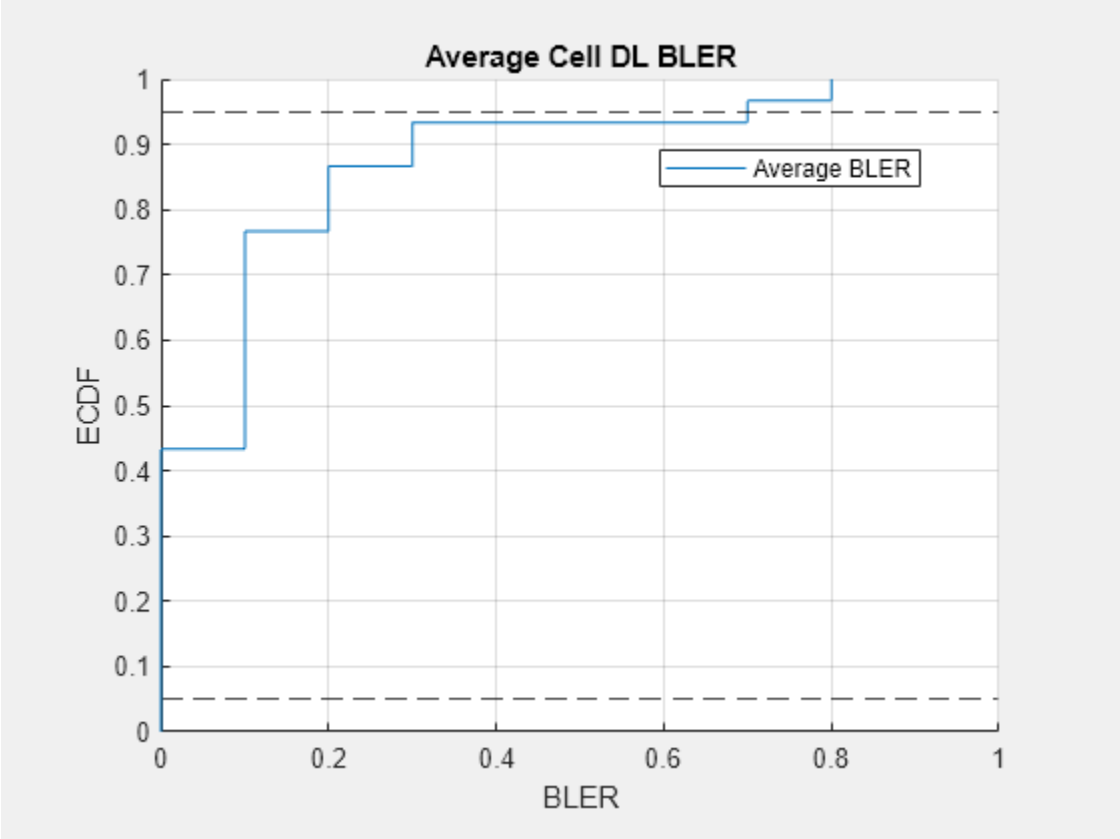
```
simulationLogFile = "simulationLogs"; % For logging the simulation traces
```

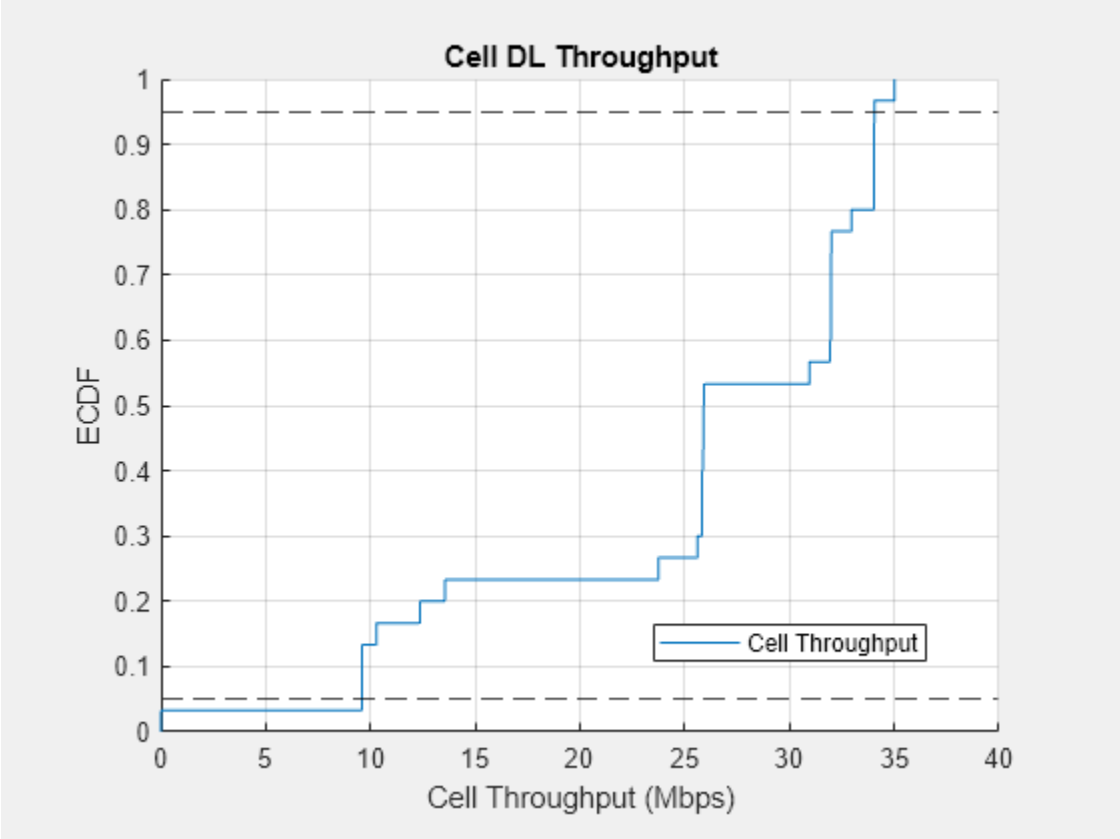
Run the simulation for the specified number of frames `numFrameSimulation`.

```
% Calculate the simulation duration (in seconds)
simulationTime = numFrameSimulation*1e-2;
% Run the simulation
run(networkSimulator,simulationTime)
```




SCOPE
MEASUREMENTS
UE-1 UE-2 UE-3 UE-4 UE-5
UE-6 UE-7 UE-8 UE-9 UE-10
Cell Peak Data Rate
Cell-1 DL Tx rate (Mbps)
Cell-1 DL Resource Share (%)
Cell-1 DL Throughput (Mbps)
Cell-1 DL Buffer Status (KB)
Time (secs)
x10^-3
Processing
Frames = 31 T = 0.0300000


Average Cell DL BLER
Average BLER
ECDF
BLER
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1



Results Summary

Display the system KPIs, including cell throughput, spectral efficiency and, empirical cumulative distribution function (ECDF) plots for cell throughput and average BLER. Using the ECDF plots, you can find the percentage of users with a throughput or BLER that is less than or equal to a value on the *x*-axis.

```
% Read performance metrics
displayPerformanceIndicators(metricsVisualizer)
```

```
Peak DL Throughput: 31.11 Mbps. Achieved Cell DL Throughput: 25.22 Mbps
Achieved DL Throughput for each UE: [2.03        4.75        5.05        2.19        1.92
Peak DL spectral efficiency: 6.22 bits/s/Hz. Achieved DL spectral efficiency for cell: 5.04 bits/
Block error rate for each UE in the downlink direction: [0.071       0.071       0.207       0.1
```

```
if enableTraces
    simulationLogs = cell(1,1);
    if gNB.DuplexMode == "FDD"
        logInfo = struct("DLTimeStepLogs",[],"ULTimeStepLogs",[], ...
            "SchedulingAssignmentLogs",[],"PhyReceptionLogs",[]);
        [logInfo.DLTimeStepLogs,logInfo.ULTimeStepLogs] = getSchedulingLogs(simSchedulingLogger)
    else % TDD
        logInfo = struct("TimeStepLogs",[],"SchedulingAssignmentLogs",[],"PhyReceptionLogs",[]);
        logInfo.TimeStepLogs = getSchedulingLogs(simSchedulingLogger);
    end
    % Get the scheduling assignments log
    logInfo.SchedulingAssignmentLogs = getGrantLogs(simSchedulingLogger);
    % Get the PHY reception logs
```

```
    logInfo.PhyReceptionLogs = getReceptionLogs(simPhyLogger);
    % Save simulation logs in a MAT-file
    simulationLogs{1} = logInfo;
    save(simulationLogFile,"simulationLogs")
end
```

Local Functions

Set up CDL channel instances for the cell.

```
function channels = createMultiUserCDLChannels(channelConfig,gNB,UEs)
    numUEs = length(UEs);
    numNodes = length(gNB) + numUEs;
    channels = cell(numNodes,numNodes);

    waveformInfo = nrOFDMInfo(gNB.NumResourceBlocks,gNB.SubcarrierSpacing/1e3);
    sampleRate = waveformInfo.SampleRate;

    % Create a CDL channel model object configured with the desired delay
    % profile, delay spread, and Doppler frequency
    channel = nrCDLChannel;
    channel.CarrierFrequency = gNB.CarrierFrequency;
    % Delay profile. You can configure it as CDL- A,B,C,D,E
    channel.DelayProfile = channelConfig.DelayProfile;
    channel.DelaySpread = channelConfig.DelaySpread; % Delay Spread
    % Configure antenna down-tilt as 12 (degrees)
    channel.TransmitArrayOrientation = [0 12 0]';
    channel.SampleRate = sampleRate;
    channel.ChannelFiltering = false;

    % For each UE set DL channel instance
    for ueIdx = 1:numUEs

        % Create a copy of the original channel
        cdl = hMakeCustomCDL(channel);

        % Configure the channel seed based on the UE number
        % (results in independent fading for each UE)
        cdl.Seed = 73 + (ueIdx - 1);

        % Set antenna panel
        cdl = hArrayGeometry(cdl,gNB.NumTransmitAntennas,UEs(ueIdx). ...
            NumReceiveAntennas,"downlink");

        % Compute the LOS angle from gNB to UE
        [~,depAngle] = rangeangle(UEs(ueIdx).Position', ...
            gNB.Position');

        % Configure the azimuth and zenith angle offsets for this UE
        cdl.AnglesAoD(:) = cdl.AnglesAoD(:) + depAngle(1);
        % Convert elevation angle to zenith angle
        cdl.AnglesZoD(:) = cdl.AnglesZoD(:) - cdl.AnglesZoD(1) + (90 - depAngle(2));

        channels{gNB.ID,UEs(ueIdx).ID} = cdl;
    end
end
```

Further Explorations

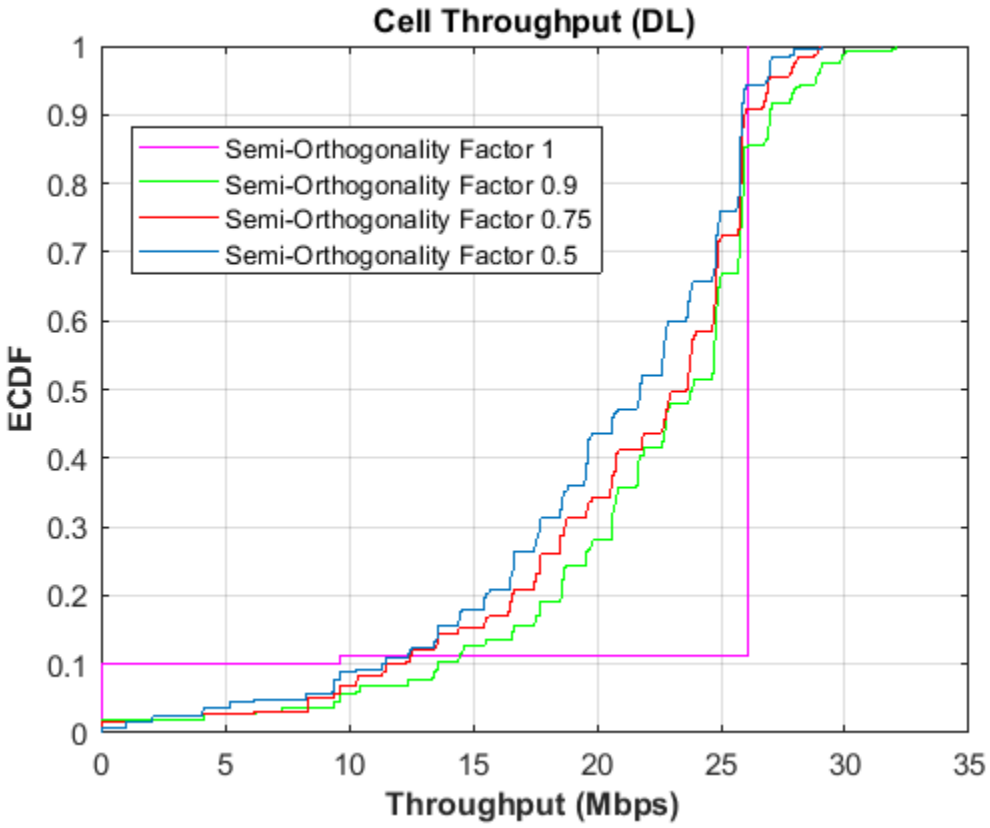
Try running the example for these use cases.

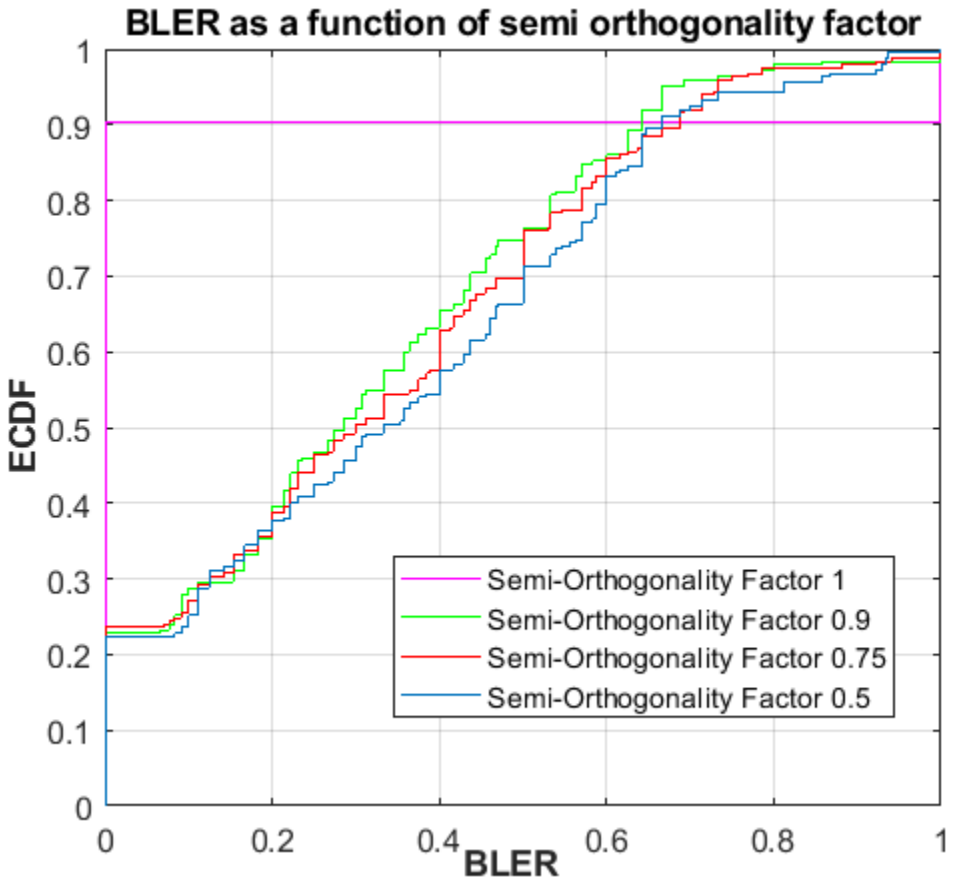
1. Analyze the impact of various CDL delay profiles (CDL-A/B/C/D/E) on the user-pairing strategy and system KPIs.
2. Analyze the impact of antenna configurations, such as CSI antenna ports, panel configuration, antenna orientation, and channel delay profile, on the system KPIs.
3. Modify the DL MU-MIMO configuration, and observe the impact on the system KPIs.

This list shows the relationship between `SemiOrthogonalityFactor` and BLER for a static scenario with 40 UEs. This simulation setup does not compute effective MCS if the scheduler pairs the users.

Semi Orthogonality Factor	Avg. BLER
1	0.1
0.9	0.3
0.75	0.33
0.5	0.36

This figure illustrates the impact of the `SemiOrthogonalityFactor` on cell throughput and average BLER. For the considered scenario, orthogonal users does not exists when you set `SemiOrthogonalityFactor` to `1` and use CSI Type II precoder.


Cell Throughput (DL)
ECDF
Throughput (Mbps)
Semi-Orthogonality Factor 1
Semi-Orthogonality Factor 0.9
Semi-Orthogonality Factor 0.75
Semi-Orthogonality Factor 0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
30
35


BLER as a function of semi orthogonality factor
ECDF
BLER
Semi-Orthogonality Factor 1
Semi-Orthogonality Factor 0.9
Semi-Orthogonality Factor 0.75
Semi-Orthogonality Factor 0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.2
0.4
0.6
0.8
1

Appendix

The example uses these helper classes and functions:

- `helperNRMetricsVisualizer.m` – Implements metrics visualization functionality
- `helperNRSchedulingLogger.m` – Implements scheduling information logging functionality
- `helperNRPhyLogger.m` – Implements PHY packet reception information logging functionality
- `hNRCustomChannelModel.m` – Implements channel modeling functionality
- `hMakeCustomCDL.m` – Configures CDL with custom delay profile
- `hArrayGeometry.m` – Configures antenna array geometry for CDL channel model

References

1 3GPP TS 38.214. "NR; Physical layer procedures for data." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Objects
`wirelessNetworkSimulator` | `nrGNB` | `nrUE`

Related Examples

- “NR Cell Performance Evaluation with MIMO” on page 6-41
- “5G NR Downlink CSI Reporting” on page 5-47
- “TDD Reciprocity-Based PDSCH MU-MIMO Using SRS” on page 5-32

Test and Measurement

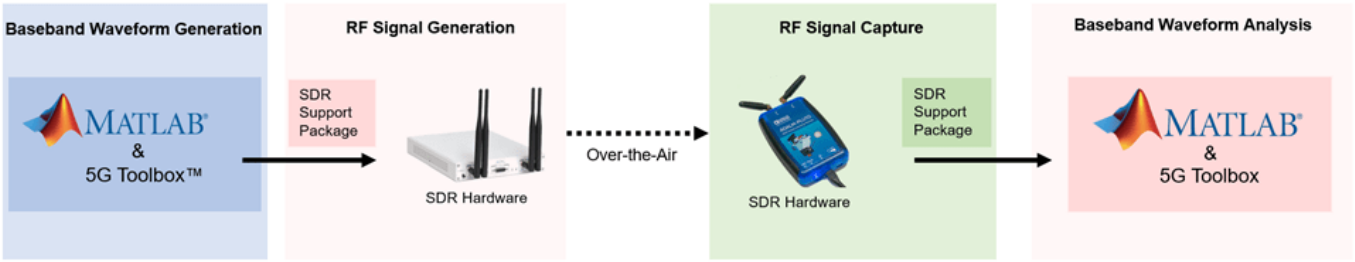
5G NR Waveform Capture and Analysis Using Software-Defined Radio

This example shows how to use the 5G Waveform Generator app to generate and transmit a standard-compliant 5G NR waveform continuously over the air using a software-defined radio (SDR). The example then shows how to capture the transmitted waveform from the air using an SDR and analyze the signal in MATLAB®.

Introduction

This diagram shows the main steps of the example.

1. Generate a baseband waveform by using the **5G Waveform Generator** app.
2. Transmit the generated waveform over the air, directly from the app, by using a connected SDR hardware (requires an SDR support package).
3. Capture the over-the-air waveform in MATLAB by using another connected SDR hardware or the same SDR hardware (requires an SDR support package).
4. Analyze the PDCCH/PDSCH error vector magnitude (EVM) of the captured waveform in MATLAB.



To transmit and capture waveforms with SDRs, you must install the corresponding hardware support package. This list provides information on which radios can be used by this example along with the required products.

- ADALM-PLUTO (requires Communications Toolbox Support Package for Analog Devices® ADALM-PLUTO Radio)
- USRP™ B200/B210/N200/N210/USRP2/N300/N310/N320/N321/X300/X310 (requires Communications Toolbox Support Package for USRP™ Radio)
- USRP™ E310/E312 (requires Communications Toolbox Support Package for USRP™ Embedded Series Radio)
- AD9361/FMCOMMS2/3/4/5 (requires Communications Toolbox Support Package for Xilinx® Zynq®-Based Radio)

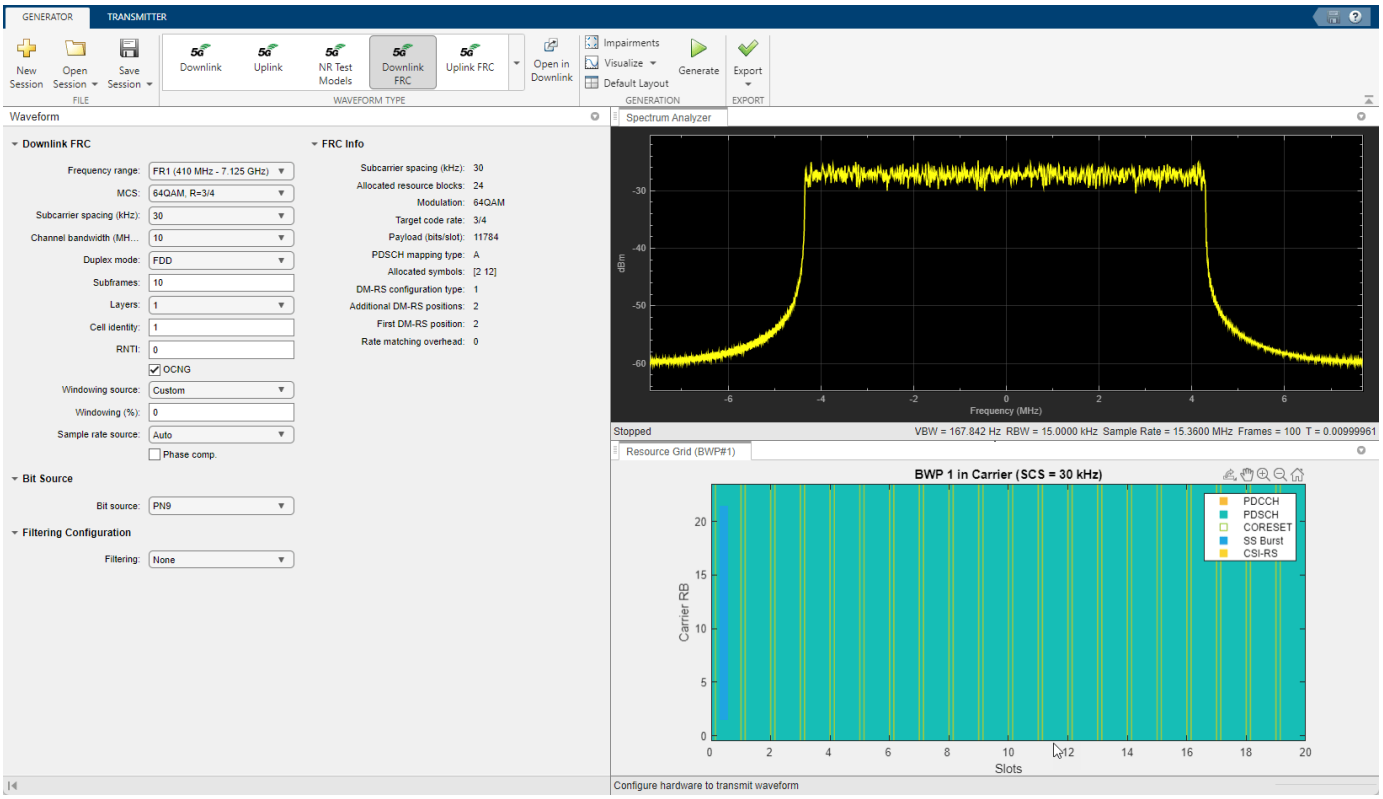
Generate Baseband Waveform

In MATLAB, on the **Apps** tab, click the **5G Waveform Generator** app.

In the **Waveform Type** section, click **Downlink FRC**. In the leftmost pane of the app, you can set the parameters for the selected waveform. For this example:

- Set **Frequency range** to `FR1 (410 MHz - 7.125 GHz)`.
- Set **MCS** to `64QAM, R=3/4`.
- Set **Subcarrier spacing (kHz)** to `30`.
- Set **Channel bandwidth (MHz)** to `10`.
- Set **Duplex mode** to `FDD`.
- Set **RNTI** to 0.
- Select the **OCNG** check box.

On the app toolstrip, click **Generate**.

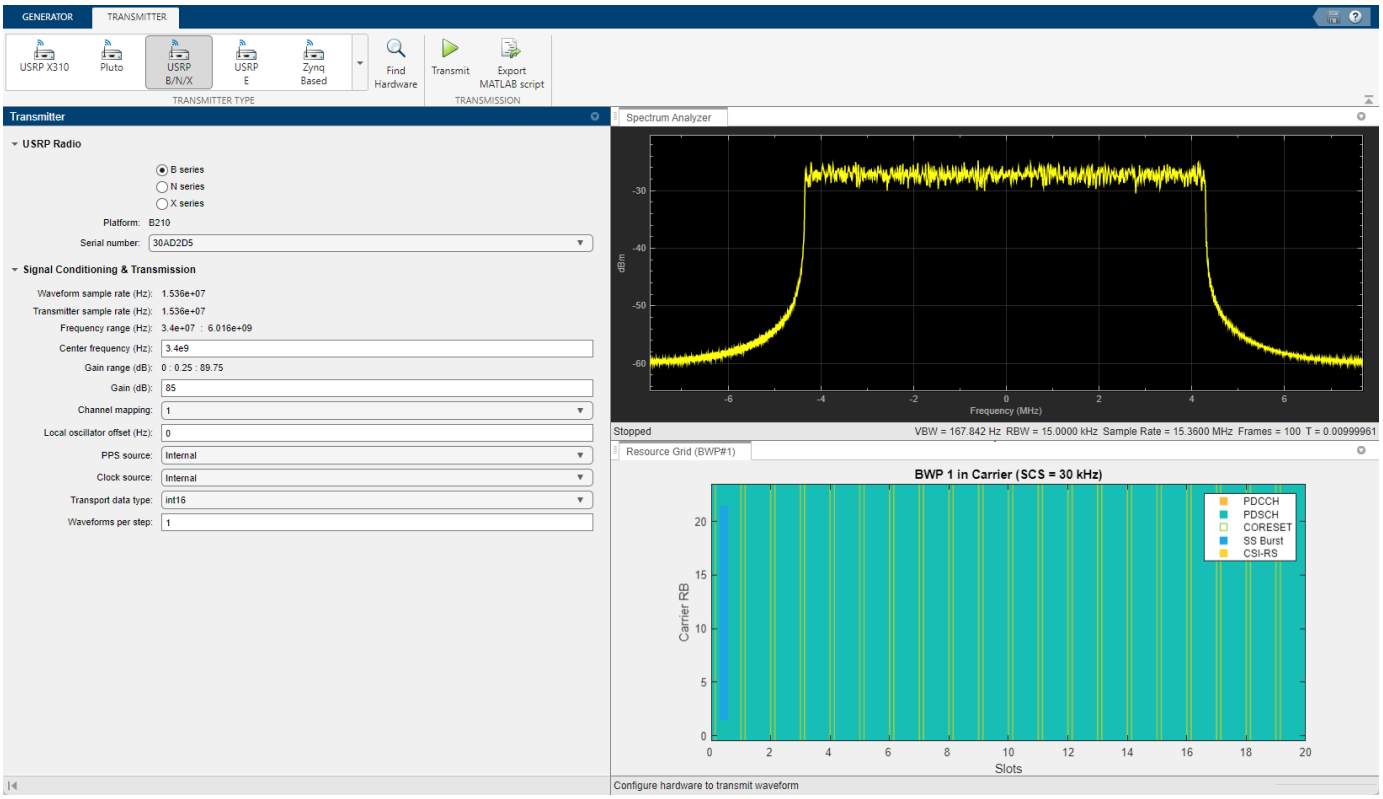


Transmit Generated Waveform

On the **Transmitter** tab of the app, in the **Transmitter Type** section, select your SDR for waveform transmission.

- If you have an ADALM-PLUTO radio corresponding to the Communications Toolbox Support Package for Analog Devices® ADALM-PLUTO Radio, select `Pluto`.
- If you have a USRP™ B200, B210, N200, N210, USRP2, N300, N310, N321, X300, or X310 radio corresponding to the Communications Toolbox Support Package for USRP™ Radio, select `USRP B/N/X`. In this case, this examples requires a secondary SDR and MATLAB session to capture the transmitted signal. Otherwise, you can use the same SDR for reception as you do for transmission.
- If you have a USRP™ E310 or E312 radio corresponding to the Communications Toolbox Support Package for USRP™ Embedded Series Radio, select `USRP E`.
- If you have a AD936x or FMCOMMS5 radio corresponding to the Communications Toolbox Support Package for Xilinx® Zynq®-Based Radio, select `Zynq Based`.

Specify your transmission parameters, such as **Center frequency (Hz)** and **Gain (dB).** The app automatically obtains the baseband sample rate from the generated waveform and performs any oversampling, if necessary.



Capture Transmitted Waveform

Set Up Receiver Processing Parameters

Set the downlink waveform parameters used in the app for receiver processing.

```
rc   = DL-FRC-FR1-64QAM ; % Reference channel
bw   = 10MHz            ; % Channel bandwidth
scs  = 30kHz            ; % Subcarrier spacing
dm   = FDD              ; % Duplexing mode
rnti = 0                ; % Radio network temporary identifier
```

Configure SDR Receiver Object

To capture the waveform that the app continously transmits, create an `hSDRReceiver` object and then set the object properties.

- Set `rx.CenterFrequency` to `3.4e9`, as specified in the **Transmitter** tab of the app.
- Set `rx.SampleRate` to a value greater than or equal to the **Transmitter sample rate (Hz)** parameter value in the app.

- Set `rx.Gain` to any valid value that provides a successful decode of the captured waveform. Valid values of gain include `'AGC Fast Attack'`, `'AGC Slow Attack'`, or an integer value for the Zynq-Based, USRP Embedded, or ADALM-PLUTO radios. USRP radios only support integer values.

```
rx    = hSDRReceiver(Pluto); % SDR receiver object
rx.CenterFrequency = 3.4e9;
rx.SampleRate      = 30.72e6;
rx.Gain            = 60;
```

Specify the number of contiguous 5G frames to capture. One frame equates to 10 milliseconds.

```
framesToCapture    = 1;

% Derived parameters
captureDuration = milliseconds(10)*(framesToCapture+1); % Increase capture frame by 1 to account
```

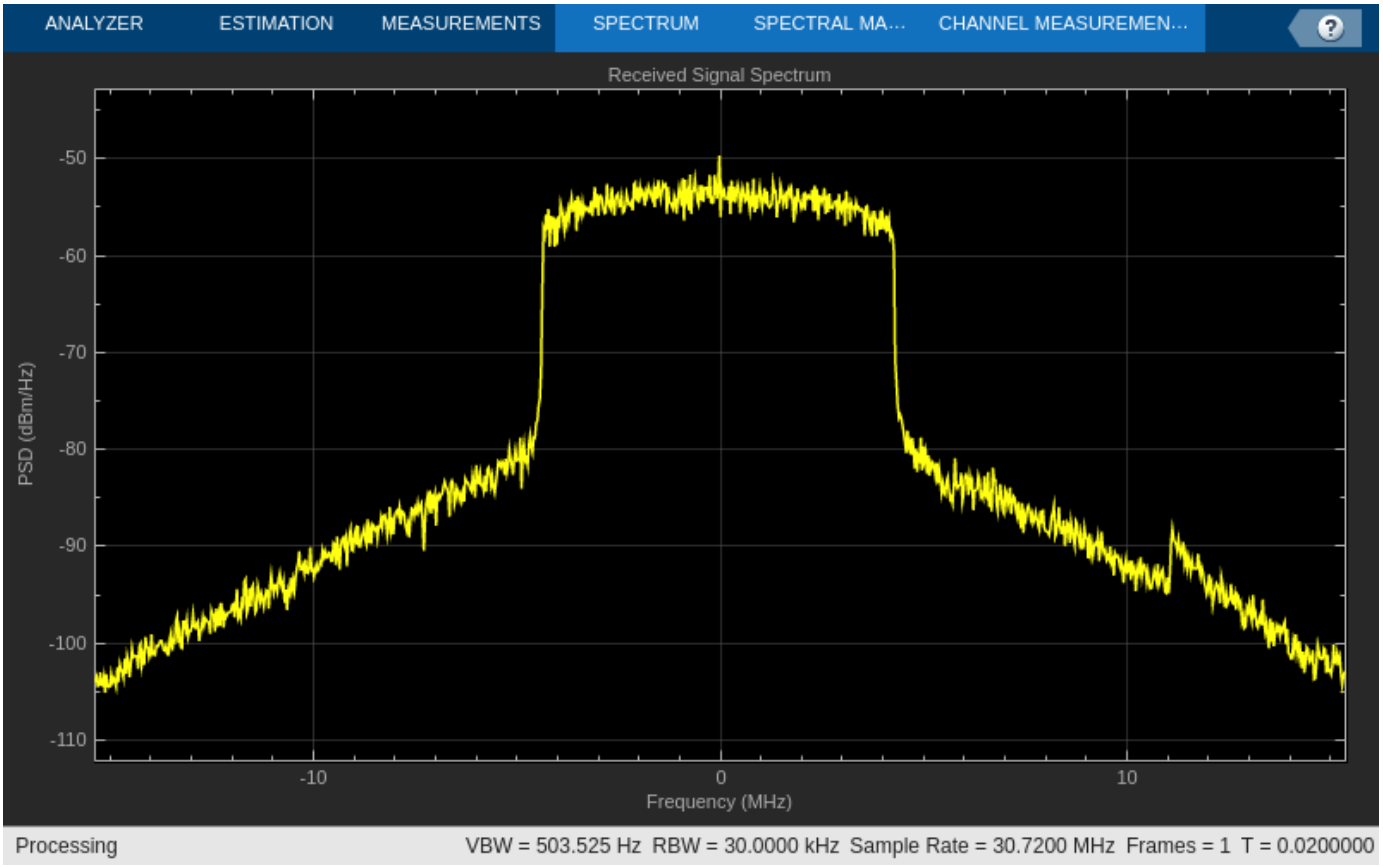
Initiate Waveform Capture

Initiate a capture of the 5G NR waveform transmitted by the app.

```
rxWaveform = capture(rx,captureDuration);

## Establishing connection to hardware. This process can take several seconds.
```

Plot the power spectral density (PSD) of the received signal.

```
spectrumPlotRx = spectrumAnalyzer;
spectrumPlotRx.SampleRate =  rx.SampleRate;
spectrumPlotRx.SpectrumType = "Power density";
spectrumPlotRx.YLabel = "PSD";
spectrumPlotRx.Title = "Received Signal Spectrum";
spectrumPlotRx(rxWaveform);
```



Measure EVM of Received Waveform

Generate and extract a `nrDLCarrierConfig` object for a specific test model (TM) or fixed reference channel (FRC) using the `hNRReferenceWaveformGenerator` helper file.

```
rcwavegen = hNRReferenceWaveformGenerator(rc,bw,scs,dm);
cfgDL = rcwavegen.Config;
```

Use the `hNRDownlinkEVM` function to analyze the waveform. In this example, the function performs these steps.

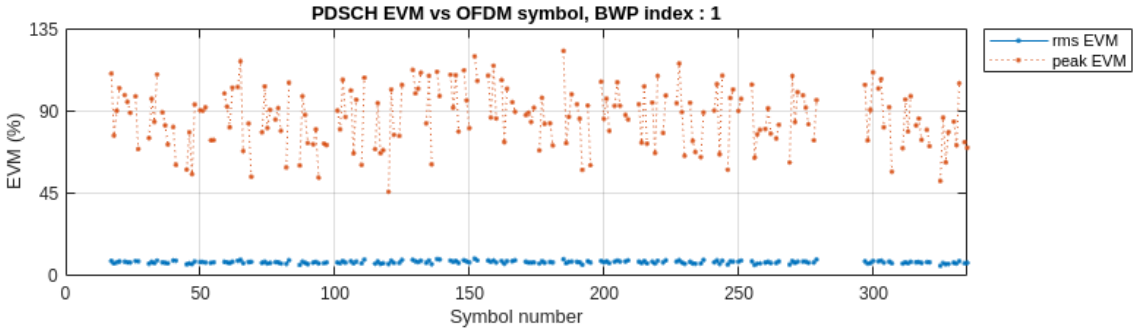
- Estimates and compensates for any frequency offset.
- Estimates and corrects I/Q imbalance.
- Synchronizes the DM-RS over one frame for frequency division duplexing (FDD) or two frames for time division duplexing (TDD).
- Demodulates the received waveform.
- Estimates the channel.
- Equalizes the symbols.
- Estimates and compensates for common phase error (CPE).
- Computes the physical downlink shared channel (PDSCH) EVM.
- Computes the physical downlink control channel (PDCCH) EVM.

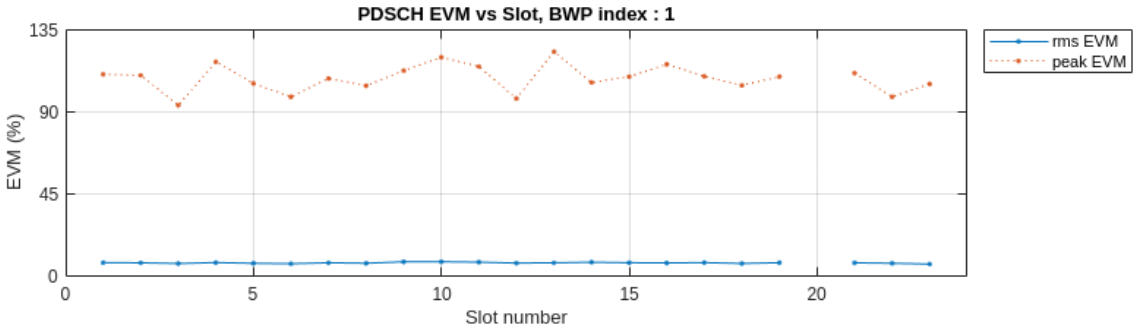
For more information on the hNRDownlinkEVM function, see the “EVM Measurement of 5G NR Downlink Waveforms with RF Impairments” on page 7-79 example.

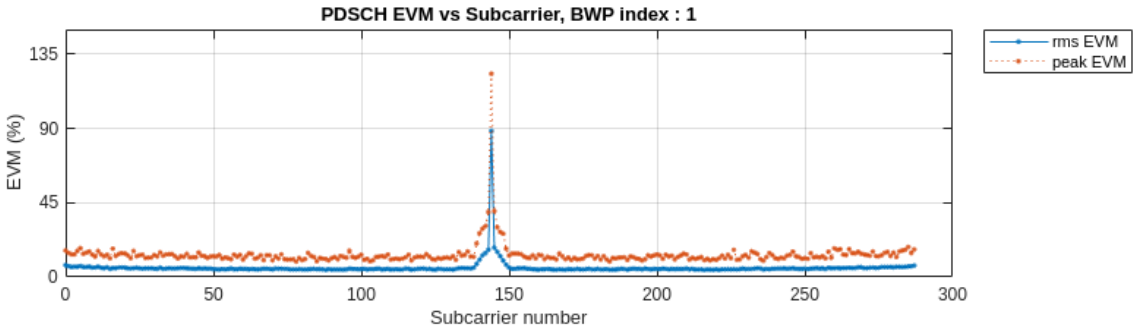
Define the configuration settings for the hNRPDSCHEVM function.

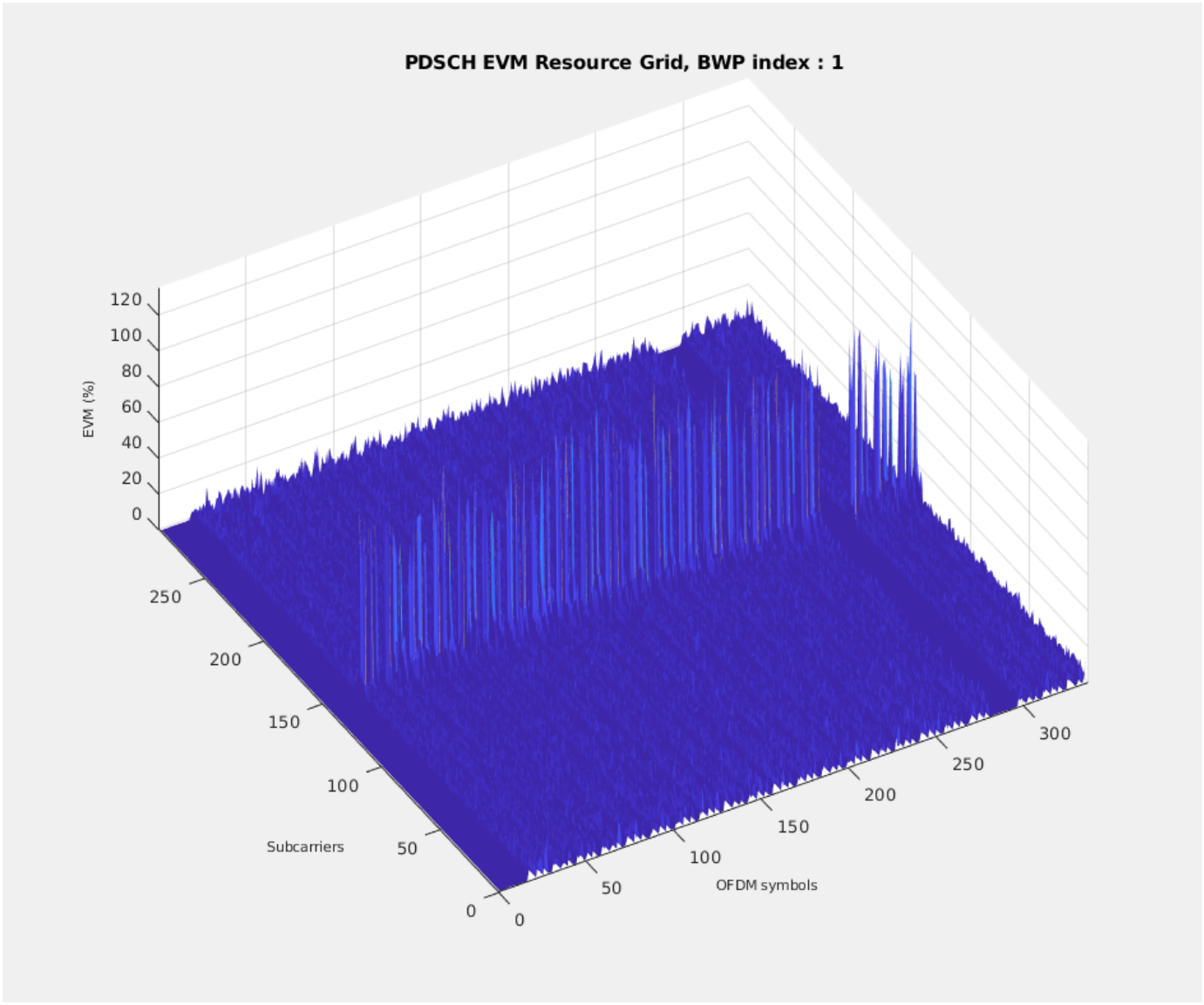
```
cfg = struct();
cfg.PlotEVM = true;             % Plot EVM statistics
cfg.DisplayEVM = true;          % Print EVM statistics
cfg.Label = rc;                 % Set to TM name of captured waveform
cfg.SampleRate = rx.SampleRate; % Use sample rate during capture
cfg.IQImbalance = true;
cfg.TargetRNTIs = rnti;
cfg.CorrectCoarseFO = true;
cfg.CorrectFineFO = true;

[evmInfo,eqSym,refSym] = hNRDownlinkEVM(cfgDL,rxWaveform,cfg);
```

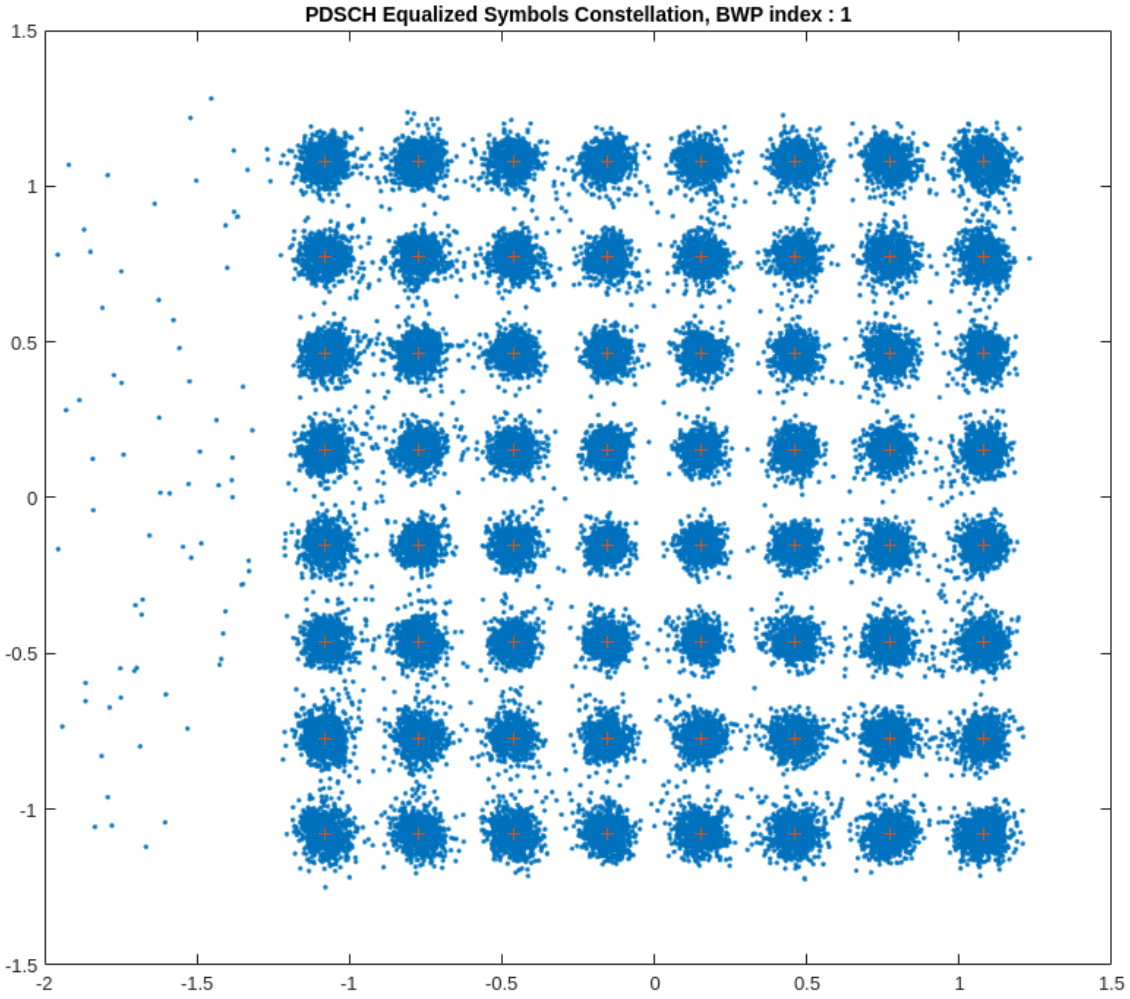
```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 1: 7.744 110.834%
PDSCH RMS EVM, Peak EVM, slot 2: 7.645 110.288%
PDSCH RMS EVM, Peak EVM, slot 3: 7.256 93.874%
PDSCH RMS EVM, Peak EVM, slot 4: 7.766 117.640%
PDSCH RMS EVM, Peak EVM, slot 5: 7.388 105.840%
PDSCH RMS EVM, Peak EVM, slot 6: 7.154 98.483%
PDSCH RMS EVM, Peak EVM, slot 7: 7.664 108.556%
PDSCH RMS EVM, Peak EVM, slot 8: 7.427 104.623%
PDSCH RMS EVM, Peak EVM, slot 9: 8.205 112.831%
PDSCH RMS EVM, Peak EVM, slot 10: 8.218 120.166%
PDSCH RMS EVM, Peak EVM, slot 11: 8.015 115.119%
PDSCH RMS EVM, Peak EVM, slot 12: 7.493 97.563%
PDSCH RMS EVM, Peak EVM, slot 13: 7.659 123.200%
PDSCH RMS EVM, Peak EVM, slot 14: 7.972 106.357%
PDSCH RMS EVM, Peak EVM, slot 15: 7.704 109.554%
PDSCH RMS EVM, Peak EVM, slot 16: 7.557 116.319%
PDSCH RMS EVM, Peak EVM, slot 17: 7.745 109.726%
PDSCH RMS EVM, Peak EVM, slot 18: 7.268 104.755%
PDSCH RMS EVM, Peak EVM, slot 19: 7.697 109.558%
PDSCH RMS EVM, Peak EVM, slot 21: 7.655 111.468%
PDSCH RMS EVM, Peak EVM, slot 22: 7.394 98.480%
PDSCH RMS EVM, Peak EVM, slot 23: 6.969 105.538%
Averaged RMS EVM frame 0: 7.667%
```


PDSCH EVM vs OFDM symbol, BWP index : 1
EVM (%)
Symbol number
rms EVM
peak EVM
0
45
90
135
0
50
100
150
200
250
300


PDSCH EVM vs Slot, BWP index : 1
EVM (%)
Slot number
rms EVM
peak EVM
0
45
90
135
0
5
10
15
20


PDSCH EVM vs Subcarrier, BWP index : 1
EVM (%)
Subcarrier number
rms EVM
peak EVM
0
45
90
135
0
50
100
150
200
250
300


PDSCH EVM Resource Grid, BWP index : 1
EVM (%)
120
100
80
60
40
20
0
Subcarriers
250
200
150
100
50
0
OFDM symbols
0
50
100
150
200
250
300




```
Averaged overall PDSCH RMS EVM: 7.667%
Overall PDSCH Peak EVM = 123.2002%
```

The output of the `hNRDownlinkEVM` function shows that the demodulation of the received waveform is successful. The interference from the DC component of the SDR to the DC subcarrier causes high EVM values in the measurements.

See Also

Apps
5G Waveform Generator

Related Examples

- “5G NR Waveform Acquisition and Analysis” on page 7-135

More About

- “Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio”
- “Communications Toolbox Support Package for USRP Radio”
- “Communications Toolbox Support Package for USRP Embedded Series Radio”
- “Communications Toolbox Support Package for Xilinx Zynq-Based Radio”

5G NR-TM and FRC Waveform Generation

This example shows how to generate standard-compliant 5G NR test models (NR-TMs) and uplink and downlink fixed reference channels (FRCs) for frequency range 1 (FR1) and frequency range 2 (FR2). For the NR-TM and FRC waveform generation, you can specify the NR-TM or FRC name, the channel bandwidth, the subcarrier spacing, and the duplexing mode.

Introduction

The 3GPP 5G NR standard defines sets of link and waveform configurations for the purposes of conformance testing. Two specific types of downlink conformance waveforms are NR test models (NR-TM), for the purpose of base station (BS) RF testing, and downlink fixed reference channels (FRC), for user equipment (UE) input testing.

The NR-TMs for FR1 are defined in TS 38.141-1 Section 4.9.2, and the NR-TMs for FR2 are defined in TS 38.141-2 Section 4.9.2.

They are used in a range of RF tests, including:

- BS output power
- Timing alignment error (TAE)
- Occupied bandwidth emissions
- Adjacent channel leakage ratio (ACLR)
- Operating band unwanted emissions
- Transmitter spurious emissions
- Transmitter intermodulation

Specific test models are aimed at specific sets of measurements.

The physical downlink shared channel (PDSCH) FRC for FR1 are defined in TS 38.101-1 Annex A.3, and for FR2 are defined in TS 38.101-2 Annex A.3.

They are used in a number of UE tests, including:

- UE receiver requirements
- Maximum UE input level testing

The physical uplink shared channel (PUSCH) FRC for FR1 and FR2 are defined in TS 38.104 Annex A.

They are used in a number of base station reception tests, including:

- Reference sensitivity
- Adjacent channel selectivity (ACS)
- In-band and out-of-band blocking
- Receiver intermodulation
- In-channel selectivity
- Dynamic range
- Performance requirements

NR-TMs and FRCs are defined across a standardized set of transmission bandwidth configurations for a valid range of channel bandwidth and subcarrier spacing combinations.

This reference application example uses the MATLAB class `hNRReferenceWaveformGenerator`. This class provides access to the bandwidth configuration tables, the Release 15, 16, and 17 test model and FRC lists, and provides baseband waveform generation and resource grid visualization.

The `hNRReferenceWaveformGenerator` class contains two constant MATLAB table properties. The `FR1BandwidthTable` property contains the FR1 transmission bandwidth configurations defined in TS 38.104 Table 5.3.2-1. See also the FR1 maximum transmission bandwidth configurations defined in TS 38.101-1 Table 5.3.2-1. The `FR2BandwidthTable` property contains the FR2 transmission bandwidth configurations defined in TS 38.104 Tables 5.3.2-2 and 5.3.2-3. See also the FR2 maximum transmission bandwidth configurations defined in TS 38.101-2 Table 5.3.2-1.

```
% NR transmission bandwidth configurations
fr1bandwidthtable = hNRReferenceWaveformGenerator.FR1BandwidthTable
```

```
fr1bandwidthtable=3×15 table
              5MHz    10MHz    15MHz    20MHz    25MHz    30MHz    35MHz    40MHz    45MHz    50MH
              ____    _____    _____    _____    _____    _____    _____    _____    _____    ___

    15kHz      25       52       79      106      133      160      188      216      242      270
    30kHz      11       24       38       51       65       78       92      106      119      133
    60kHz     NaN       11       18       24       31       38       44       51       58       65
```

```
fr2bandwidthtable = hNRReferenceWaveformGenerator.FR2BandwidthTable
```

```
fr2bandwidthtable=4×7 table
              50MHz    100MHz    200MHz    400MHz    800MHz    1600MHz    2000MHz
              _____    ______    ______    ______    ______    _______    _______

    60kHz       66       132       264       NaN       NaN       NaN        NaN
    120kHz      32        66       132       264       NaN       NaN        NaN
    480kHz     NaN       NaN       NaN        66       124       248        NaN
    960kHz     NaN       NaN       NaN        33        62       124        148
```

The `hNRReferenceWaveformGenerator` class also contains two constant properties which list the test model names for FR1 (TS 38.141-1 Section 4.9.2) and test model names for FR2 (TS 38.141-2 Section 4.9.2).

```
% Release 15, 16, and 17 NR-TM test models for FR1 and FR2
fr1testmodels = hNRReferenceWaveformGenerator.FR1TestModels
```

```
fr1testmodels = 10x1 string
    "NR-FR1-TM1.1"
    "NR-FR1-TM1.2"
    "NR-FR1-TM2"
    "NR-FR1-TM2a"
    "NR-FR1-TM2b"
    "NR-FR1-TM3.1"
    "NR-FR1-TM3.1a"
    "NR-FR1-TM3.1b"
    "NR-FR1-TM3.2"
    "NR-FR1-TM3.3"
```

```
fr2testmodels = hNRReferenceWaveformGenerator.FR2TestModels
```

```
fr2testmodels = 5x1 string
    "NR-FR2-TM1.1"
    "NR-FR2-TM2"
    "NR-FR2-TM2a"
    "NR-FR2-TM3.1"
    "NR-FR2-TM3.1a"
```

For the downlink FRCs, the class contains additional constant properties which list the downlink FRC names for FR1 (TS 38.101-1 Annex A.3) and for FR2 (TS 38.101-2 Annex A.3).

```
% Release 15, 16, and 17 downlink fixed reference channels for FR1 and FR2
fr1downlinkfrc = hNRReferenceWaveformGenerator.FR1DownlinkFRC
```

```
fr1downlinkfrc = 4x1 string
    "DL-FRC-FR1-QPSK"
    "DL-FRC-FR1-64QAM"
    "DL-FRC-FR1-256QAM"
    "DL-FRC-FR1-1024QAM"
```

```
fr2downlinkfrc = hNRReferenceWaveformGenerator.FR2DownlinkFRC
```

```
fr2downlinkfrc = 3x1 string
    "DL-FRC-FR2-QPSK"
    "DL-FRC-FR2-16QAM"
    "DL-FRC-FR2-64QAM"
```

For the uplink FRCs, the class contains two constant properties which list the uplink FRC names for FR1 and FR2 (TS 38.104 Annex A).

```
% Release 15 uplink fixed reference channels for FR1 and FR2
fr1uplinkfrc = hNRReferenceWaveformGenerator.FR1UplinkFRC
```

```
fr1uplinkfrc = 89x1 string
    "G-FR1-A1-1"
    "G-FR1-A1-2"
    "G-FR1-A1-3"
    "G-FR1-A1-4"
    "G-FR1-A1-5"
    "G-FR1-A1-6"
    "G-FR1-A1-7"
    "G-FR1-A1-8"
    "G-FR1-A1-9"
    "G-FR1-A2-1"
    "G-FR1-A2-2"
    "G-FR1-A2-3"
    "G-FR1-A2-4"
    "G-FR1-A2-5"
    "G-FR1-A2-6"
    "G-FR1-A3-1"
    "G-FR1-A3-2"
    "G-FR1-A3-3"
    "G-FR1-A3-4"
    "G-FR1-A3-5"
    "G-FR1-A3-6"
```

```
    "G-FR1-A3-7"
    "G-FR1-A3-8"
    "G-FR1-A3-9"
    "G-FR1-A3-10"
    "G-FR1-A3-11"
    "G-FR1-A3-12"
    "G-FR1-A3-13"
    "G-FR1-A3-14"
    "G-FR1-A3-15"
      ⋮
```

```
fr2uplinkfrc = hNRReferenceWaveformGenerator.FR2UplinkFRC
```

```
fr2uplinkfrc = 37x1 string
    "G-FR2-A1-1"
    "G-FR2-A1-2"
    "G-FR2-A1-3"
    "G-FR2-A1-4"
    "G-FR2-A1-5"
    "G-FR2-A3-1"
    "G-FR2-A3-2"
    "G-FR2-A3-3"
    "G-FR2-A3-4"
    "G-FR2-A3-5"
    "G-FR2-A3-6"
    "G-FR2-A3-7"
    "G-FR2-A3-8"
    "G-FR2-A3-9"
    "G-FR2-A3-10"
    "G-FR2-A3-11"
    "G-FR2-A3-12"
    "G-FR2-A4-1"
    "G-FR2-A4-2"
    "G-FR2-A4-3"
    "G-FR2-A4-4"
    "G-FR2-A4-5"
    "G-FR2-A4-6"
    "G-FR2-A4-7"
    "G-FR2-A4-8"
    "G-FR2-A4-9"
    "G-FR2-A4-10"
    "G-FR2-A5-1"
    "G-FR2-A5-2"
    "G-FR2-A5-3"
      ⋮
```

For more information, access the help of `hNRReferenceWaveformGenerator` by typing `'doc hNRReferenceWaveformGenerator'`.

NR-TM and PDSCH FRC Waveform Generation

Each PDSCH reference waveform is defined by a combination of:

- NR-TM or FRC name
- Channel bandwidth

- Subcarrier spacing
- Duplexing mode

Different NR-TMs are defined for FR1 and FR2. Depending on the test model purposes, NR-TMs have varying PDSCH characteristics. For example: full band, single modulation scheme, or full band, multiple modulation schemes with varying power boosting/deboosting or single, varying PRB allocation. Common features to all NR-TMs are: no SS burst, PDSCH mapping type A with one (FR2) or two (FR1) DM-RS positions per slot transmission, and a single PDCCH across two symbols with NCCE = 1. There is no transport or DCI coding used and the input to the PDSCH and PDCCH is all 0's or PN23. FDD NR-TM waveforms are 10 ms in length and TDD cases are 20 ms. PT-RS are specified for FR2 NR-TM.

By comparison, downlink FRC waveforms contain transport coded PDSCH using RV = 0. The reference PDSCH are not defined in slots which overlap the SS burst (slot 0 or slots 0 and 1). They use front loaded PDSCH mapping type A with 2 additional DM-RS positions. There is no FDM between the PDSCH and the DM-RS. The full-band PDSCH start at symbol 2 and the first 2 symbols in a slot contain a full occupied CORESET. The FRC waveforms generated in this example do not contain additional OCNG. Power levels for all resource elements are uniform. The transport block data source is ITU PN9.

The channel bandwidth and subcarrier spacing combination have to be a valid pair from the associated FR bandwidth configuration table. The standard only defines FR2 NR-TM and FRC for TDD but with this example you can also create FDD waveforms.

This MATLAB code creates an `hNRReferenceWaveformGenerator` object for the selected NR-TM or FRC configuration. You can use this object to generate the associated baseband waveform and to display the underlying PRB and subcarrier-level resource grids.

```
% Select the NR-TM or PDSCH FRC waveform parameters
dlnrref = NR-FR1-TM3.2 (...  ;  % Model name and properties
bw      = 10MHz (FR1)        ;  % Channel bandwidth
scs     = 15kHz (FR1)        ;  % Subcarrier spacing
dm      = FDD                ;  % Duplexing mode
ncellid = 1                  ;  % NCellID

% Run this entire section to generate the required waveform
Generate

% Create generator object for the above NR-TM/PDSCH FRC reference model
dlrefwavegen = hNRReferenceWaveformGenerator(dlnrref,bw,scs,dm,ncellid)

dlrefwavegen =
  hNRReferenceWaveformGenerator with properties:

    FR1BandwidthTable: [3x15 table]
    FR2BandwidthTable: [4x7 table]
        FR1TestModels: [10x1 string]
        FR2TestModels: [5x1 string]
       FR1DownlinkFRC: [4x1 string]
       FR2DownlinkFRC: [3x1 string]
```

```
         FR1UplinkFRC: [89x1 string]
         FR2UplinkFRC: [37x1 string]
               Config: [1x1 nrDLCarrierConfig]
           IsReadOnly: 1
      ConfiguredModel: {["NR-FR1-TM3.2"]  ["10MHz"]  ["15kHz"]  ["FDD"]  [1]  ["17.8.0"]}
           TargetRNTI: 1
```

```
% Generate waveform
[dlrefwaveform,dlrefwaveinfo,dlresourceinfo] = generateWaveform(dlrefwavegen);

% View transmission information about the set of PDSCH within the waveform
dlresourceinfo.WaveformResources.PDSCH
```

```
ans=1×3 struct array with fields:
    Name
    CDMLengths
    Resources
```

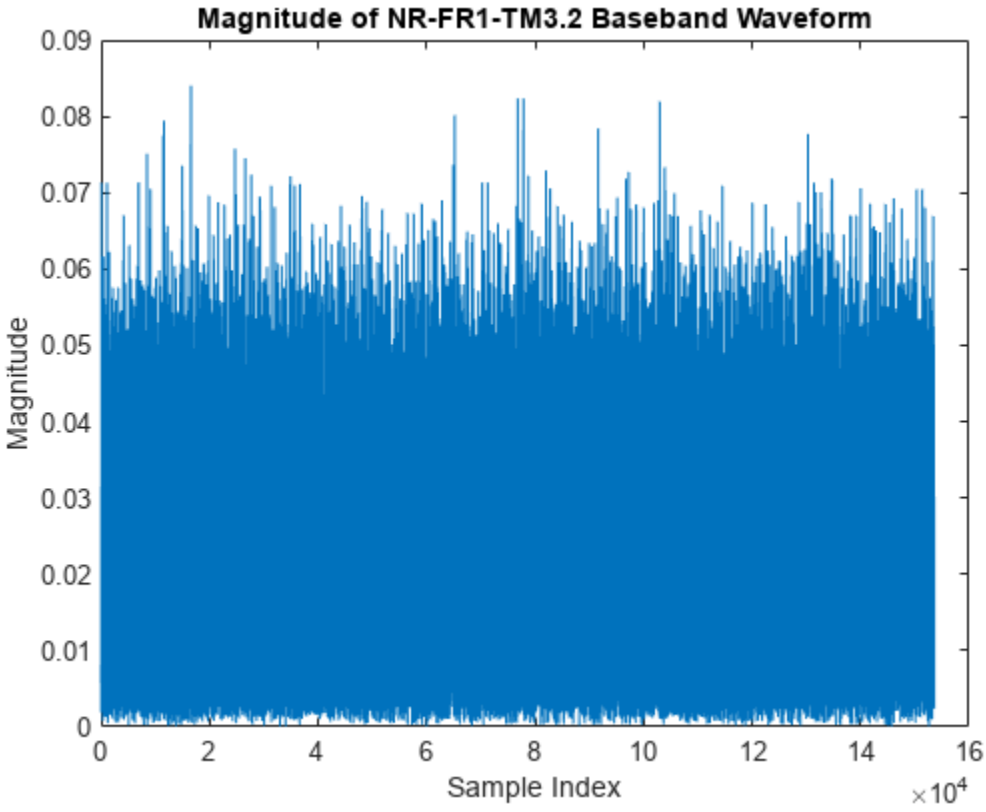
```
% View detailed information about one of the PDSCH sequences
dlresourceinfo.WaveformResources.PDSCH(1).Resources
```

```
ans=1×10 struct array with fields:
    NSlot
    TransportBlockSize
    TransportBlock
    RV
    Codeword
    G
    Gd
    ChannelIndices
    ChannelSymbols
    DMRSIndices
    DMRSSymbols
    DMRSSymbolSet
    PTRSIndices
    PTRSSymbols
    PTRSSymbolSet
```

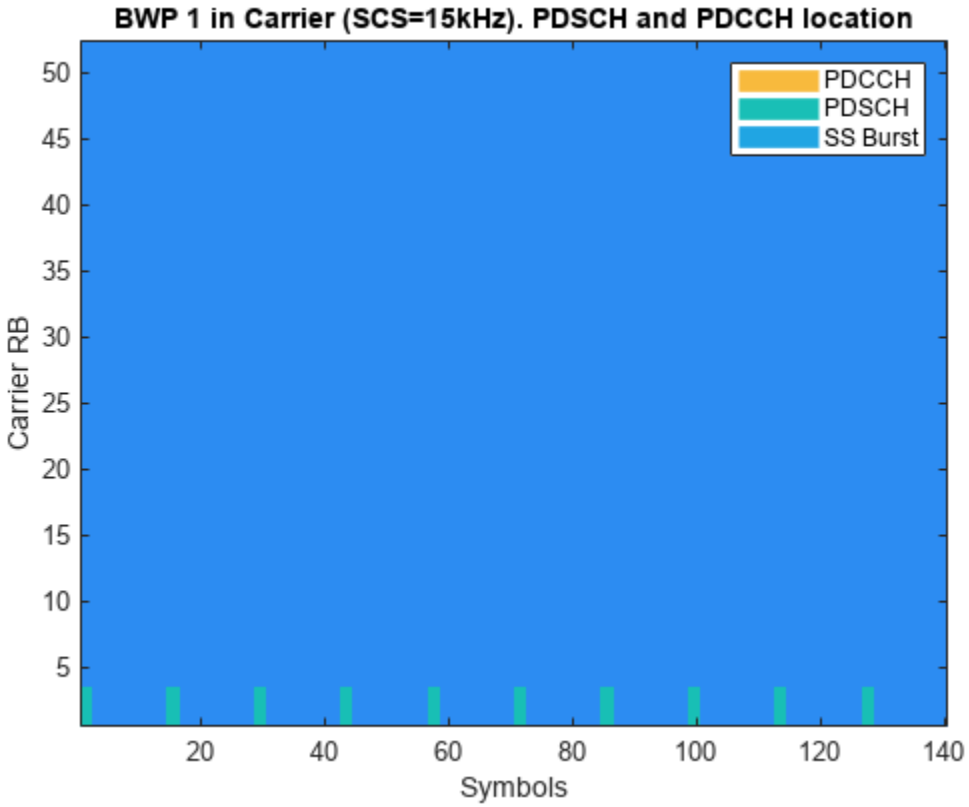
```
% Waveform sample rate (Hz)
samplerate = dlrefwaveinfo.Info.SampleRate
```

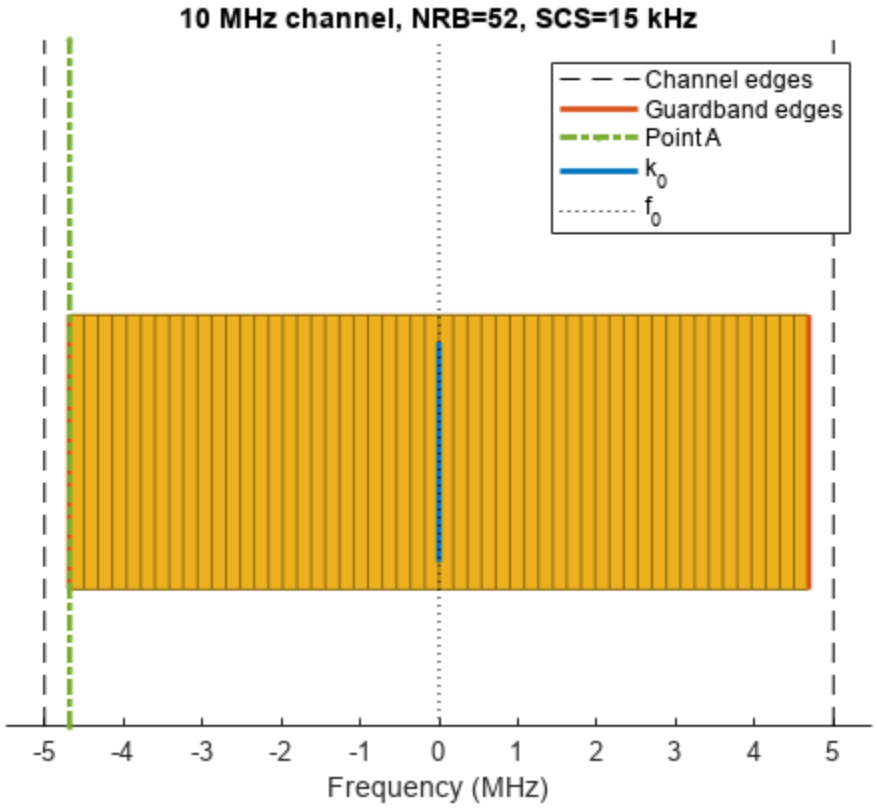
```
samplerate = 15360000
```

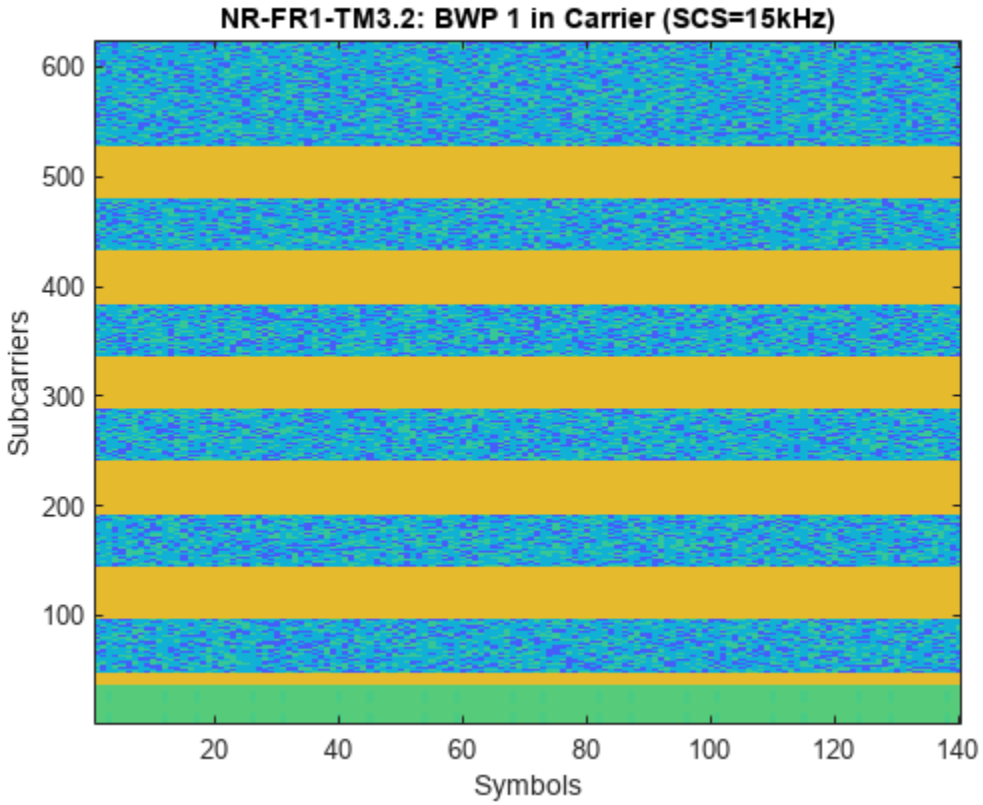
```
plot(abs(dlrefwaveform)); title(sprintf('Magnitude of %s Baseband Waveform',dlnrref)); xlabel('Sa
```




```
% Visualize the associated PRB and subcarrier resource grids
displayResourceGrid(dlrefwavegen);
```


BWP 1 in Carrier (SCS=15kHz). PDSCH and PDCCH location
PDCCH
PDSCH
SS Burst
Carrier RB
Symbols
50
45
40
35
30
25
20
15
10
5
20
40
60
80
100
120
140


10 MHz channel, NRB=52, SCS=15 kHz
Channel edges
Guardband edges
Point A
k0
f0
-5
-4
-3
-2
-1
0
1
2
3
4
5
Frequency (MHz)




```
fullparameterset = dlrefwavegen.Config   % Full low-level parameter set
```

```
fullparameterset =
  nrDLCarrierConfig with properties:

              Label: 'NR-FR1-TM3.2'
      FrequencyRange: 'FR1'
    ChannelBandwidth: 10
             NCellID: 1
        NumSubframes: 10
    WindowingPercent: 0
          SampleRate: []
    CarrierFrequency: 0
         SCSCarriers: {[1x1 nrSCSCarrierConfig]}
      BandwidthParts: {[1x1 nrWavegenBWPConfig]}
             SSBurst: [1x1 nrWavegenSSBurstConfig]
             CORESET: {[1x1 nrCORESETConfig]}
        SearchSpaces: {[1x1 nrSearchSpaceConfig]}
               PDCCH: {[1x1 nrWavegenPDCCHConfig]}
               PDSCH: {[1x1 nrWavegenPDSCHConfig]  [1x1 nrWavegenPDSCHConfig]  [1x1 nrWavegenPDS
               CSIRS: {[1x1 nrWavegenCSIRSConfig]}
```

```
% Make the Config parameters writable and boost the power on all PDSCH DM-RS
dlrefwavegen = makeConfigWritable(dlrefwavegen)
```

```
dlrefwavegen =
  hNRReferenceWaveformGenerator with properties:

    FR1BandwidthTable: [3x15 table]
    FR2BandwidthTable: [4x7 table]
        FR1TestModels: [10x1 string]
        FR2TestModels: [5x1 string]
       FR1DownlinkFRC: [4x1 string]
       FR2DownlinkFRC: [3x1 string]
         FR1UplinkFRC: [89x1 string]
         FR2UplinkFRC: [37x1 string]
               Config: [1x1 nrDLCarrierConfig]
           IsReadOnly: 0
      ConfiguredModel: {["NR-FR1-TM3.2"]  ["10MHz"]  ["15kHz"]  ["FDD"]  [1]  ["17.8.0"]}
           TargetRNTI: 1
```

```
% Set DM-RS power parameter on all the PDSCH
pdscharray = [dlrefwavegen.Config.PDSCH{:}];       % Extract all PDSCH configs into an array
[pdscharray.DMRSPower] = deal(3);                  % Boost the DM-RS power on all the PDSCH
dlrefwavegen.Config.PDSCH = num2cell(pdscharray);  % Reassign the updated PDSCH configs
```

PUSCH FRC Waveform Generation

Each PUSCH FRC reference channel definition in TS 38.104 Annex A explicitly defines a number of key parameters including:

- Frequency range
- Channel bandwidth
- Subcarrier spacing
- Code rate
- Modulation
- DM-RS configuration

Additionally, the associated receiver tests introduce some additional parameters that are not specified in the TS 38.104 Annex A tables, for example, the general test parameters defined in:

- Table 8.2.1.1-1 (Conducted performance requirements for PUSCH without transform precoding)
- Table 8.2.2.1-1 (Conducted performance requirements for PUSCH with transform precoding)
- Table 11.2.2.1.1-1 (Radiated performance requirements for BS type 2-O for PUSCH without transform precoding)
- Table 11.2.2.2.1-1 (Radiated performance requirements for BS type 2-O for PUSCH with transform precoding)

The parameter sets that are captured in the MATLAB reference waveform generator use the above specification sources. Since a given FRC can be used in different tests with different parameters requirements, the following general rules apply to the default generator configurations. All parameters can be modified after construction. Transform precoding is enabled for appropriate FRC. FR2 waveforms are TDD and 20ms in length, and FR1 waveforms are FDD and 10ms. The PUSCH FRC are defined with type A mapping, type B mapping, or, in some cases, either mapping type. In the latter case, type A mapping is configured. FR2 waveforms without transform precoding are configured with PT-RS, otherwise PT-RS are off. Scrambling identities are set to 0. Power levels for all resource elements are uniform. The transport block data source is ITU PN9 with RV = 0 i.e. no retransmissions.

This MATLAB code creates an `hNRReferenceWaveformGenerator` object for the selected PUSCH FRC configuration. Due to the large number of FRC, the live script FRC drop-down lists only those from section TS 38.104 A.1 (reference sensistivity, ACS, in-band blocking etc.) and A.2 (dynamic range). The performance test FRC defined in A.3, A.4, A.5 can be chosen by specifying the FRC name string directly in the code below. After the generator object is created, all configuration parameters can be changed by making them writable using the `makeConfigWritable` function.

```
% Select the PUSCH FRC waveform
ulnrref = G-FR1-A1-1 (SCS...  ;  % This live script down-drop list is preconfigured for TS 38.1

% Possible overrides to Annex A definitions (empty values provide the Annex A defaults)
bw      = [];  % Bandwidth override (5,10,15,20,25,30,40,50,60,70,80,90,100,200,400,800,1600,200
scs     = [];  % Subcarrier spacing override (15,30,60,120,480,960 kHz)
dm      = [];  % Duplexing mode override ("FDD","TDD")
ncellid = [];  % Cell identity override (used to control scrambling identities)

% Run this entire section to generate the required waveform
```

Generate

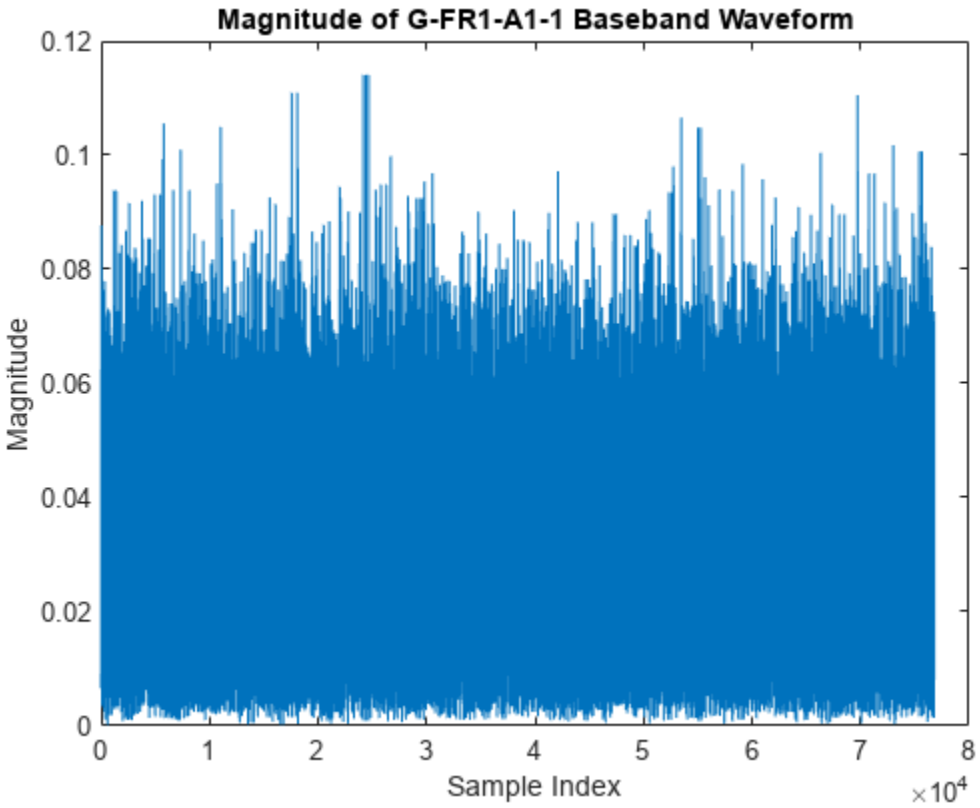
```
% Create generator object for the above PUSCH FRC reference model
ulrefwavegen = hNRReferenceWaveformGenerator(ulnrref,bw,scs,dm,ncellid)

ulrefwavegen =
  hNRReferenceWaveformGenerator with properties:

    FR1BandwidthTable: [3x15 table]
    FR2BandwidthTable: [4x7 table]
        FR1TestModels: [10x1 string]
        FR2TestModels: [5x1 string]
       FR1DownlinkFRC: [4x1 string]
       FR2DownlinkFRC: [3x1 string]
         FR1UplinkFRC: [89x1 string]
         FR2UplinkFRC: [37x1 string]
               Config: [1x1 nrULCarrierConfig]
           IsReadOnly: 1
      ConfiguredModel: {["G-FR1-A1-1"]  []  []  ["FDD"]  [0]}
           TargetRNTI: 0



% Generate waveform
[ulrefwaveform,ulrefwaveinfo,ulresourceinfo] = generateWaveform(ulrefwavegen);

% View transmission information about the set of PUSCH within the waveform
ulresourceinfo.WaveformResources.PUSCH

ans = struct with fields:
          Name: 'PUSCH sequence for G-FR1-A1-1'
    CDMLengths: [1 1]
     Resources: [1x10 struct]


% View detailed information about one of the PUSCH sequences
ulresourceinfo.WaveformResources.PUSCH(1).Resources

ans=1×10 struct array with fields:
    NSlot
```

```
    TransportBlockSize
    TransportBlock
    RV
    Codeword
    G
    Gd
    ChannelIndices
    ChannelSymbols
    DMRSIndices
    DMRSSymbols
    DMRSSymbolSet
    PTRSIndices
    PTRSSymbols
    PTRSSymbolSet
```

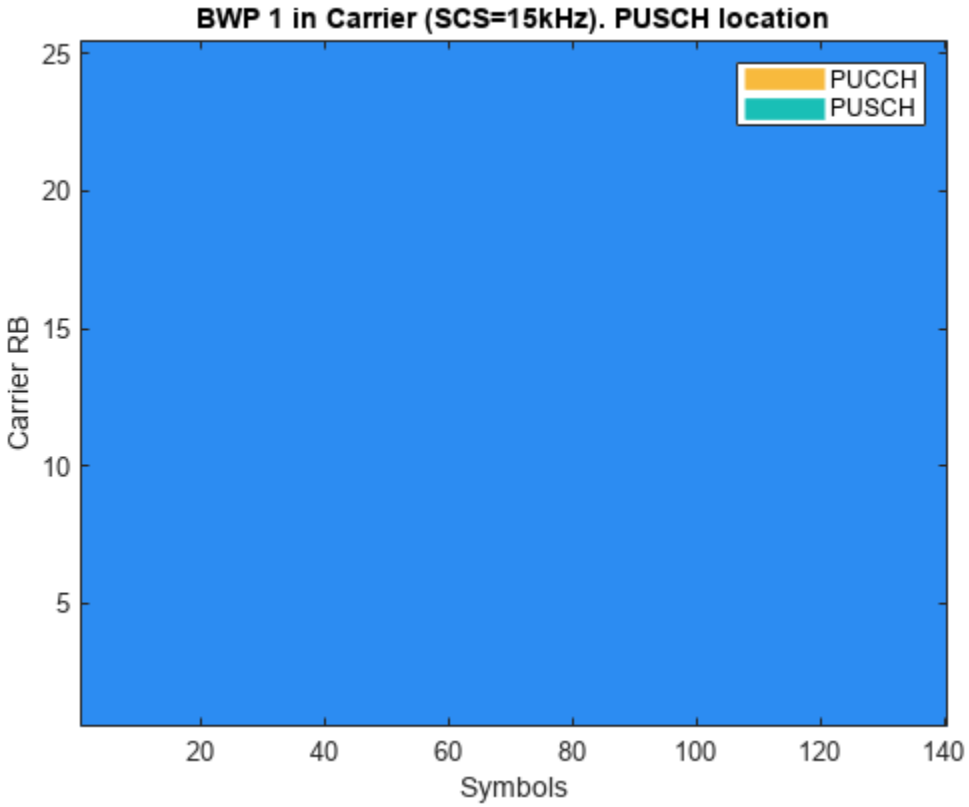
```
% Waveform sample rate (Hz)
samplerate = ulrefwaveinfo.Info.SampleRate
```

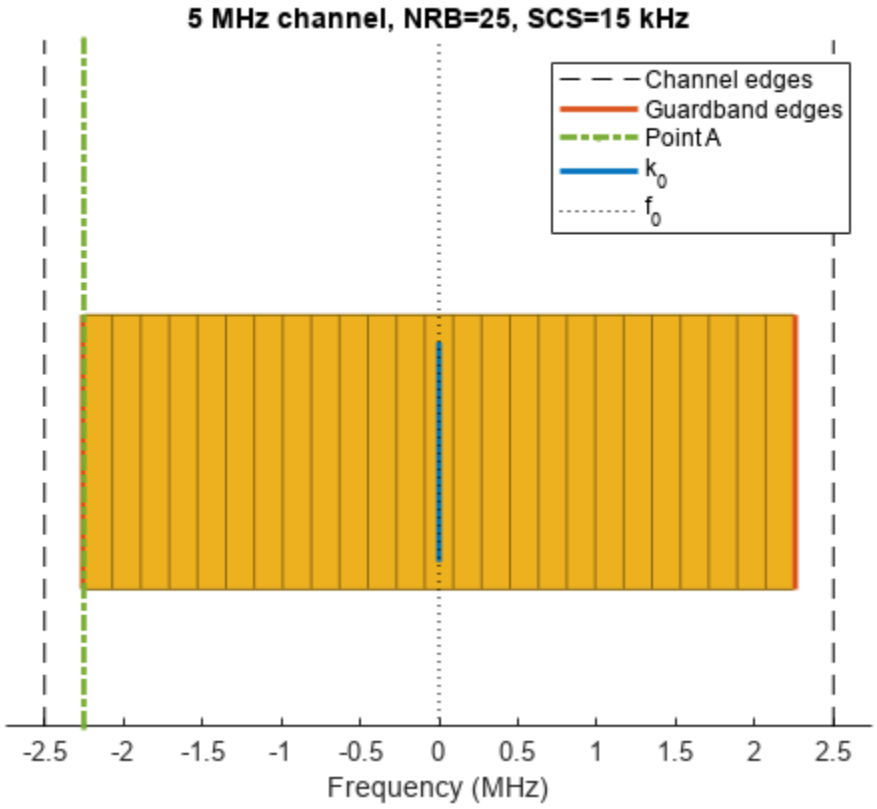
```
samplerate = 7680000
```

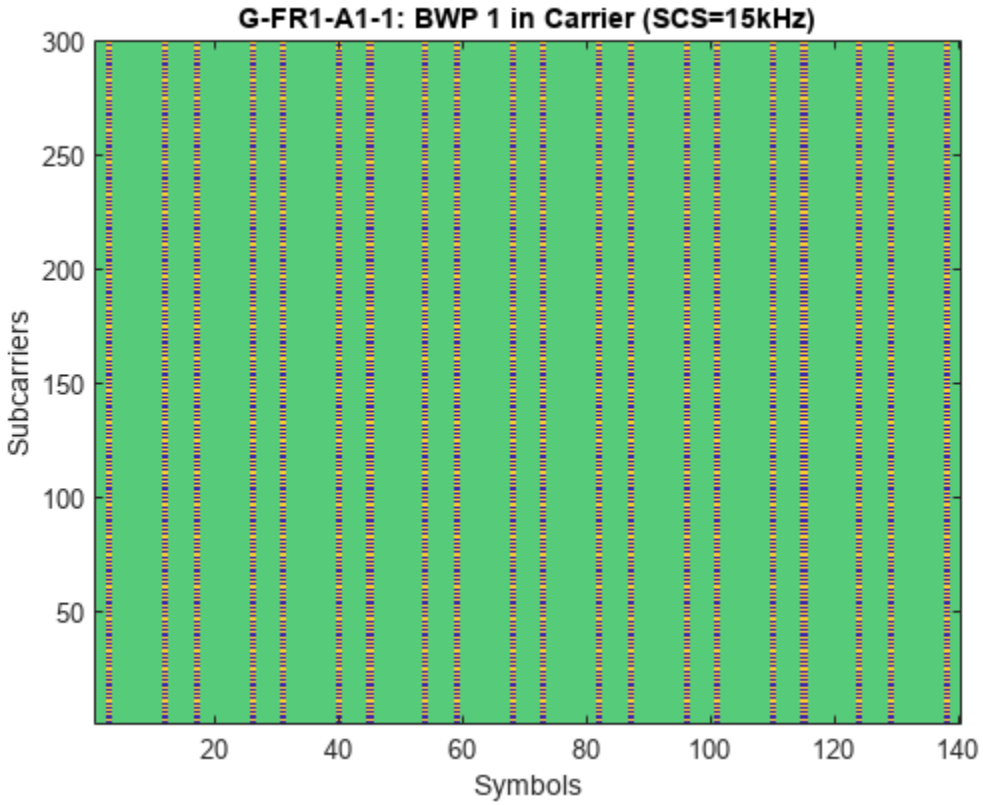
```
plot(abs(ulrefwaveform)); title(sprintf('Magnitude of %s Baseband Waveform',ulnrref)); xlabel('Sa
```




```
% Visualize the associated PRB and subcarrier resource grids
displayResourceGrid(ulrefwavegen);
```


BWP 1 in Carrier (SCS=15kHz). PUSCH location
PUCCH
PUSCH
Carrier RB
25
20
15
10
5
20
40
60
80
100
120
140
Symbols


5 MHz channel, NRB=25, SCS=15 kHz
Channel edges
Guardband edges
Point A
k_0
f_0
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
Frequency (MHz)




```
fullparameterset = ulrefwavegen.Config   % Full low-level parameter set
```

```
fullparameterset =
  nrULCarrierConfig with properties:

              Label: 'G-FR1-A1-1'
      FrequencyRange: 'FR1'
    ChannelBandwidth: 5
             NCellID: 0
        NumSubframes: 10
    WindowingPercent: 0
          SampleRate: []
    CarrierFrequency: 0
         SCSCarriers: {[1x1 nrSCSCarrierConfig]}
      BandwidthParts: {[1x1 nrWavegenBWPConfig]}
               PUSCH: {[1x1 nrWavegenPUSCHConfig]}
               PUCCH: {[1x1 nrWavegenPUCCH0Config]}
                 SRS: {[1x1 nrWavegenSRSConfig]}
```

References

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

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

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

[4] 3GPP TS 38.141-1. “NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[5] 3GPP TS 38.141-2. “NR; Base Station (BS) conformance testing Part 2: Radiated conformance testing.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Apps
5G Waveform Generator

Related Examples

- “5G NR Downlink Vector Waveform Generation” on page 1-2
- “5G NR Uplink Vector Waveform Generation” on page 2-2
- “App-Based 5G Waveform Generation” on page 7-29

App-Based 5G Waveform Generation

This example shows how to generate standard-compliant NR uplink and downlink carrier waveforms, NR test models (NR-TM), and NR uplink and downlink fixed reference channel (FRC) waveforms by using the **5G Waveform Generator** app. The example also discusses waveform exporting and transferring options available in the app.

Open 5G Waveform Generator App

On the **Apps** tab of the MATLAB® toolstrip, under **Signal Processing and Communications**, click the **5G Waveform Generator** app icon. This app opens the **Wireless Waveform Generator** app configured for 5G waveform generation.

Select 5G NR Waveform

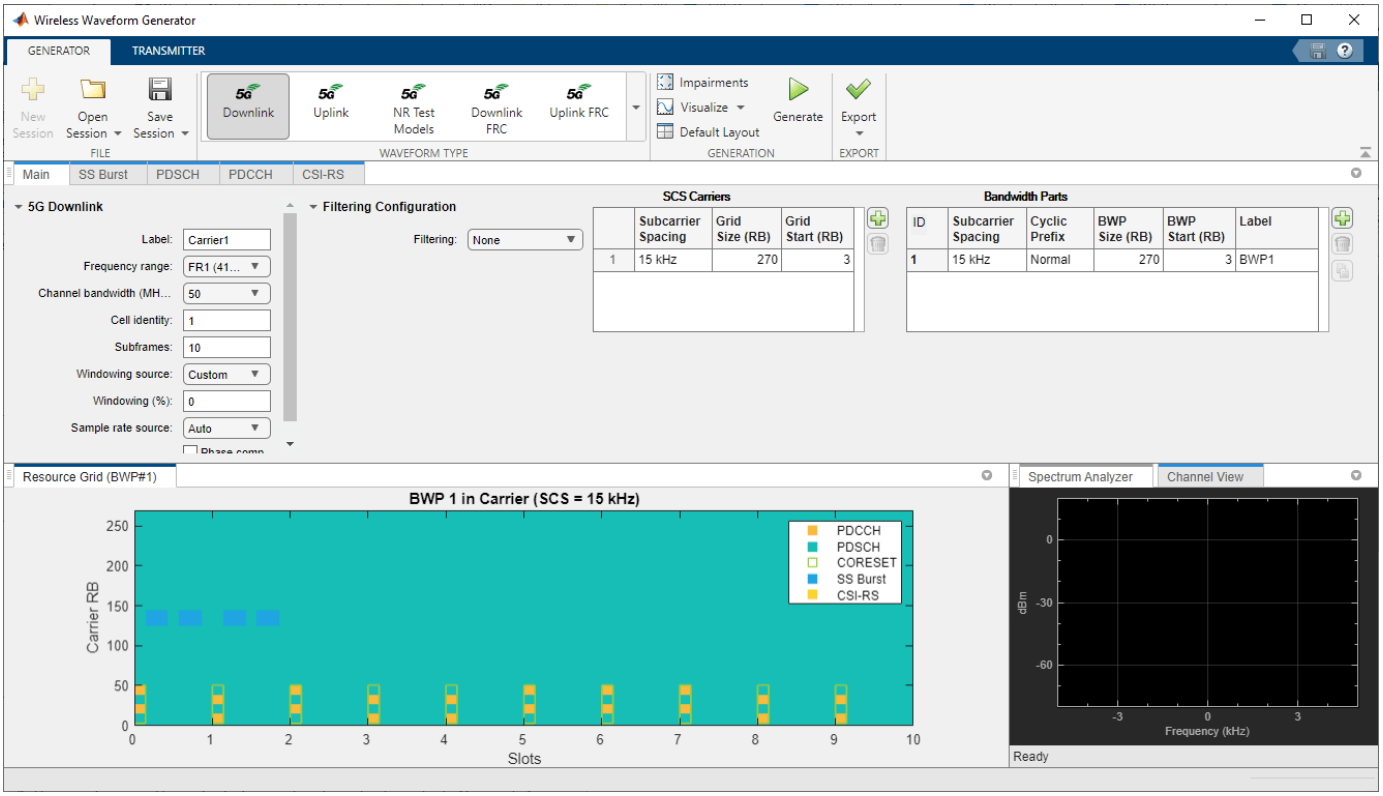
In the **Waveform Type** section on the app toolstrip, click the waveform you want to generate. Select one of these waveforms.

- `5G Downlink`
- `5G Uplink`
- `5G Test Models`
- `5G Downlink FRC`
- `5G Uplink FRC`

Generate 5G NR Waveform

Depending on the selected waveform, the app presents specific tabs where you can set the parameters of the selected waveform. On the app toolstrip, in the **Generation** section, you can add impairments and set visualization tools applicable for the selected waveform. To visualize the waveform on the selected visualization tools, click **Generate**.

For example, this figure shows the visualization results of a 5G NR downlink waveform using default parameters.



Export Generated Waveform to MATLAB Workspace or File

To export the generated waveform, on the app toolstrip, in the **Export** section, select **Export to Workspace** or **Export to File**. You can export the waveform as a structure to the MATLAB workspace or a MAT-file (`.mat`). You can also export the waveform to a baseband file (`.bb`).

Export Waveform Configuration Parameters to MATLAB Script

To export waveform configuration parameters as a MATLAB script, on the app toolstrip, in the **Export** section, select **Export to MATLAB Script**. You can run the exported MATLAB script to generate the waveform without the app.

Export Waveform Configuration Parameters to Simulink

To export waveform configuration parameters as a Simulink block, on the app toolstrip, in the **Export** section, select **Export to Simulink**. You can use the exported block to generate the waveform in a Simulink model without the app.

Transmit 5G NR Waveform

To transmit the generated waveform using a connected radio or lab test instrument, on the app toolstrip, click on the **Transmitter** tab.

- To transmit your waveforms over the air at full radio device rates, use the Wireless Testbench™ software and connect a supported radio to your computer. For a list of radios that support full device rates, see “Supported Radio Devices” (Wireless Testbench). This feature requires “Wireless Testbench”.

- To transmit a waveform by using an SDR, connect one of the supported SDRs (ADALM-Pluto, USRP™, USRP embedded series, and Xilinx® Zynq-based radios) to your computer and have the associated add-on installed. For more information, see “Transmit Using SDR”.
- To transmit a waveform by using a lab test instrument, connect one of the instruments supported by the `rfsiggen` (Instrument Control Toolbox) function to your computer. For more information, see “Quick-Control RF Signal Generator Requirements” (Instrument Control Toolbox). This feature requires “Instrument Control Toolbox”.

See Also

Apps
5G Waveform Generator

Related Examples

- “5G NR-TM and FRC Waveform Generation” on page 7-12

More About

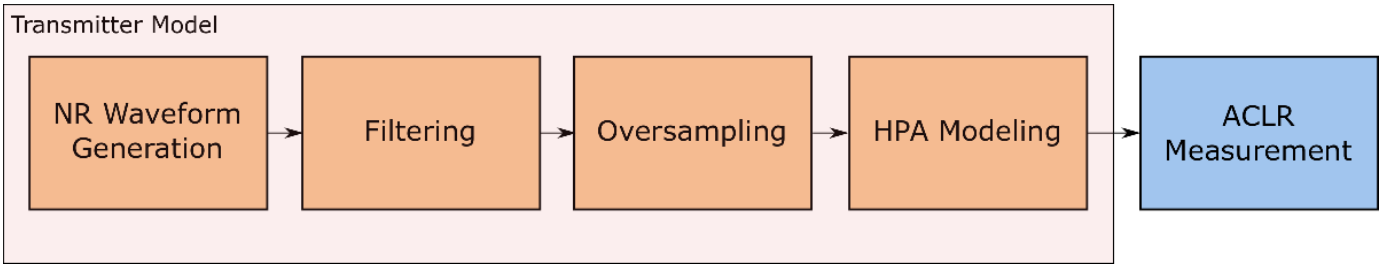
- “Create Waveforms Using Wireless Waveform Generator App”

5G NR Downlink ACLR Measurement

This example shows how to measure the adjacent channel leakage ratio (ACLR) for 5G NR test models (NR-TMs) in frequency range 1 (FR1) and FR2 using 5G Toolbox™.

Introduction

The ACLR is the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency. This example performs ACLR measurements for an NR downlink waveform, as defined in TS 38.104 Section 6.6.3. To model the effect of out-of-band spectral emissions, the example applies spectral regrowth on an oversampled waveform by using a high power amplifier (HPA) model.



Generate NR-TM Waveform

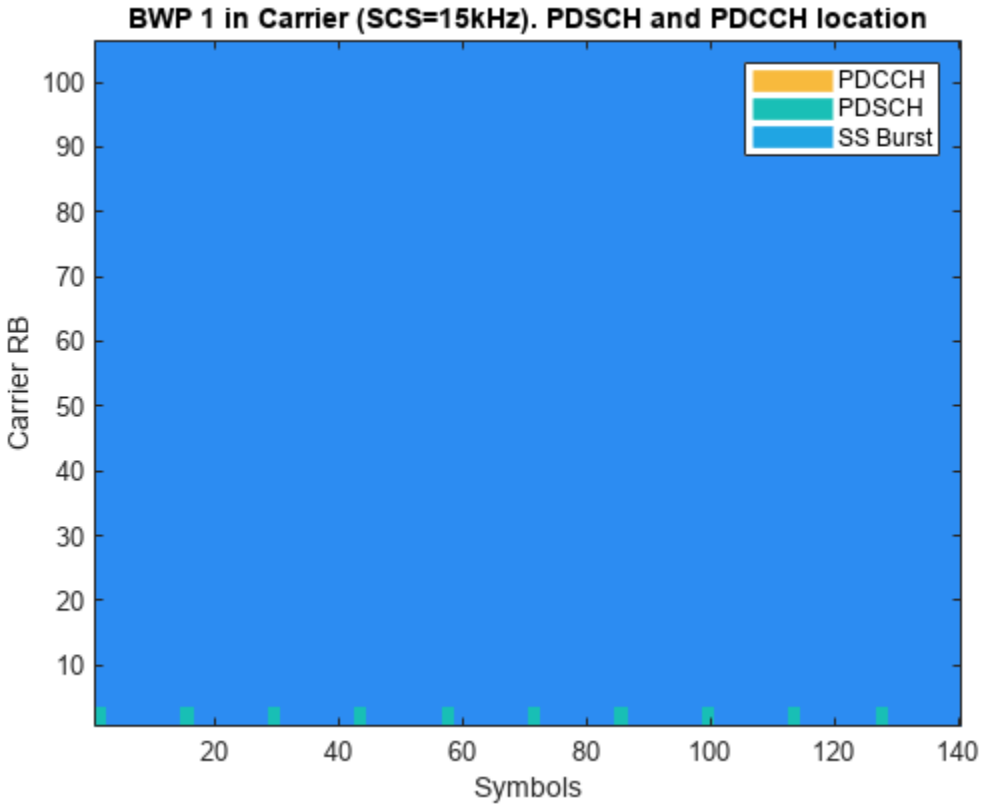
Use the MATLAB class hNRReferenceWaveformGenerator to generate 5G NR-TMs for FR1 and FR2. You can generate the NR-TM waveforms by specifying these parameters:

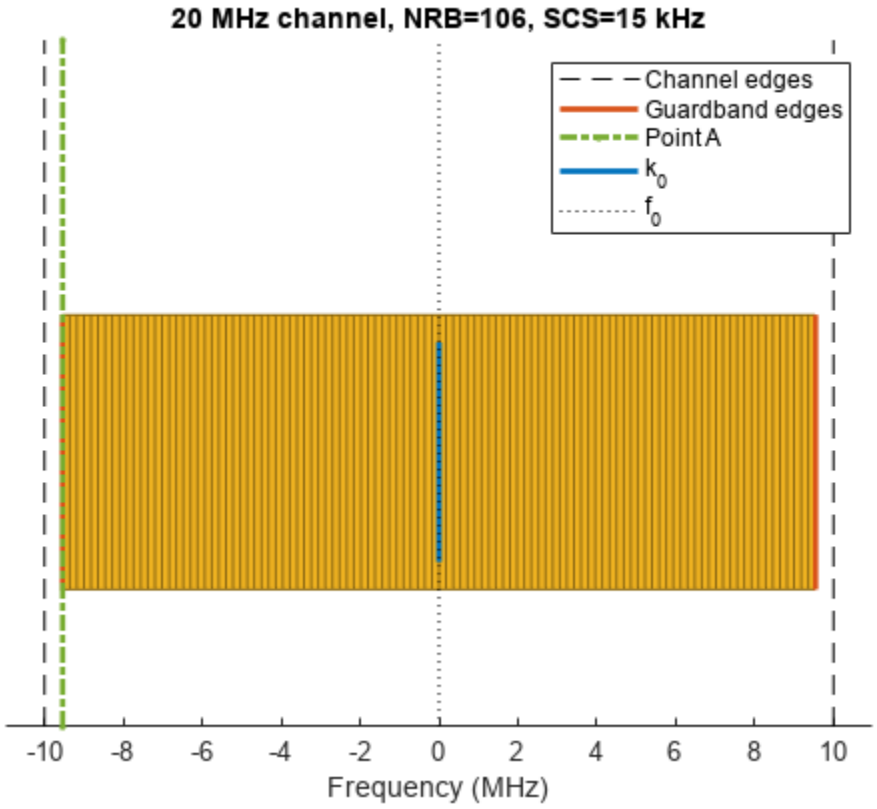
- NR-TM name
- Channel bandwidth
- Subcarrier spacing
- Duplexing mode

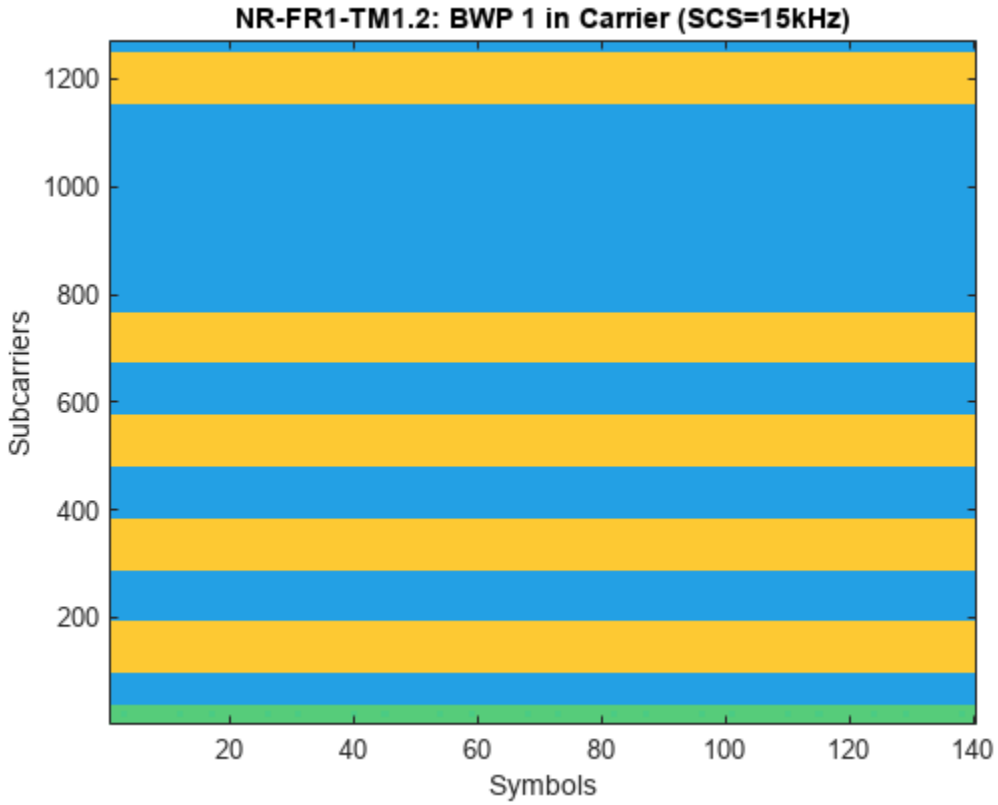
For more information, see the “5G NR-TM and FRC Waveform Generation” on page 7-12 example.

```
% Select the NR-TM waveform parameters
nrtm = NR-FR1-TM1.2 (...  ;  % NR-TM name and properties
bw   = 20MHz (FR1)        ;  % Channel bandwidth
scs  = 15kHz (FR1)        ;  % Subcarrier spacing
dm   = FDD                ;  % Duplexing mode

% Create generator object for the above NR-TM
tmWaveGen = hNRReferenceWaveformGenerator(nrtm,bw,scs,dm);

% Ensure no windowing to highlight impact of filtering on ACLR
tmWaveGen = makeConfigWritable(tmWaveGen);
tmWaveGen.Config.WindowingPercent = 0;

% Generate waveform
[tmWaveform,tmWaveInfo] = generateWaveform(tmWaveGen);
```

```
samplingRate = tmWaveInfo.Info.SamplingRate;   % Waveform sampling rate (Hz)

% Visualize the associated PRB and subcarrier resource grids
displayResourceGrid(tmWaveGen);
```




20 MHz channel, NRB=106, SCS=15 kHz
Channel edges
Guardband edges
Point A
k_0
f_0
-10
-8
-6
-4
-2
0
2
4
6
8
10
Frequency (MHz)



Display Results

- The associated PRB resource grid (top) depicts the allocation of the different components (PDCCH, PDSCH, CORESET and SS Burst) in each BWP. The grid does not plot the amplitude of the signals only the signal locations in the grid.
- The SCS specific carrier resource grids (middle) along with the minimum guardbands, aligned relative to the overall channel bandwidth.
- The subcarrier resource grid (bottom) indicates the amplitude levels of the generated waveform. If just one color is shown, all the components have the same amplitude.

Calculate ACLR Parameters

The helper function hACLRParametersNR.m calculates the parameters required for ACLR measurement. It returns a structure with these fields:

- `Bandwidth`: the channel bandwidth associated with `tmWaveform`, in Hz. This is the overall bandwidth of the assigned channel.
- `MeasurementBandwidth`: the ACLR measurement bandwidth in Hz.
- `AdjacentChannelOffset`: a vector of NR center frequencies, in Hz, for adjacent channels.
- `OSR`: the integer oversampling ratio of the input `tmWaveform` required to create a signal capable of representing 1st and 2nd adjacent channels.
- `SamplingRate`: the sampling rate of the oversampled signal used to measure ACLR. If `OSR = 1`, this signal is the input waveform; if `OSR > 1`, this signal is the input waveform upsampled by `OSR`.

Therefore: `aclrParameters.SamplingRate = OSR*samplingRate` (input waveform sampling rate).

```
aclrParameters = hACLRParametersNR(tmWaveGen.Config);
disp(aclrParameters);
```

```
                Bandwidth: 20000000
     MeasurementBandwidth: 19080000
    AdjacentChannelOffset: [-40000000 -20000000 20000000 40000000]
                      OSR: 4
             SamplingRate: 122880000
```

Filter Waveform to Improve ACLR

The generated waveform has no filtering, so there are significant out-of-band spectral emissions owing to the implicit rectangular pulse shaping in the OFDM modulation (each OFDM subcarrier has a sinc shape in the frequency domain). Filtering the waveform improves ACLR performance.

Design a filter with a transition band that starts at the edge of the occupied transmission bandwidth (`aclrParameters.MeasurementBandwidth`) and stops at the edge of the overall channel bandwidth (`aclrParameters.Bandwidth`). This filter involves no rate change, it just shapes the spectrum within the original bandwidth of the waveform.

```
% Design filter
lpFilt = designfilt('lowpassfir',...
    'PassbandFrequency',aclrParameters.MeasurementBandwidth/2,...
    'StopbandFrequency',aclrParameters.Bandwidth/2,...
    'PassbandRipple',0.1,...
    'StopbandAttenuation',80,...
    'SampleRate',samplingRate);

% Apply filter
filtTmWaveform = filter(lpFilt,tmWaveform);
```

Oversampling and HPA Nonlinearity Model

To create a signal capable of representing 1st and 2nd adjacent carriers, for example, to represent `aclrParameters.Bandwidth` with at most 85% bandwidth occupancy, oversample the NR waveform. After oversampling the signal, employ an HPA model to generate out-of-band distortion. For example, to simulate the HPA behaviour, you can use the Rapp method, which is widely used in wireless applications to generate AM/AM distortion. In MATLAB®, you can use the Memoryless Nonlinearity object to model the Rapp method. To highlight the impact of filtering on the ACLR measurements, apply the oversampling and HPA nonlinearities first to the filtered NR signal and then to the same NR signal without filtering.

```
% Apply required oversampling
resampled = resample(tmWaveform,aclrParameters.OSR,1);           % Not filtered
filtResampled = resample(filtTmWaveform,aclrParameters.OSR,1);   % Filtered

% Create and configure a memoryless nonlinearity to model the amplifier
nonLinearity = comm.MemorylessNonlinearity;
nonLinearity.Method = 'Rapp model';
nonLinearity.Smoothness = 3;              % p parameter
nonLinearity.LinearGain = 0.5;            % dB
nonLinearity.OutputSaturationLevel = 2;   % It limits the output signal level

% Signal conditioning to control the HPA input back-off level
```

```
resampled = resampled/max(abs(resampled));                % Not filtered
filtResampled = filtResampled/max(abs(filtResampled));    % Filtered

% Apply the amplifier model to the NR waveforms
txWaveform = nonLinearity(resampled);           % Not filtered
txFiltWaveform = nonLinearity(filtResampled);   % Filtered
```

Calculate NR ACLR

The hACLRMeasurementNR.m helper function measures the NR ACLR using a square window on adjacent channels. It also measures the power (in dBm) of the signal in the main channel.

```
% Calculate NR ACLR
aclr = hACLRMeasurementNR(aclrParameters,txWaveform);               % Not filtered
filtAclr = hACLRMeasurementNR(aclrParameters,txFiltWaveform);   % Filtered
```

Calculate Error Vector Magnitude

The hNRPDSCHEVM.m helper function measures the error vector magnitude (EVM) of NR-TM or fixed reference channel (FRC) waveforms. The function calculates the root mean square (RMS) and peak EVMs per OFDM symbol, slot, subcarrier, and overall EVM.

```
% EVM configuration parameters
evmCfg.PlotEVM = false;
evmCfg.SampleRate = aclrParameters.SamplingRate;
evmCfg.Label = tmWaveGen.ConfiguredModel{1};

% Measure the EVM related statistics for the transmitted waveform without filtering
evmInfo = hNRDownlinkEVM(tmWaveGen.Config,txWaveform,evmCfg)
```

```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 0.009 0.023%
PDSCH RMS EVM, Peak EVM, slot 1: 0.008 0.029%
PDSCH RMS EVM, Peak EVM, slot 2: 0.008 0.024%
PDSCH RMS EVM, Peak EVM, slot 3: 0.010 0.023%
PDSCH RMS EVM, Peak EVM, slot 4: 0.009 0.021%
PDSCH RMS EVM, Peak EVM, slot 5: 0.009 0.025%
PDSCH RMS EVM, Peak EVM, slot 6: 0.009 0.022%
PDSCH RMS EVM, Peak EVM, slot 7: 0.009 0.026%
PDSCH RMS EVM, Peak EVM, slot 8: 0.008 0.020%
PDSCH RMS EVM, Peak EVM, slot 9: 0.009 0.023%
PDCCH RMS EVM, Peak EVM, slot 0: 0.012 0.027%
PDCCH RMS EVM, Peak EVM, slot 1: 0.008 0.017%
PDCCH RMS EVM, Peak EVM, slot 2: 0.010 0.024%
PDCCH RMS EVM, Peak EVM, slot 3: 0.010 0.022%
PDCCH RMS EVM, Peak EVM, slot 4: 0.007 0.019%
PDCCH RMS EVM, Peak EVM, slot 5: 0.011 0.028%
PDCCH RMS EVM, Peak EVM, slot 6: 0.012 0.026%
PDCCH RMS EVM, Peak EVM, slot 7: 0.013 0.029%
PDCCH RMS EVM, Peak EVM, slot 8: 0.012 0.022%
PDCCH RMS EVM, Peak EVM, slot 9: 0.010 0.018%
Averaged RMS EVM frame 0: 0.009%
Averaged overall PDSCH RMS EVM: 0.009%
Overall PDSCH Peak EVM = 0.029234%
Averaged overall PDCCH RMS EVM: 0.011%
Overall PDCCH Peak EVM = 0.028711%

evmInfo = struct with fields:
    PDSCH: [1x1 struct]
```

```
    PDCCH: [1x1 struct]

% Measure the EVM related statistics for the transmitted waveform with filtering
evmInfoFilt = hNRDownlinkEVM(tmWaveGen.Config,txFiltWaveform,evmCfg);

EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 0.257 0.757%
PDSCH RMS EVM, Peak EVM, slot 1: 0.247 0.751%
PDSCH RMS EVM, Peak EVM, slot 2: 0.245 0.816%
PDSCH RMS EVM, Peak EVM, slot 3: 0.249 0.807%
PDSCH RMS EVM, Peak EVM, slot 4: 0.264 0.733%
PDSCH RMS EVM, Peak EVM, slot 5: 0.281 0.899%
PDSCH RMS EVM, Peak EVM, slot 6: 0.254 0.778%
PDSCH RMS EVM, Peak EVM, slot 7: 0.237 0.654%
PDSCH RMS EVM, Peak EVM, slot 8: 0.238 0.797%
PDCCH RMS EVM, Peak EVM, slot 0: 0.403 1.310%
PDCCH RMS EVM, Peak EVM, slot 1: 0.310 0.850%
PDCCH RMS EVM, Peak EVM, slot 2: 0.271 0.858%
PDCCH RMS EVM, Peak EVM, slot 3: 0.261 0.874%
PDCCH RMS EVM, Peak EVM, slot 4: 0.227 0.911%
PDCCH RMS EVM, Peak EVM, slot 5: 0.311 0.936%
PDCCH RMS EVM, Peak EVM, slot 6: 0.329 0.982%
PDCCH RMS EVM, Peak EVM, slot 7: 0.301 1.159%
PDCCH RMS EVM, Peak EVM, slot 8: 0.280 1.039%
Averaged overall PDSCH RMS EVM: 0.253%
Overall PDSCH Peak EVM = 0.89869%
Averaged overall PDCCH RMS EVM: 0.303%
Overall PDCCH Peak EVM = 1.3105%
```

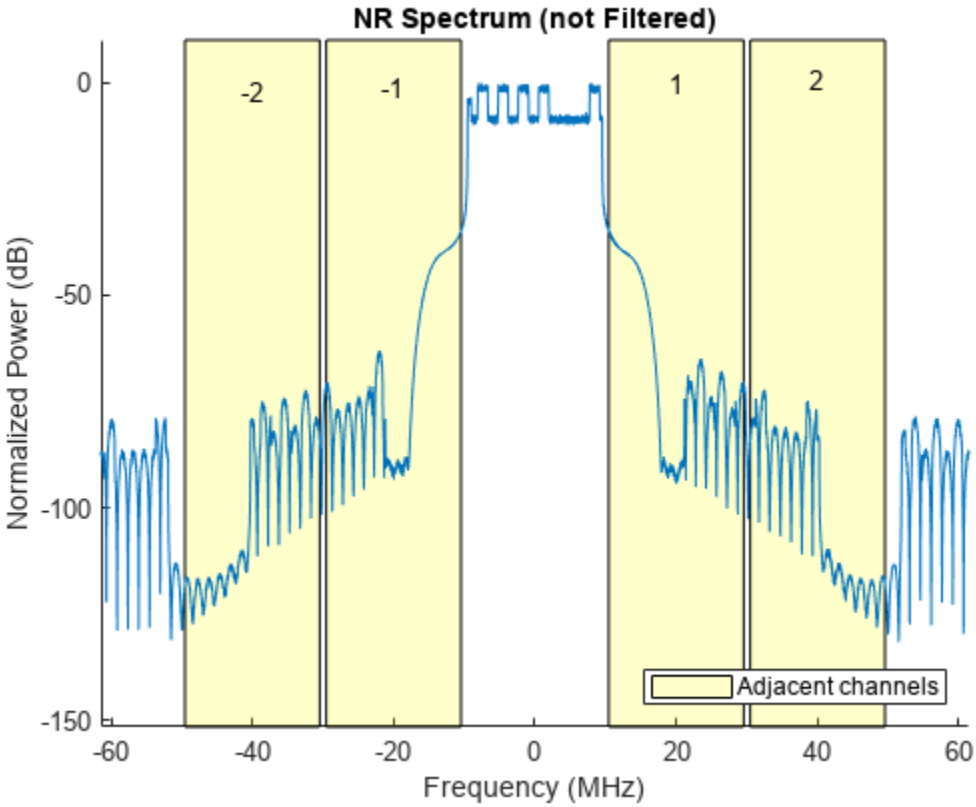
Display Results

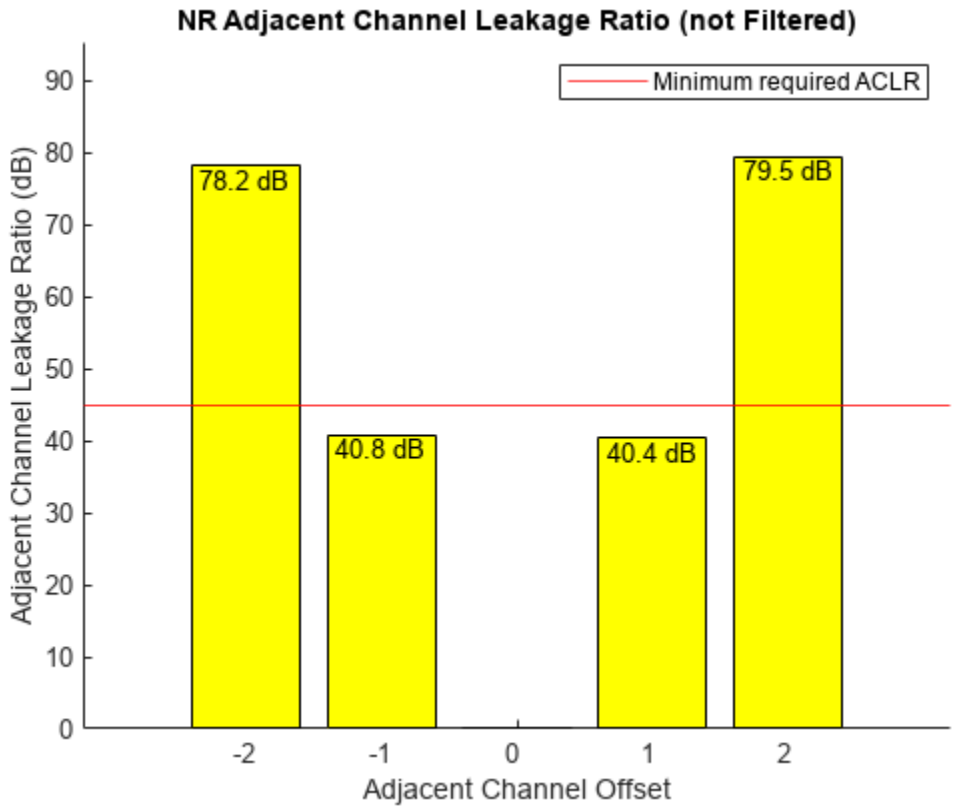
Display the spectrum and the adjacent channel leakage ratios.

Not Filtered

According to TS 38.104 Section 6.6.3.2, the minimum required ACLR for conducted measurements is 45 dB. As some of these ACLR values are lower than 45 dB, they do not fall within the requirements.

```
hACLRResultsNR(aclr,aclrParameters,txWaveform,'(not Filtered)');
```

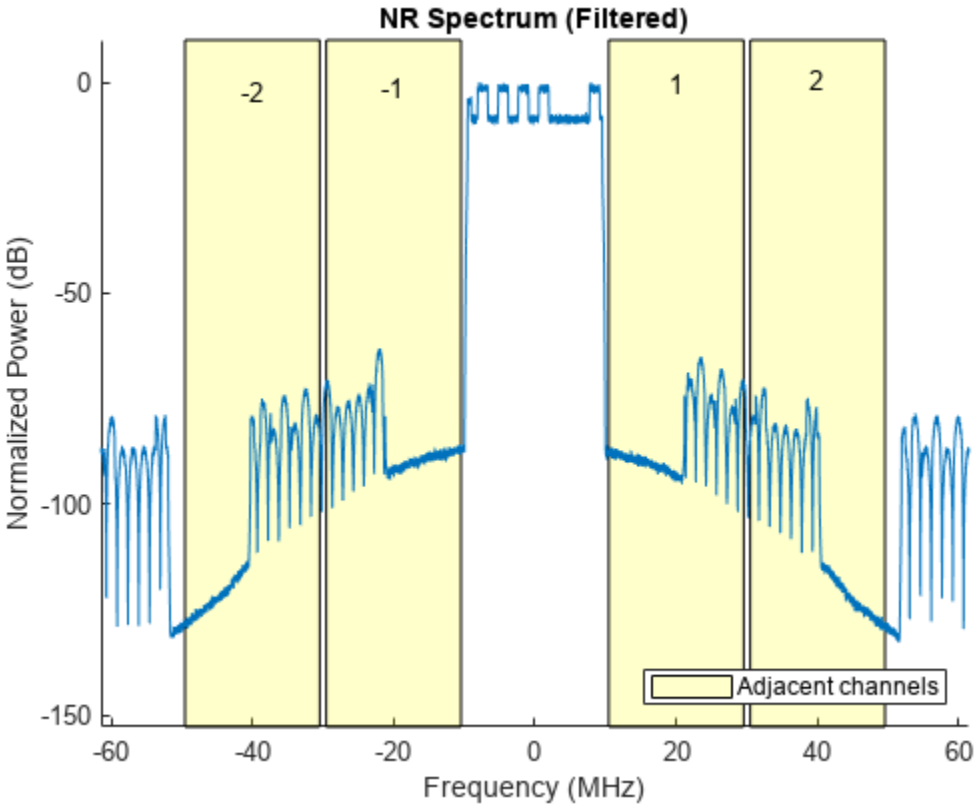

NR Spectrum (not Filtered)
-2
-1
1
2
0
-50
-100
-150
Normalized Power (dB)
-60
-40
-20
0
20
40
60
Frequency (MHz)
Adjacent channels

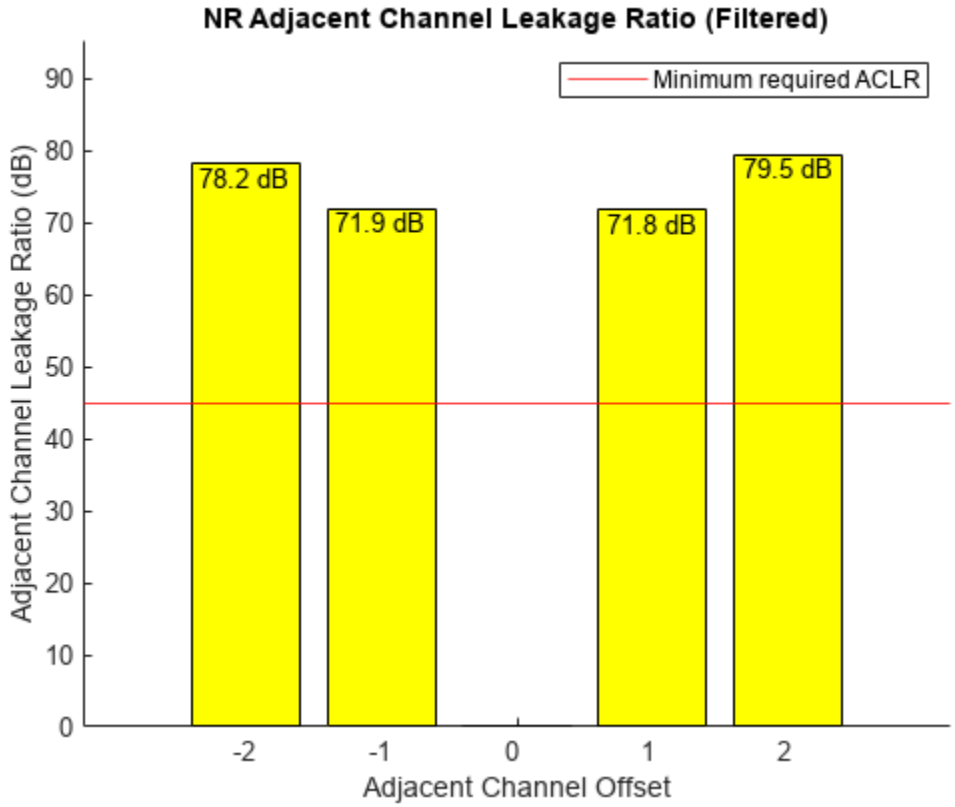


Filtered

The performance improves when the generated waveform is filtered. The ACLR results with the filtered waveform are higher than the minimum required value.

```
hACLRResultsNR(filtAclr,aclrParameters,txFiltWaveform,'(Filtered)');
```


NR Spectrum (Filtered)
-2
-1
1
2
0
-50
-100
-150
Normalized Power (dB)
-60
-40
-20
0
20
40
60
Frequency (MHz)
Adjacent channels



Local Functions

```
function hACLRResultsNR(aclr, aclrParameters, waveform, arg)

    minACLR = 45;

    % 4th input argument is plot title qualifier
    if nargin > 3
        titleText = [' ' arg];
    elseif nargin > 1
        titleText = [];
    else
        titleText = [];
        waveform = [];
    end

    % ACLR values and ticks for bar chart
    values = round([aclr(1:end/2) 0 aclr(end/2+1:end)],1);
    tick = 1:numel(values);
    ticklabel = tick-ceil(numel(tick)/2);
    labelvec = tick;
    labelvec(ceil(end/2)) = []; % Do not plot label for 0dB ACLR on channel

    % Plot NR Spectrum
    if ~isempty(waveform)
        figure;
        [spectrum, frequency] = pwelch(waveform, kaiser(8192*4,19), [], [], ...
```

```
            aclrParameters.SamplingRate, 'centered', 'power');
        frequency = frequency * 10^(-6); % MHz
        spectrum = 10*log10(spectrum / max(spectrum));
        adjacentChannelLabel = [ticklabel(1:floor(length(ticklabel)/2)) ...
        ticklabel(floor(length(ticklabel)/2)+2:end)];
        % Select 'x' and 'y' limits to show the adjacent channels in the plot
        xLimitRight = aclrParameters.AdjacentChannelOffset + (aclrParameters.MeasurementBandwidt
        xLimitRight = xLimitRight * 10^(-6); % MHz
        xLimitLeft = aclrParameters.AdjacentChannelOffset - (aclrParameters.MeasurementBandwidth/
        xLimitLeft = xLimitLeft * 10^(-6); % MHz
        yLimits = [min(spectrum)-20 max(spectrum)+10];
        ylim(yLimits);
        xlim([min(frequency) max(frequency)])
        hold on;
        for i = 1:length(aclrParameters.AdjacentChannelOffset)
            patch('XData',[xLimitRight(i) xLimitRight(i) xLimitLeft(i) ...
                xLimitLeft(i)],'YData', [yLimits fliplr(yLimits)], ...
                'FaceColor','y','FaceAlpha',0.2) % Plot adjacent channels
            text(aclrParameters.AdjacentChannelOffset(i)*10^(-6), i, sprintf('%d', ...
                adjacentChannelLabel(i)), 'HorizontalAlignment', 'Center', ...
                'VerticalAlignment', 'Top'); % Plot adjacent channel labels
        end
        plot(frequency, spectrum);
        hold off;
        xlabel('Frequency (MHz)');
        ylabel('Normalized Power (dB)');
        title(strcat ('NR Spectrum', titleText));
        legend('Adjacent channels', 'Location', 'SouthEast')
    end

    % Plot NR ACLR
    figure;
    hold on;
    yline(minACLR,'r');
    bar(values, 'BaseValue', 0, 'FaceColor', 'yellow');
    hold off;
    set(gca, 'XTick', tick, 'XTickLabel', ticklabel, 'YLim', ...
        [0 0.2*max(values)+max(values)]);
    for i = labelvec
        text(i, values(i), sprintf('%0.1f dB',values(i)), ...
            'HorizontalAlignment', 'Center', 'VerticalAlignment', 'Top');
    end
    title(strcat('NR Adjacent Channel Leakage Ratio', titleText));
    xlabel('Adjacent Channel Offset');
    ylabel('Adjacent Channel Leakage Ratio (dB)');
    legend('Minimum required ACLR');

end
```

References

[1] 3GPP TS 38.104. “NR; Base Station (BS) radio transmission and reception.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “5G NR-TM and FRC Waveform Generation” on page 7-12

Modeling and Testing an NR RF Transmitter

The example shows how to characterize the impact of RF impairments, such as in-phase and quadrature (IQ) imbalance, phase noise, and power amplifier (PA) nonlinearities on the performance of a new radio (NR) radio frequency (RF) transmitter. The NR RF transmitter is modeled in Simulink® using 5G Toolbox™ and RF Blockset™.

Introduction

This example shows how to characterize the impact of RF impairments such as IQ imbalance, phase noise, and PA nonlinearities on the performance of an NR RF transmitter. To evaluate the performance, the example considers these measurements:

- Error vector magnitude (EVM): vector difference at a given time between the ideal (transmitted) signal and the measured (received) signal. Annex B and Annex C of TS 38.104 define an alternative method for computing the EVM in FR1 and FR2, respectively.
- Adjacent channel leakage ratio (ACLR): measure of the amount of power leaking into adjacent channels and is defined as the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency.
- Occupied bandwidth: bandwidth that contains 99% of the total integrated power of the signal, centered on the assigned channel frequency.
- Channel power: filtered mean power centered on the assigned channel frequency.
- Complementary cumulative distribution function (CCDF): probability of a signal's instantaneous power to be a level specified above its average power.

The example works on a subframe by subframe basis and uses a Simulink model to perform these steps:

1. Generate the baseband waveform using 5G Toolbox features.
2. Oversample and filter the waveform by using an FIR Interpolation block.
3. Import the baseband waveform into the RF Transmitter Subsystem block implemented by using RF Blockset blocks. The model uses an RF intermediate frequency to carry the baseband information in RF Blockset.
4. Model the effects of upconverting the waveform to the carrier frequency by using the RF Transmitter Subsystem block. This block models the impairments introduced by an RF transmitter using RF Blockset blocks.
5. Calculate the ACLR/ACPR, occupied bandwidth, and channel power and depict the spectral mask by using the Spectrum Analyzer block.
6. Compute the CCDF and PAPR.
7. Downsample and filter the waveform by using an FIR Decimation block.
8. Extract the data symbols and measure the EVM by demodulating the baseband waveform.

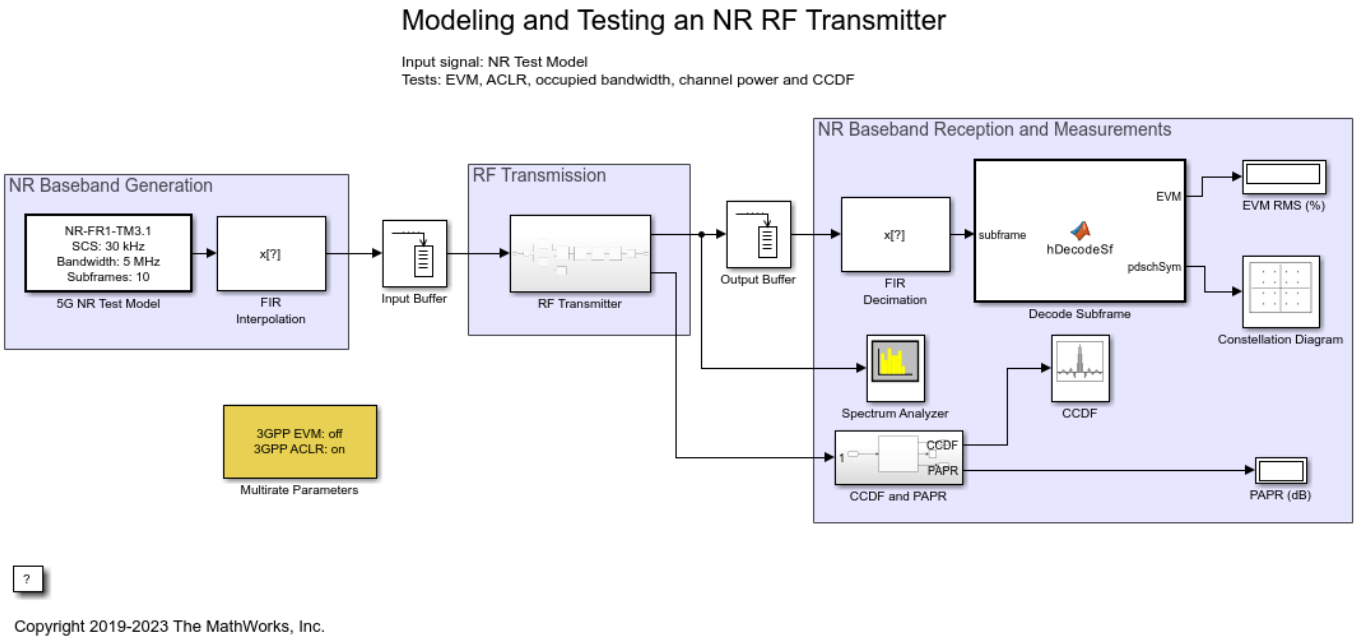
The Simulink model uses 5G Toolbox and DSP System Toolbox™ features to process the baseband signal (steps 1, 2, and 5-8) and uses RF Blockset blocks to model the RF transmitter (steps 3 and 4). This model supports Normal and Accelerator simulation modes.

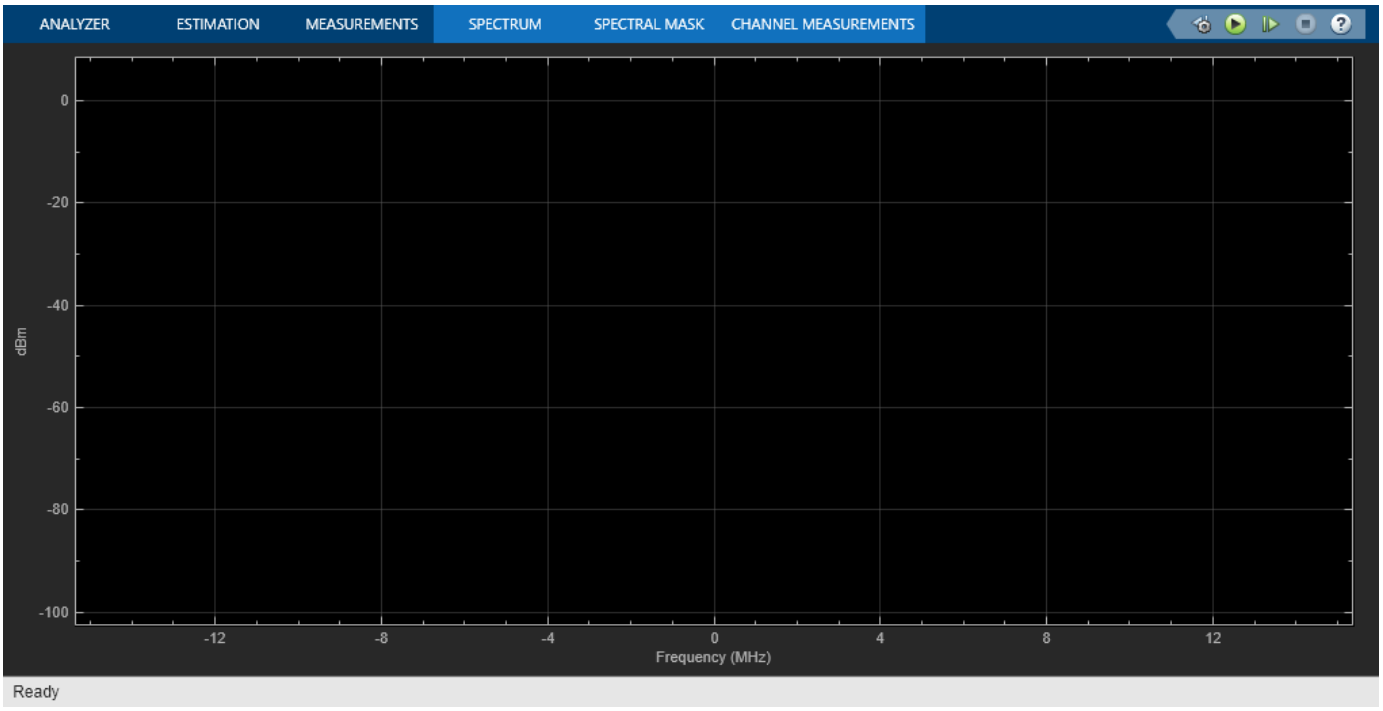
Simulink Model Structure

The model contains three main components:

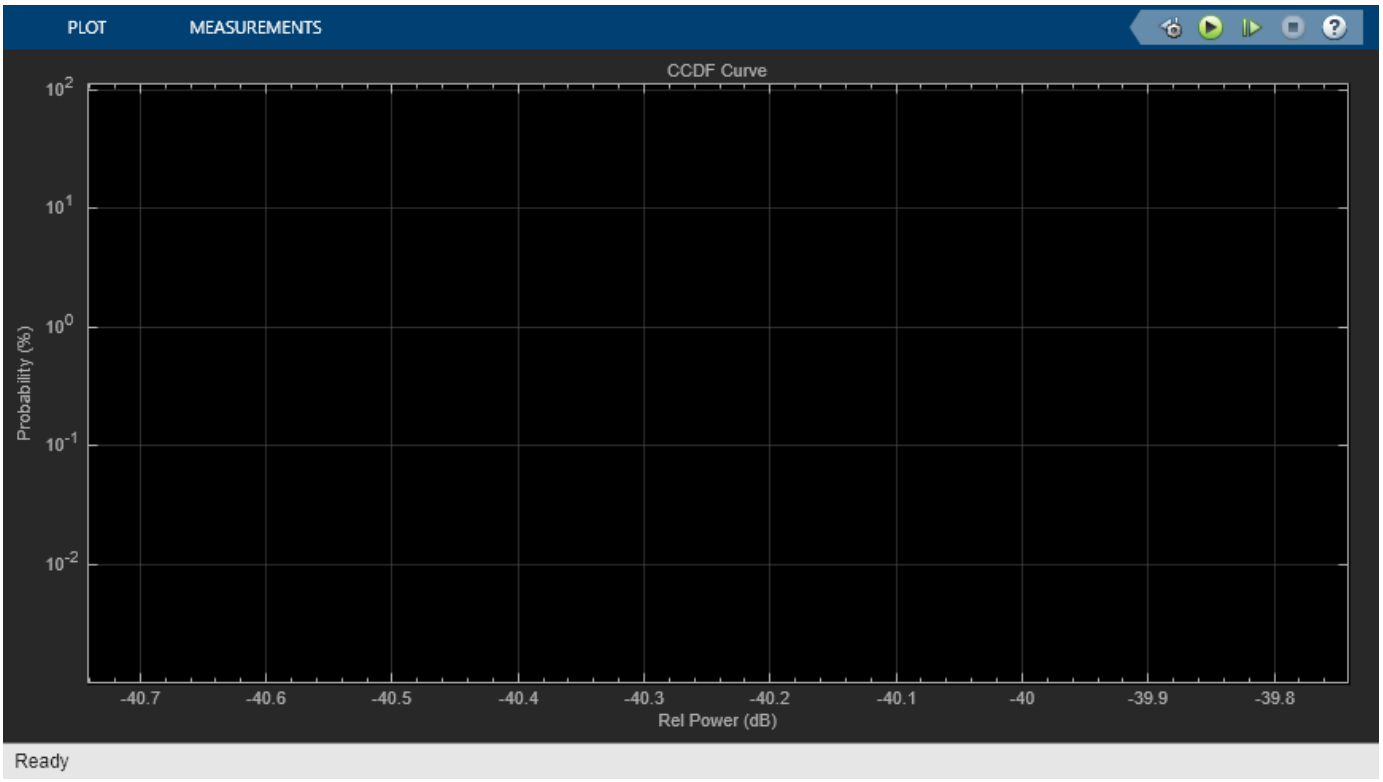
- NR Baseband Generation: generates the baseband NR waveform
- RF Transmission: models the effects of upconverting the waveform to the carrier frequency
- NR Baseband Reception and Measurements: performs the RF measurements and calculates EVM by demodulating the baseband waveform

```
modelName = 'NRModelingAndTestingRFTransmitterModel';
open_system(modelName);
```



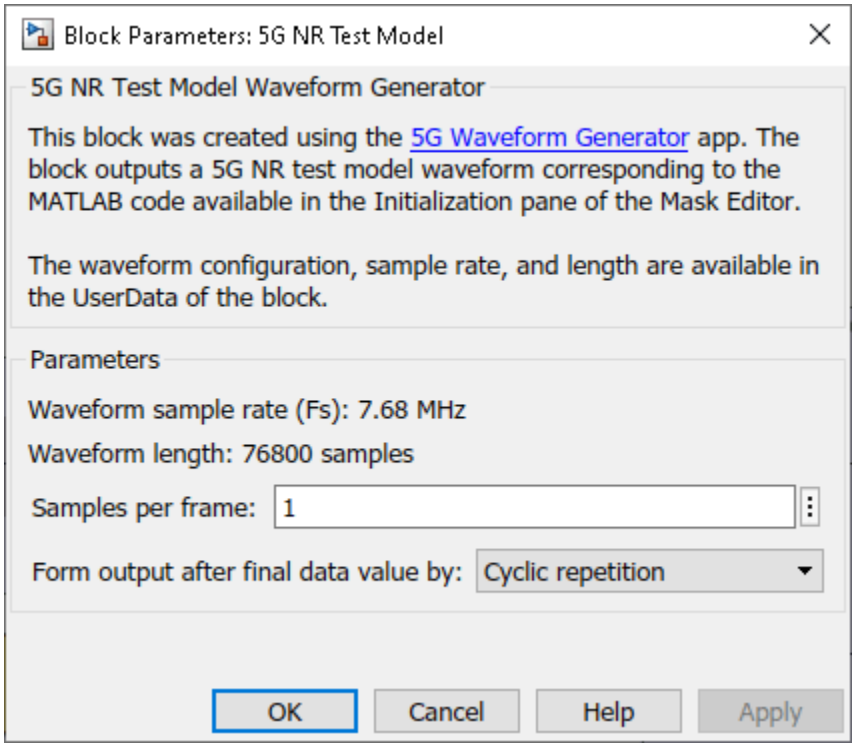



PLOT
MEASUREMENTS
1.2
0.8
0.4
0
-0.4
-0.8
-1.2
Quadrature Amplitude
-2
-1
0
1
2
In-phase Amplitude
Ready

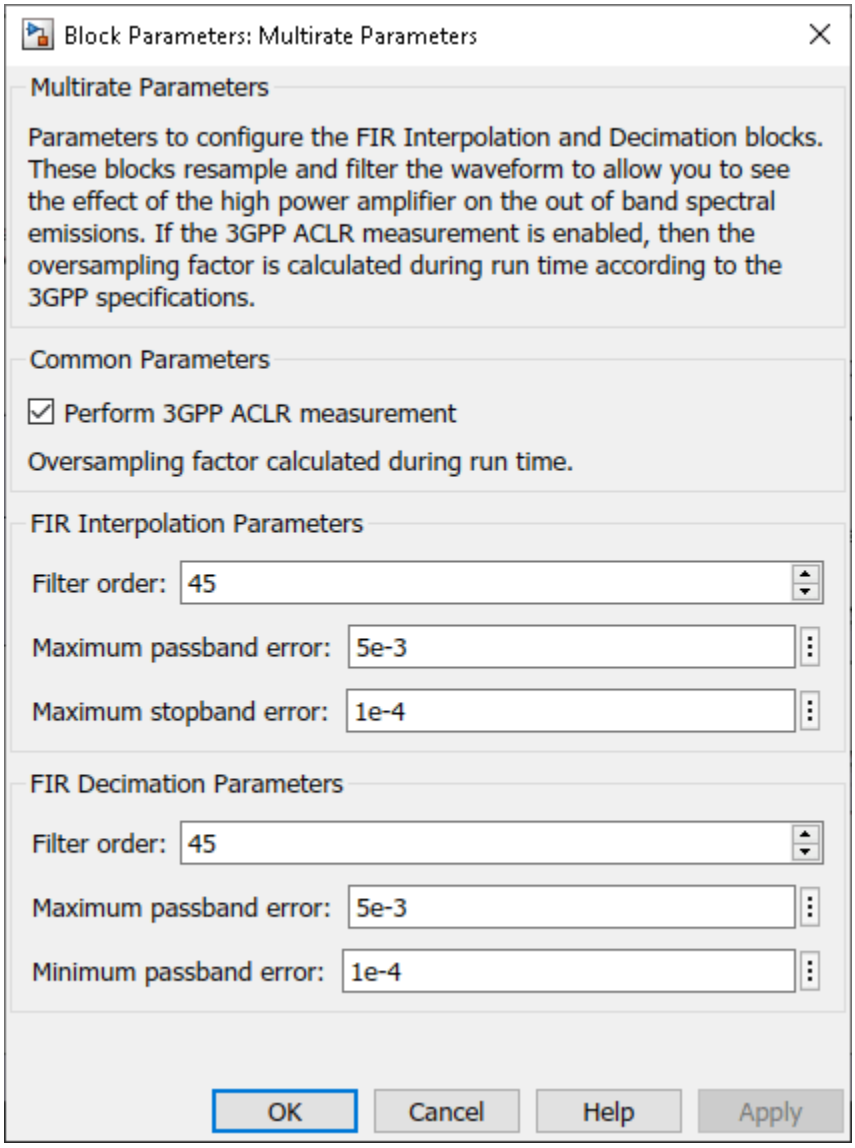

PLOT
MEASUREMENTS
CCDF Curve
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
Probability (%)
-40.7
-40.6
-40.5
-40.4
-40.3
-40.2
-40.1
-40
-39.9
-39.8
Rel Power (dB)
Ready

NR Baseband Generation

The 5G NR Test Model block transmits standard-compliant 5G NR test model 3.1 (NR-TM3.1) waveform for frequency range 1 (FR1), as defined in TS 38.141-1. This block is generated using the **5G Waveform Generator** app. You can access the waveform configuration parameters in the user data of the block. This example uses the **InitFcn** in the **Model callbacks** to store the structure available in the user data in a Base Workspace variable, `NRInfo`. For more information about this block, see Waveform From Wireless Waveform Generator App.



To display the effect of the high-power amplifier (HPA) on the out-of-band spectral emissions, the FIR Interpolation block oversamples and filters the waveform. At the output of the RF Transmitter Subsystem block, the FIR Decimation block downsamples the waveform back to the original sampling rate. The Multirate Parameters block provides an interface to configure the parameters of the FIR Interpolation and Decimation blocks.



The Multirate Parameters block also provides the option to enable or disable the 3GPP TS 38.141-1 ACLR test. To visualize the spectral regrowth, the ACLR test oversamples the waveform. If the **Perform 3GPP ACLR measurement** parameter of the Multirate Parameters block is enabled, the oversampling factor depends on the waveform configuration and is set such that the generated signal is capable of representing first and second adjacent channels. To specify the **Oversampling factor**, disable the 3GPP ACLR test. The **Oversampling factor** parameter defines the **Interpolation factor** in the FIR Interpolation block and the **Decimation factor** in the FIR Decimation block.

RF Transmission

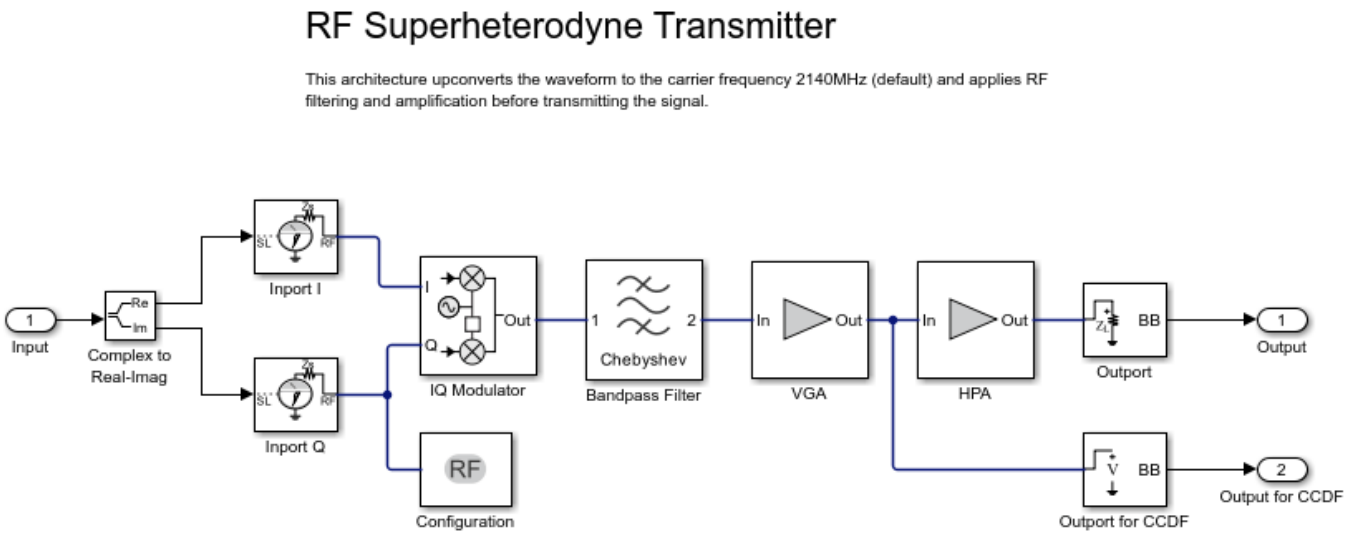
The RF Transmitter Subsystem block is based on a superheterodyne transmitter architecture. This architecture models the effects of upconverting the waveform to the carrier frequency by characterizing these RF components:

- IQ modulator consisting of mixers, phase shifter, and local oscillator

- Bandpass filter
- Power amplifier

In addition to these components, this RF Transmitter Subsystem block also includes a variable gain amplifier (VGA) to control the input back-off (IBO) level of the HPA.

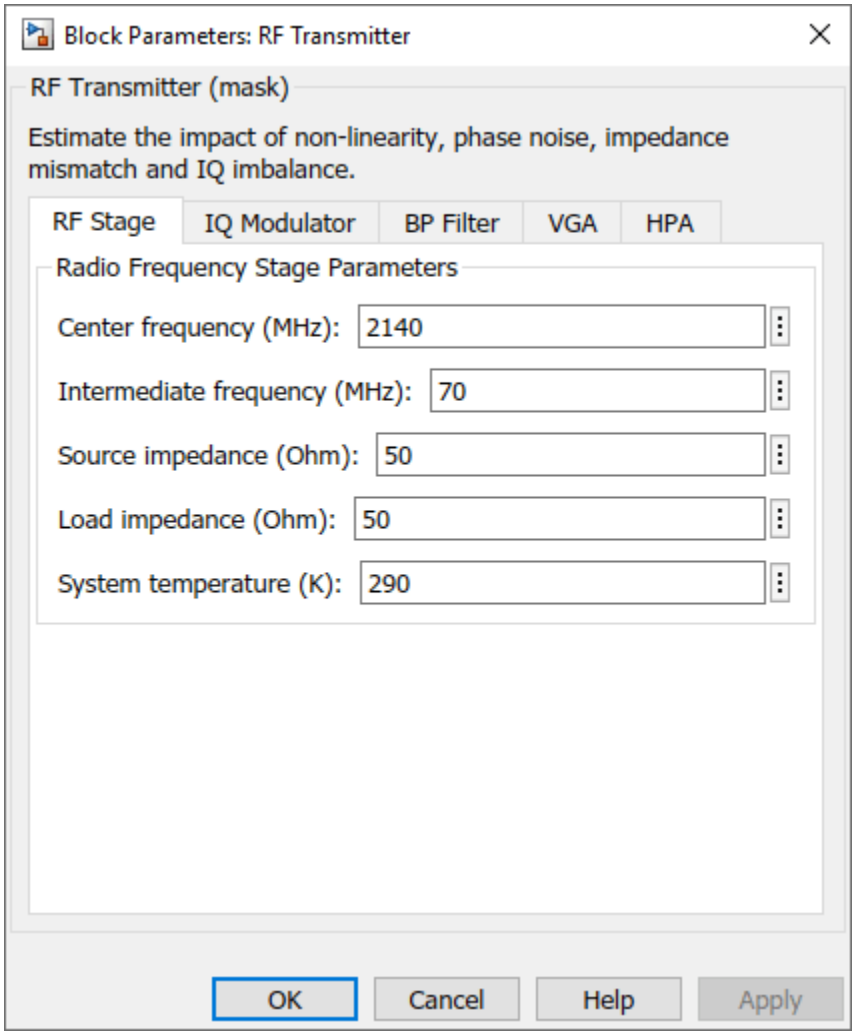
```
set_param(modelName,'Open','off');
RFTransmitterBlock = [modelName '/RF Transmitter'];
set_param(RFTransmitterBlock,'Open','on');
```



Use an Input Buffer block to send one sample at a time to the RF Transmitter Subsystem block.

The Inport block inside the RF Transmitter Subsystem block converts the Simulink complex baseband waveform into the RF Blockset Circuit Envelope simulation environment. The **Carrier frequencies** parameter of the Inport block specifies the center frequency of the carrier in the RF Blockset domain. The Outport block converts the RF Blockset signal back into Simulink complex baseband.

You can configure the RF Transmitter components using the RF Transmitter Subsystem block mask.



The RF Transmitter Subsystem block models typical impairments, including:

- I/Q imbalance as a result of gain or phase mismatches between the parallel sections of the transmitter chain dealing with the IQ signal paths.
- Phase noise as a secondary effect directly related to the thermal noise within the active devices of the oscillator.
- HPA nonlinearities due to DC power limitation when the amplifier works in saturation region.

Before sending the samples onto the Decode Subframe block, the Output Buffer (after the RF Transmitter) buffers all samples within a subframe.

The use of buffers in the model generates time delays. As the duration of the delay is equivalent to the transmission of a subframe, the Decode Subframe block does not demodulate the first subframe.

NR Baseband Reception and Measurements

The Decode Subframe block performs OFDM demodulation of the received subframe, channel estimation, and equalization to recover and plot the PDSCH symbols in the Constellation Diagram. This block also averages the EVM over time and frequency and plots these values:

- EVM per OFDM symbol: EVM averaged over each OFDM symbol.
- EVM per slot: EVM averaged over the allocated PDSCH symbols within a slot.
- EVM per subcarrier: EVM averaged over the allocated PDSCH symbols within a subcarrier.
- Overall EVM: EVM averaged over the allocated PDSCH symbols transmitted. It is reflected in the EVM RMS display block.

Annex B and Annex C of TS 38.104 define an alternative method for computing the EVM in FR1 and FR2, respectively. You can enable this method by enabling **Perform 3GPP EVM measurement** parameter of the Multirate Parameters block.

The Spectrum Analyzer block provides frequency-domain measurements such as ACLR (referred to as ACPR) and occupied bandwidth. If you disable the **Perform 3GPP ACLR measurement** parameter of the Multirate Parameters block, you can select the oversampling factor and the Spectrum Analyzer block measures the occupied bandwidth.

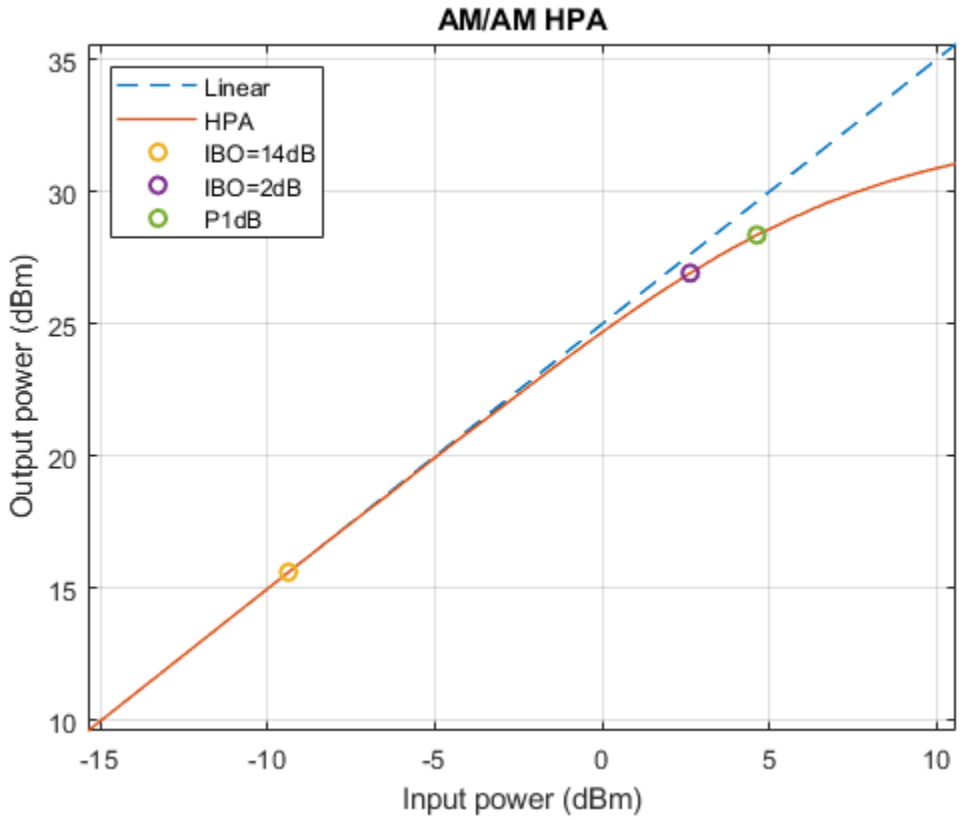
A Power Meter block, called CCDF and PAPR, connected at the input of the HPA block depicts the CCDF and PAPR measurements.

The Decode Subframe block discards the first received subframe (1 ms) due to processing delays. Therefore, to receive one frame, you must simulate 11 ms for FDD (10 ms for the frame plus 1 ms for the initially discarded subframe period). If the simulation time is longer than 11 ms, the 5G NR Test Model block cyclically transmits the same NR frame.

Effect of Power Amplifier Nonlinearities

To characterize the impact of HPA nonlinearities in the EVM and ACLR evaluations, you can measure the amplitude-to-amplitude modulation (AM/AM) of the HPA. The AM/AM refers to the output power levels in terms of the input power levels. The local function `hPlotHPACurve` displays the AM/AM characteristic of the HPA selected for this model.

```
hPlotHPACurve();
figHPA = gcf;
```



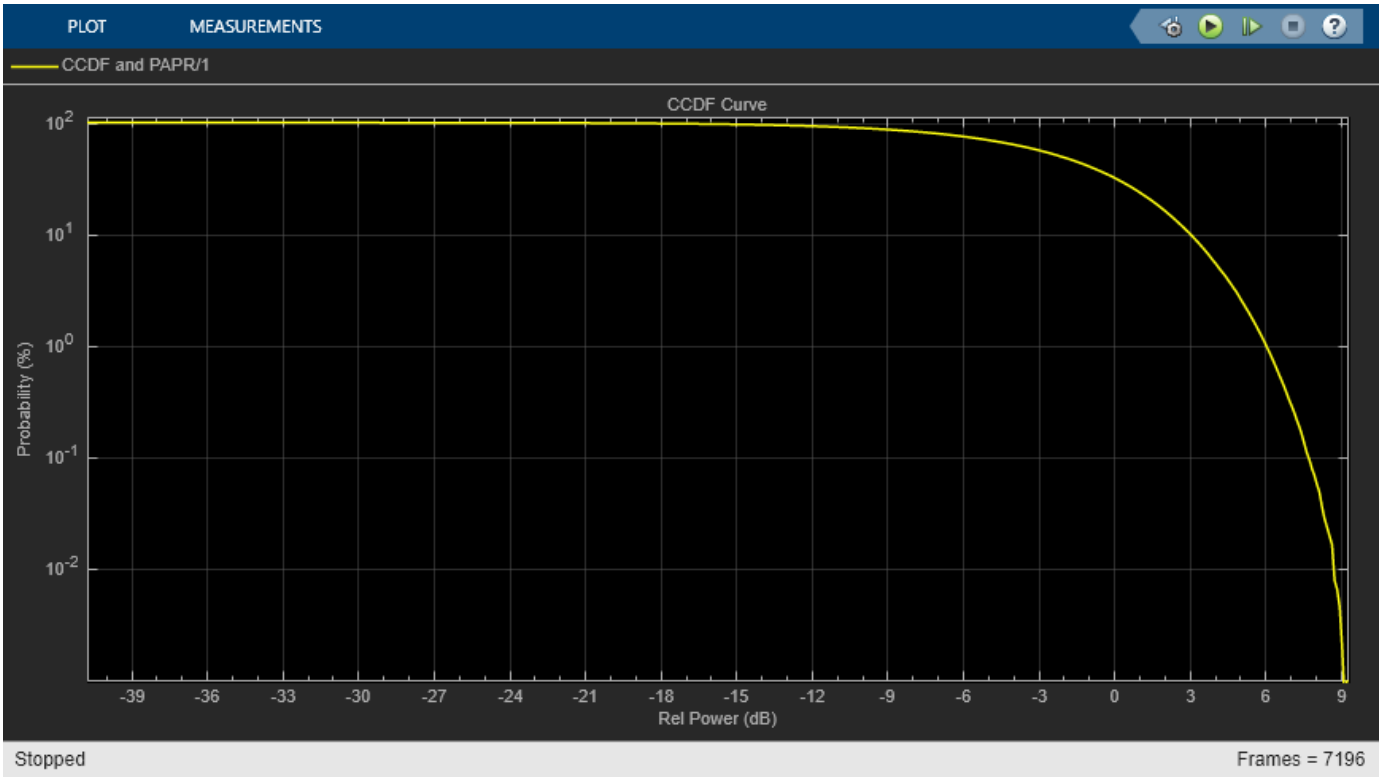
P1dB is the power at 1 dB compression point and is usually used as a reference when selecting the IBO level of the HPA. You can see the HPA impact on the RF transmitter by analyzing the EVM and ACLR results for different operating points of the HPA. For example, compare the case when IBO = 14 dB, corresponding to HPA operating in linear region, with the case when IBO = 2 dB, corresponding to HPA operating in full saturation. The gain of the VGA controls the IBO level. To keep a VGA linear behavior, select gain values lower than 20 dB.

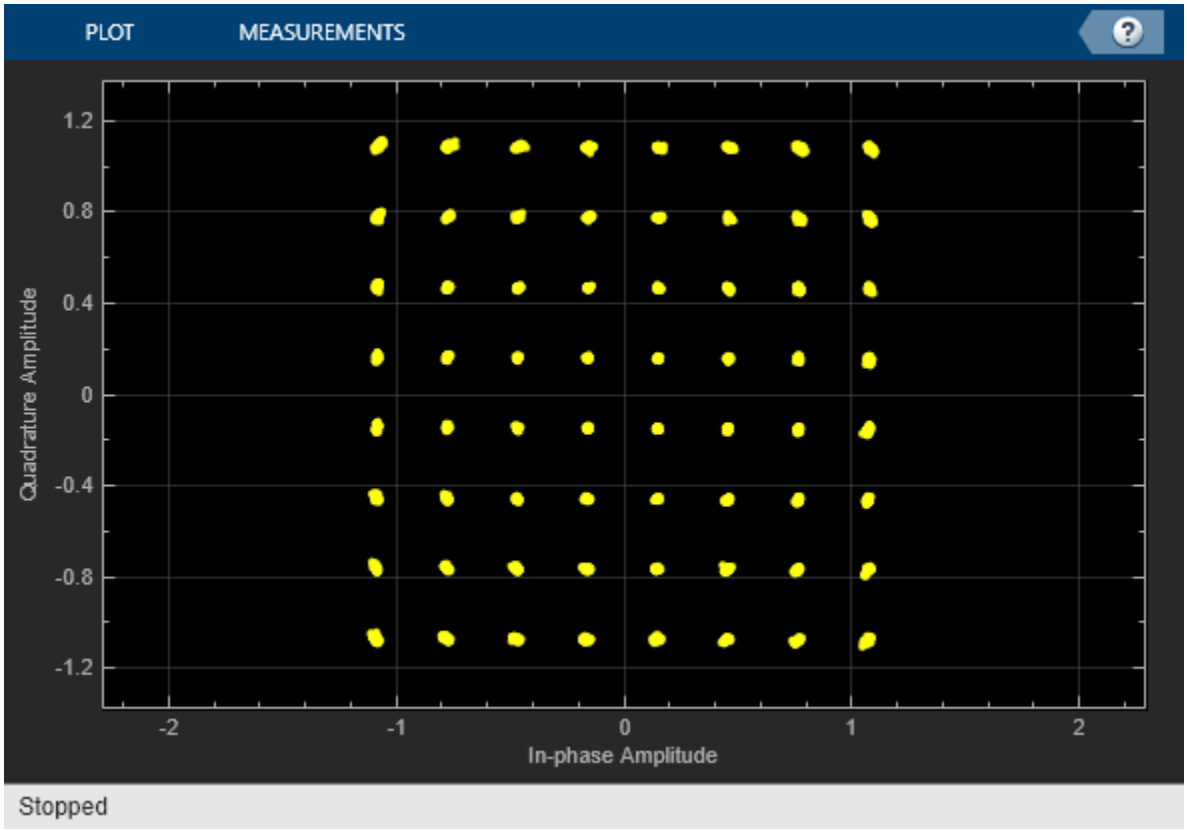
- *Linear HPA (IBO = 14 dB)*. To operate at an IBO level of 14 dB, set the **Available power gain** parameter of the VGA block to 0 dB. Run the simulation to capture, for instance, 4 subframes (5 ms). During simulation, the model displays the EVM and ACLR measurements and the constellation diagram.

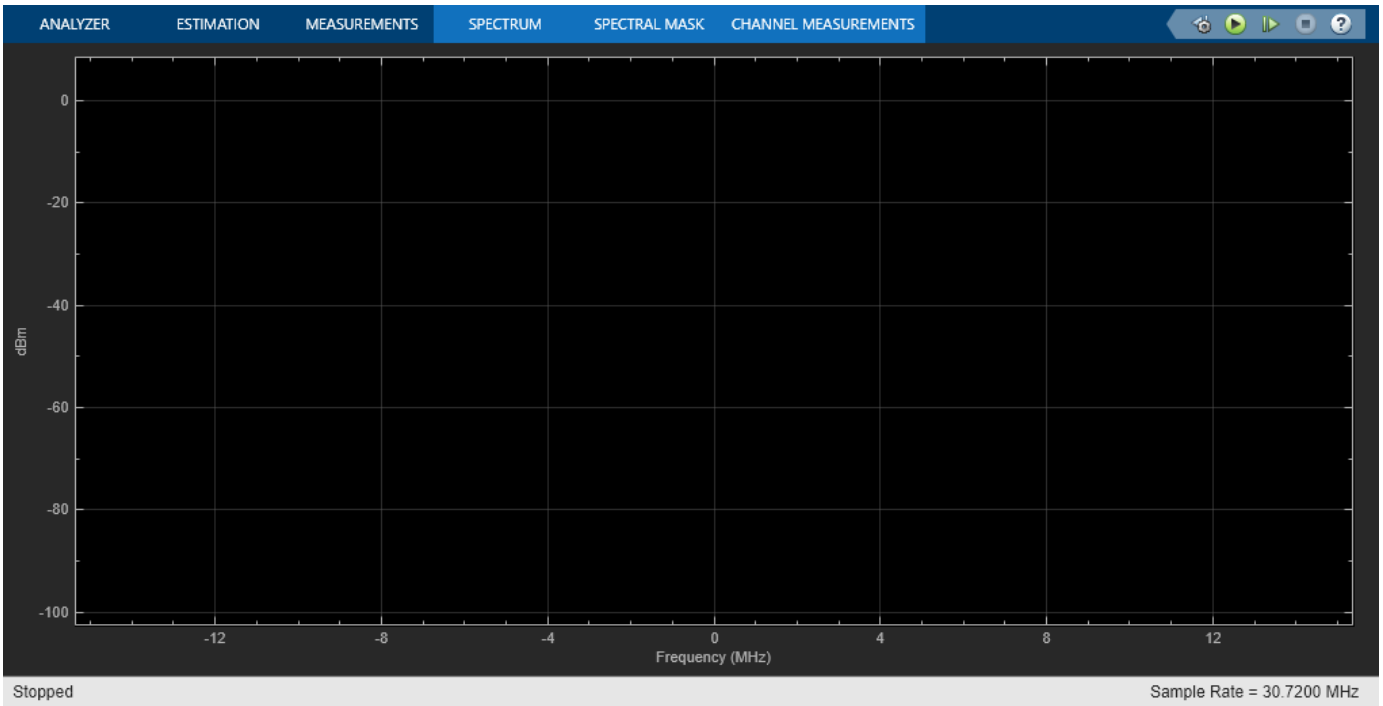
```
set_param(RFTransmitterBlock,'vgaGain','0');
sim(modelName);
```

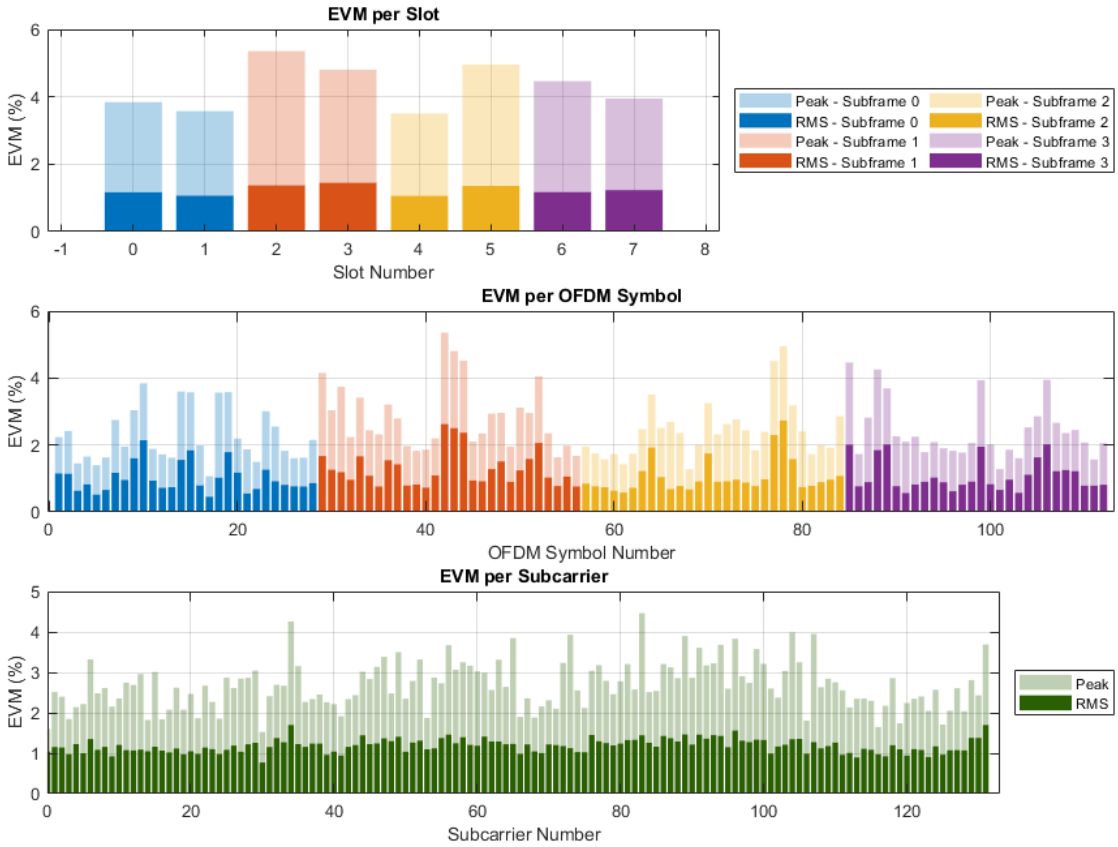
```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 1.168 3.841%
PDSCH RMS EVM, Peak EVM, slot 1: 1.064 3.578%
Averaged overall PDSCH RMS EVM: 1.118%
Overall PDSCH Peak EVM = 3.8414%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 2: 1.370 5.358%
PDSCH RMS EVM, Peak EVM, slot 3: 1.444 4.804%
Averaged overall PDSCH RMS EVM: 1.408%
Overall PDSCH Peak EVM = 5.3577%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 4: 1.058 3.506%
```

```
PDSCH RMS EVM, Peak EVM, slot 5: 1.355 4.955%
Averaged overall PDSCH RMS EVM: 1.216%
Overall PDSCH Peak EVM = 4.9547%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 6: 1.173 4.465%
PDSCH RMS EVM, Peak EVM, slot 7: 1.229 3.950%
Averaged overall PDSCH RMS EVM: 1.201%
Overall PDSCH Peak EVM = 4.465%
```




PLOT
MEASUREMENTS
1.2
0.8
0.4
0
-0.4
-0.8
-1.2
Quadrature Amplitude
-2
-1
0
1
2
In-phase Amplitude
Stopped


ANALYZER
ESTIMATION
MEASUREMENTS
SPECTRUM
SPECTRAL MASK
CHANNEL MEASUREMENTS
0
-20
-40
-60
-80
-100
dBm
-12
-8
-4
0
4
8
12
Frequency (MHz)
Stopped
Sample Rate = 30.7200 MHz



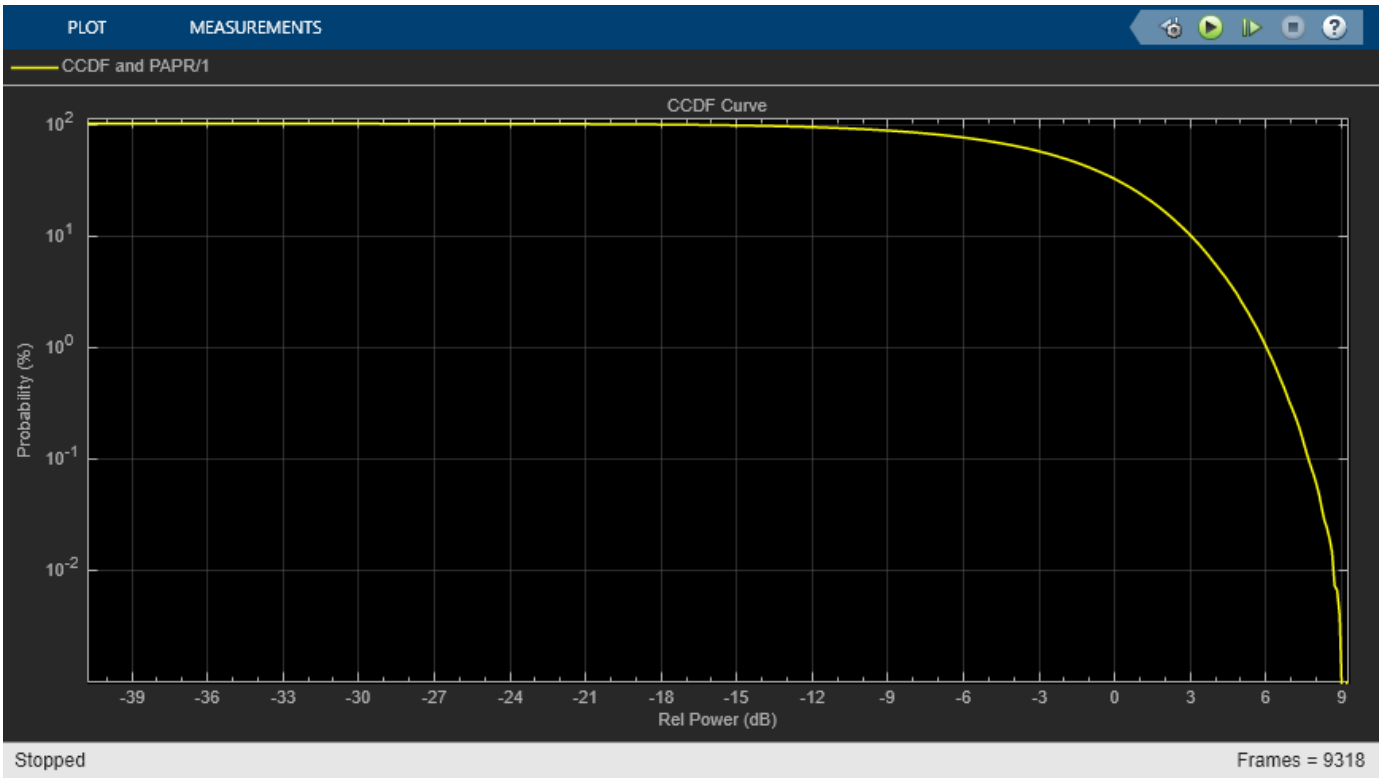
According to TS 38.104, the minimum required ACLR for conducted measurements is 45 dB and the maximum required EVM when the constellation is 64-QAM is 8%. As the ACLR values are higher than 45 dB, and the overall EVM, which is around 1.2%, is lower than 8%, both measurements fall within the requirements.

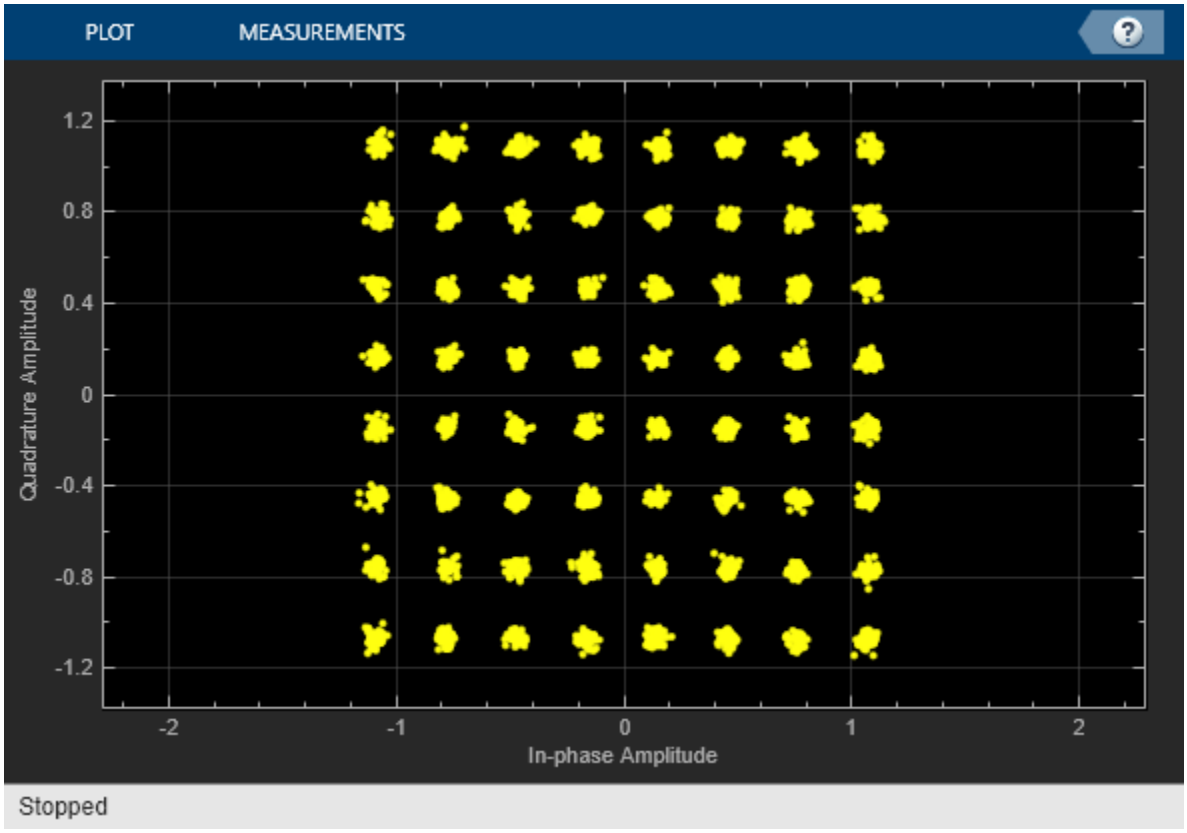
- *Nonlinear HPA (IBO = 2 dB)*. To operate at an IBO level of 2 dB, set the **Available power gain** parameter of the VGA block to 12 dB.

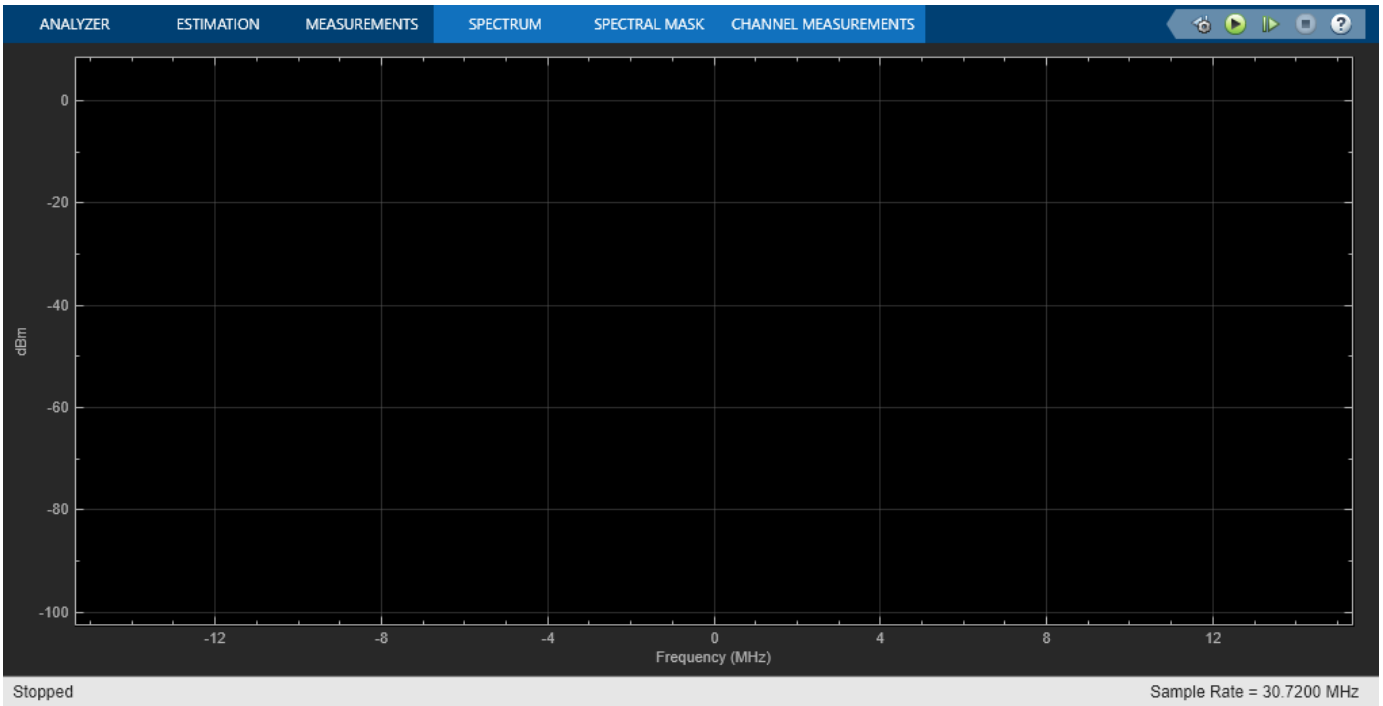
```
set_param(RFTransmitterBlock,'vgaGain','12');
sim(modelName);

% Restore to default parameters
set_param(RFTransmitterBlock,'vgaGain','0');
```

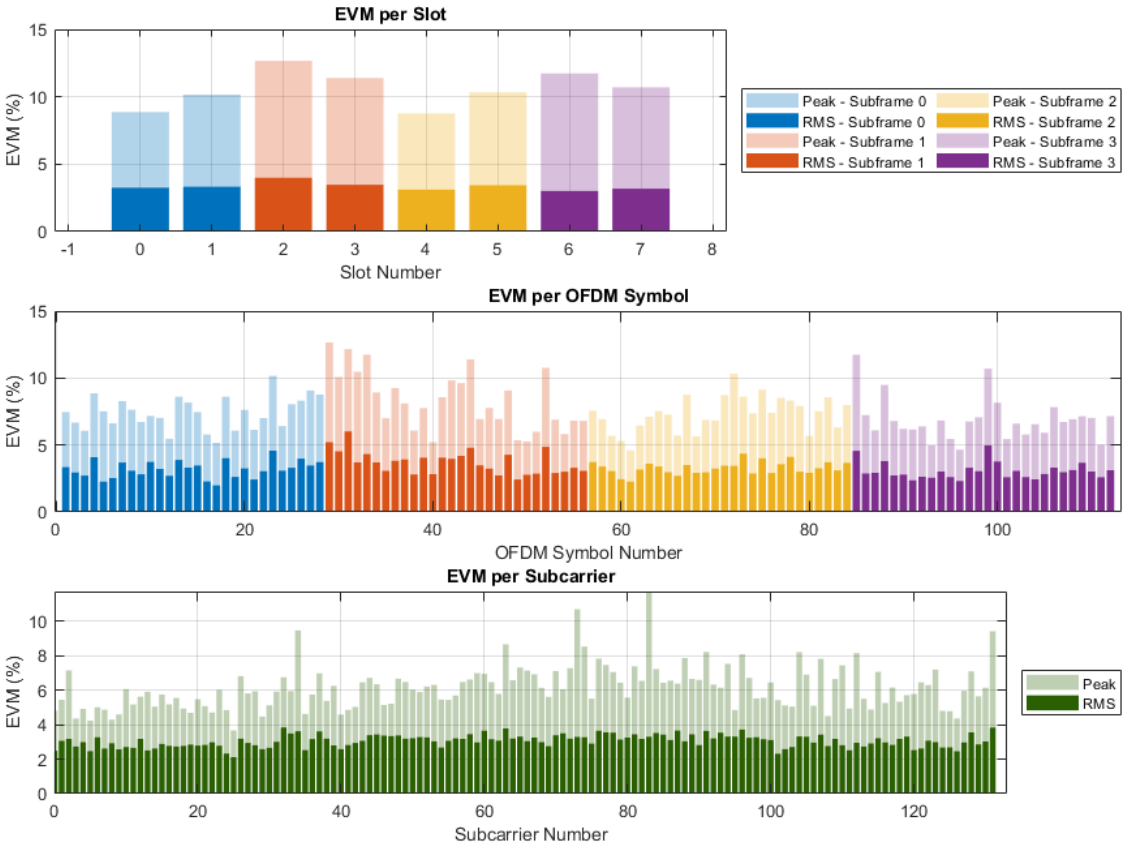
```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 3.248 8.864%
PDSCH RMS EVM, Peak EVM, slot 1: 3.332 10.166%
Averaged overall PDSCH RMS EVM: 3.290%
Overall PDSCH Peak EVM = 10.1664%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 2: 3.994 12.668%
PDSCH RMS EVM, Peak EVM, slot 3: 3.473 11.397%
Averaged overall PDSCH RMS EVM: 3.743%
Overall PDSCH Peak EVM = 12.6681%
```

```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 4: 3.113 8.757%
PDSCH RMS EVM, Peak EVM, slot 5: 3.441 10.331%
Averaged overall PDSCH RMS EVM: 3.281%
Overall PDSCH Peak EVM = 10.3314%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 6: 3.009 11.736%
PDSCH RMS EVM, Peak EVM, slot 7: 3.180 10.707%
Averaged overall PDSCH RMS EVM: 3.096%
Overall PDSCH Peak EVM = 11.7364%
```




PLOT
MEASUREMENTS
1.2
0.8
0.4
0
-0.4
-0.8
-1.2
Quadrature Amplitude
-2
-1
0
1
2
In-phase Amplitude
Stopped


ANALYZER
ESTIMATION
MEASUREMENTS
SPECTRUM
SPECTRAL MASK
CHANNEL MEASUREMENTS
0
-20
-40
-60
-80
-100
dBm
-12
-8
-4
0
4
8
12
Frequency (MHz)
Stopped
Sample Rate = 30.7200 MHz



Compared to the previous case, the constellation diagram is distorted and the spectral regrowth is higher. In terms of the measurements, the first adjacent channel ACLR does not fall within the requirements of TS 38.104 and the overall EVM, which is around 3%, is higher than in the previous case.

Summary and Further Exploration

This example demonstrates how to model and test an NR RF transmitter in Simulink. The RF transmitter consists of an IQ modulator, a bandpass filter and amplifiers. To evaluate the performance, the Simulink model considers ACLR and EVM measurements. The example highlights the effect of HPA nonlinearities on the performance of the RF Transmitter. You can explore the impact of altering other impairments as well. For example:

- Increase I/Q imbalance by using the **I/Q gain mismatch (dB)** and **I/Q phase mismatch (Deg)** parameters on the **IQ Modulator** tab of the RF Transmitter Subsystem block.
- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **IQ Modulator** tab of the RF Transmitter Subsystem block.

The RF Transmitter is configured to work with the current NR-TM waveform parameters selected in the 5G NR Test Model block and with a carrier centered at 2140 MHz (FR1). If you modify the **Center frequency (MHz)** parameter of the RF Transmitter Subsystem block or the waveform configuration of the 5G NR Test Model block, check if you need to update the parameters of the RF Transmitter

components and the FIR filters as these parameters are set to work with the current example configuration. For instance, a change in the carrier frequency requires revising the **Passband frequencies** and **Stopband frequencies** parameters of the Bandpass Filter block inside the RF Transmitter. If you select a bandwidth wider than 20 MHz, check if you need to update the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the IQ Modulator (RF Blockset) block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution.

You can use this example as the basis for testing NR-TM waveforms for different RF configurations. You can try replacing the RF Transmitter Subsystem block by another RF subsystem and then configure the model accordingly.

To use a different NR-TM waveform, open the **5G Waveform Generator** app, select the NR-TM configuration, and export a new block. For more information on how to generate and use this block, see “Generate Wireless Waveform in Simulink Using App-Generated Block”.

References

1. 3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
2. 3GPP TS 38.104. "NR; Base Station (BS) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local functions

```
function hPlotHPACurve()
% Plots the AM/AM characteristic of the HPA.

    % HPA input and output power levels obtained from simulation
    driverGain = [-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 ...
        18 19 20]; %#ok<NASGU> % Gain of the VGA (just for reference)
    inputPower = [-15.382 -14.382 -13.382 -12.382 -11.382 -10.382 -9.382 -8.382 ...
        -7.382 -6.356 -5.383 -4.383 -3.357 -2.357 -1.358 -0.358 0.641 1.640 2.639 ...
        3.637 4.635 5.633 6.629 7.624 8.616 9.606 10.592];  % HPA input power
    outputPower = [9.6140 10.6140 11.6140 12.6140 13.6140 14.6140 15.614 16.605 ...
        17.595 18.582 19.566 20.545 21.517 22.477 23.436 24.361 25.261 26.121 26.926 ...
        27.684 28.368 28.968 29.531 30 30.412 30.764 31.069];  % HPA output power
    linearGain = outputPower(1) - inputPower(1);  % Linear gain

    % Plot AM/AM characteristic of HPA
    plot(inputPower,inputPower+linearGain,'--','LineWidth',1), hold on, grid on;
    plot(inputPower,outputPower,'LineWidth',1);
    plot(inputPower(7),outputPower(7),'o','LineWidth',1.5); % IBO=14dB
    plot(inputPower(19),outputPower(19),'o','LineWidth',1.5); % IBO=2dB
    plot(inputPower(21),outputPower(21),'o','LineWidth',1.5); % P1dB
    legend('Linear','HPA','IBO=14dB','IBO=2dB','P1dB','Location','NorthWest');
    xlabel('Input power (dBm)');
    ylabel('Output power (dBm)');
    title('AM/AM HPA');
    axis('tight');
```

```
end
```

See Also

Related Examples

- “Modeling and Testing an NR RF Receiver with LTE Interference” on page 7-62
- “Generate Wireless Waveform in Simulink Using App-Generated Block”

Modeling and Testing an NR RF Receiver with LTE Interference

The example shows how to characterize the impact of RF impairments in the RF reception of a new radio (NR) waveform when coexisting with a long-term evolution (LTE) interference. The baseband waveforms are generated using 5G Toolbox™ and LTE Toolbox™, and the RF receiver is modeled using RF Blockset™.

Introduction

This example characterizes the impact of LTE interference on the RF reception of an NR waveform. To evaluate the impact of the interference, the example performs these measurements:

- Error vector magnitude (EVM): vector difference at a given time between the ideal (transmitted) signal and the measured (received) signal. Annex B and Annex C of TS 38.104 define an alternative method for computing the EVM in FR1 and FR2, respectively.
- Adjacent channel leakage ratio (ACLR): measure of the amount of power leaking into adjacent channels. It is defined as the ratio of the filtered mean power centered on the assigned channel frequency to the filtered mean power centered on an adjacent channel frequency.
- Occupied bandwidth: bandwidth that contains 99% of the total integrated power of the signal, centered on the assigned channel frequency.
- Channel power: filtered mean power centered on the assigned channel frequency.

The impact of the receiver RF impairments such as phase noise and amplifier nonlinearities are also considered.

The example works on a subframe by subframe basis and uses a Simulink model to perform these steps:

1. Generate the baseband NR waveform using 5G Toolbox features.
2. Generate the baseband LTE waveform (interference) using LTE Toolbox features.
3. Match the sampling rate of the two signals by using the Sample-Rate Match block.
4. To capture spectral regrowth, oversample the waveforms by a factor of 4 or 5 by using the FIR Interpolation block.
5. Import the baseband waveforms into the RF Receiver Subsystem block implemented by using RF Blockset blocks. The model provides an RF frequency to each waveform to carry the baseband information in RF Blockset.
6. Model the effects of downconverting the NR waveform to an intermediate frequency by using the RF Receiver Subsystem block. This block models the impairments introduced by an RF receiver using RF Blockset blocks.
7. Calculate the ACLR/ACPR, occupied bandwidth, and channel power using the Spectrum Analyzer block.
8. Downsample the NR waveform by using an FIR Decimation block to compensate for the upsampling performed by the FIR Interpolation block.
9. Downsample the NR waveform by using an FIR Rate Conversion block to compensate for any upsampling performed by the Sample-Rate Match block.
10. Extract the data symbols and measure the EVM by demodulating the baseband waveform.

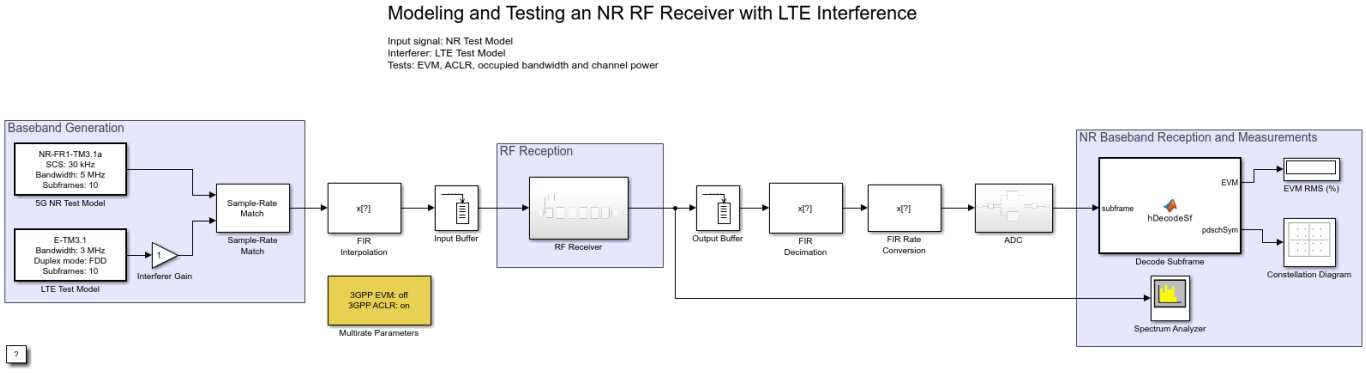
The Simulink model uses 5G Toolbox, LTE Toolbox, and DSP System Toolbox™ features to process the baseband waveforms (steps 1-4 and 7-10) and uses RF Blockset blocks to model the RF receiver (steps 5 and 6). This model supports Normal and Accelerator simulation modes.

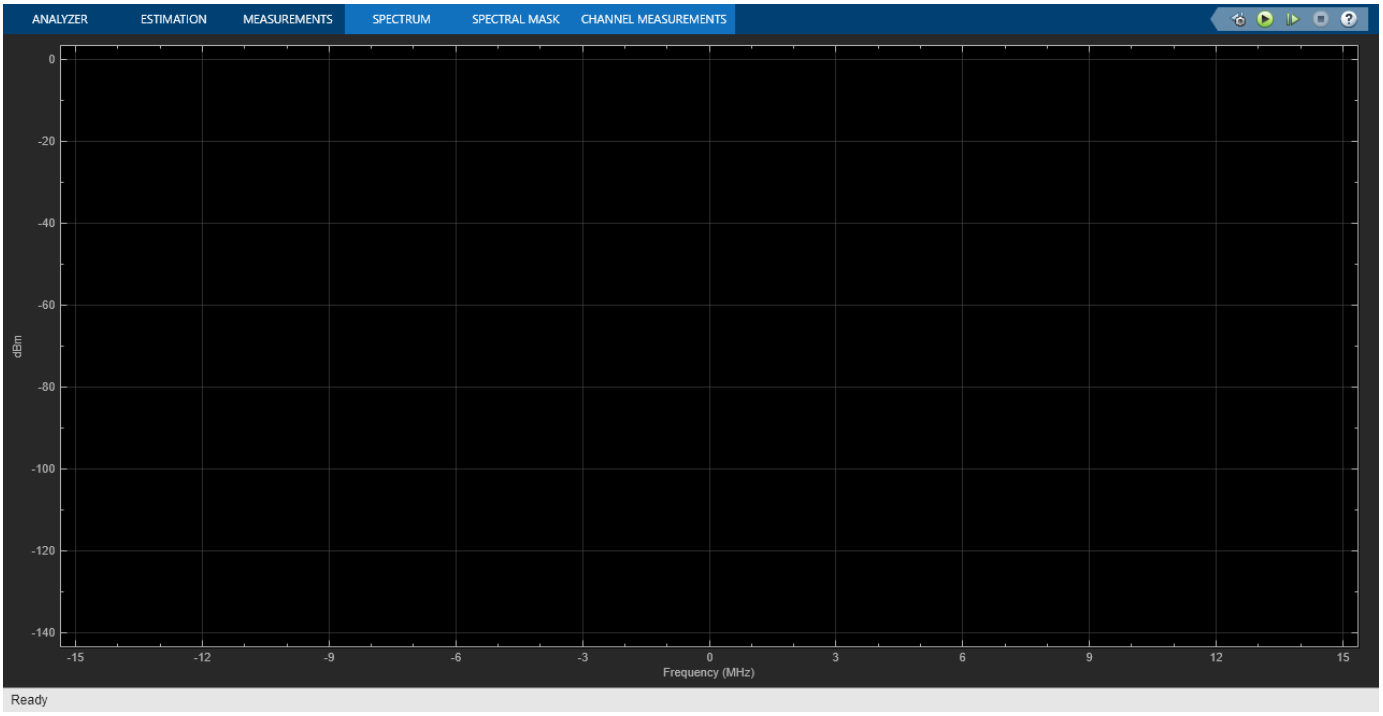
Simulink Model Structure

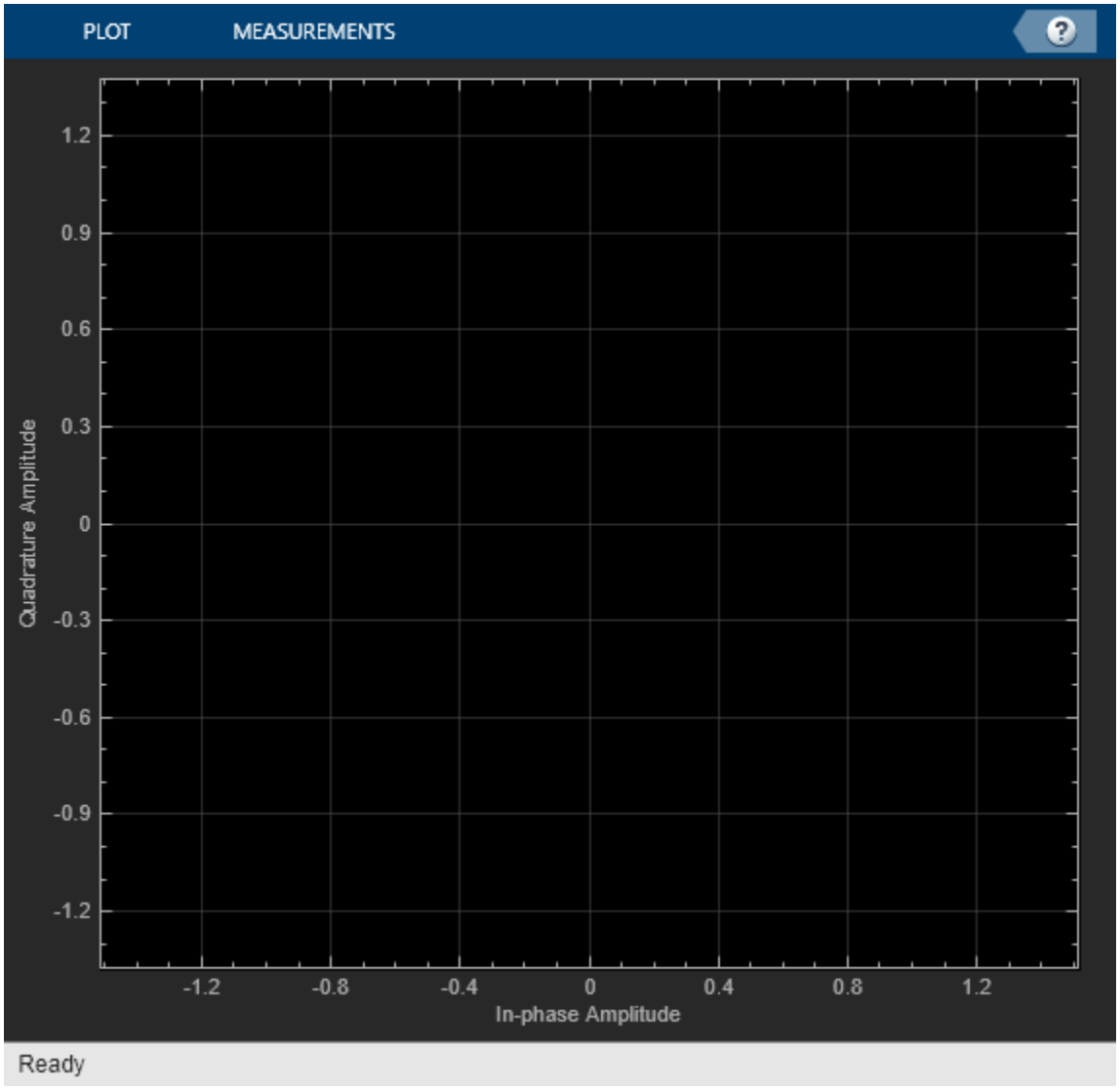
The model contains three main parts:

- Baseband Generation: generates the baseband NR and LTE waveforms
- RF Reception: models the effects of combining both waveforms in the RF domain and downconverting the NR waveform to an intermediate frequency
- NR Baseband Reception and Measurements: performs the RF measurements and calculates EVM by demodulating the NR baseband waveform

```
modelName = 'NewRadioRFReceiverWithLTEInterferenceModel';
open_system(modelName);
```

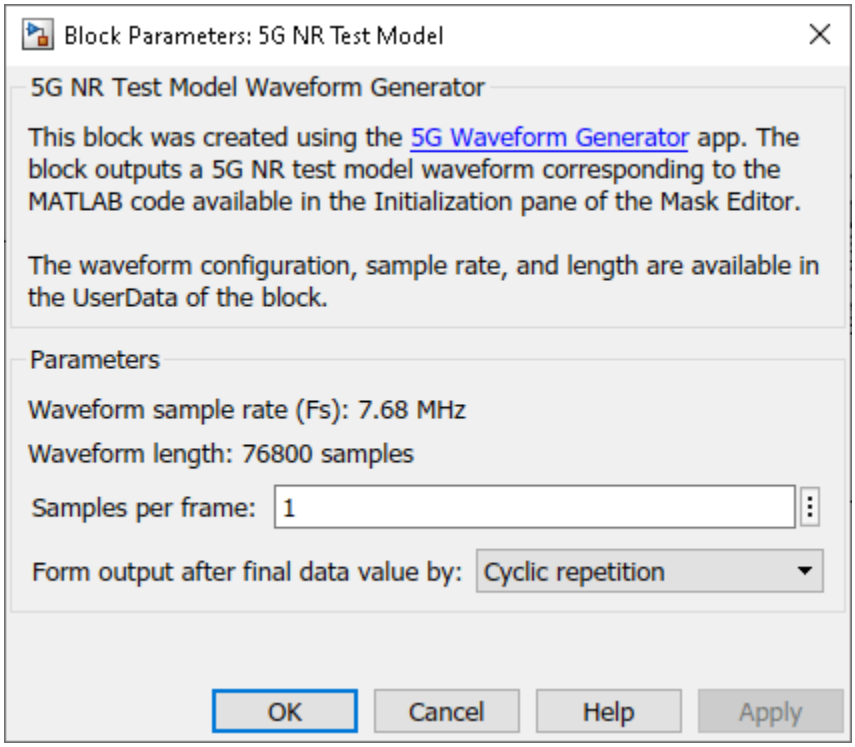




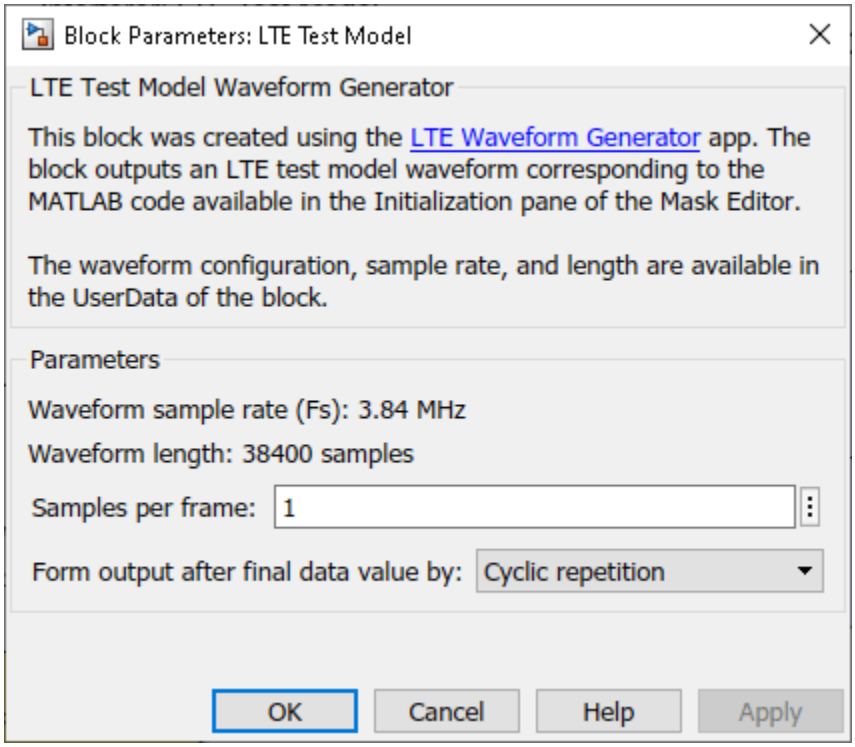


Baseband Generation

The 5G NR Test Model block transmits the standard-compliant 5G NR test model 3.1 (NR-TM3.1) waveform for frequency range 1 (FR1), as defined in TS 38.141-1. This block is generated using the **5G Waveform Generator** app. You can access the waveform configuration parameters in the user data of the block. This example uses the **InitFcn** in the **Model callbacks** to store the structure available in the user data in a Base Workspace variable, `NRInfo`. For more information about this block, see Waveform From Wireless Waveform Generator App.



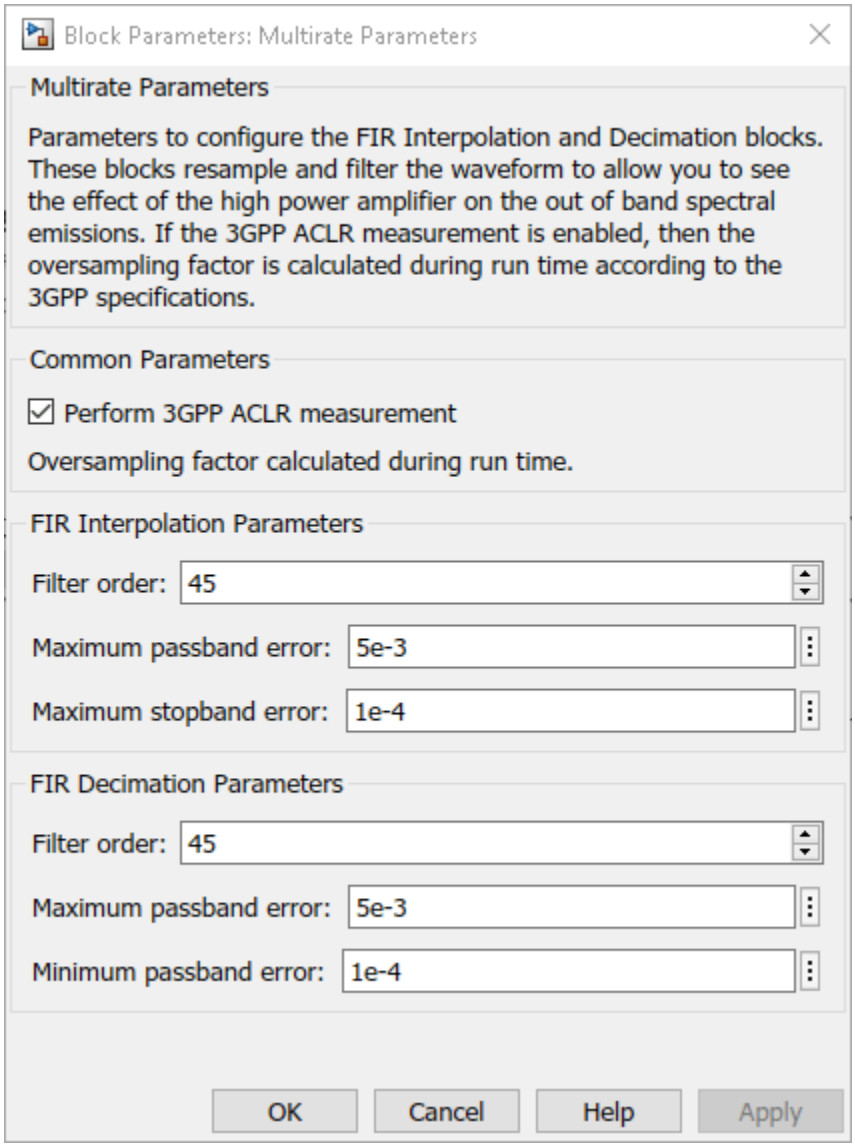
Similarly, the LTE Test Model block transmits the standard-compliant LTE test model 3.1 (E-TM3.1) waveform, as defined in TS 36.141. This block is generated using the **LTE Waveform Generator (LTE Toolbox)** app. The example also uses the **InitFcn** in the **Model callbacks** to store the E-TM configuration available in the block user data in a Base Workspace variable, `LTEInfo`. For more information about this block, see Waveform From Wireless Waveform Generator App (LTE Toolbox).



The Interferer Gain block controls the level of the LTE interference. You can disable the interference by setting this gain to 0.

The Sample-Rate Match block upsamples the waveform with lower sample rate to match the sample rate of the other waveform. The waveforms must have the same sample rate when combining the waveforms in the RF Receiver block. The Sample-Rate Match block also concatenates both waveforms horizontally, one column per waveform.

To capture spectral regrowth, the FIR Interpolation block oversamples and filters both waveforms. The Multirate Parameters block provides an interface to configure the parameters of the FIR Interpolation and Decimation blocks.



The Multirate Parameters block also provides the option to enable or disable the 3GPP TS 38.141-1 ACLR test. To visualize the spectral regrowth, the ACLR test oversamples the waveform. If the **Perform 3GPP ACLR measurement** parameter of the Multirate Parameters block is enabled, the oversampling factor depends on the waveform configuration and is set such that the generated signal is capable of representing first and second adjacent channels. To specify the **Oversampling factor**, disable the 3GPP ACLR test. The **Oversampling factor** parameter defines the **Interpolation factor** in the FIR Interpolation block and the **Decimation factor** in the FIR Decimation block.

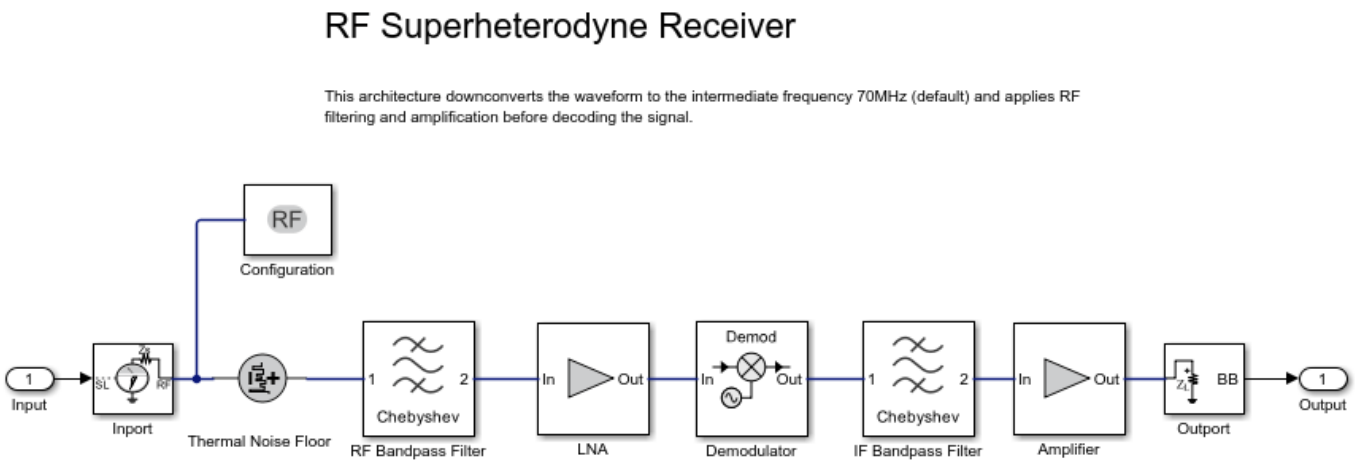
RF Reception

The RF Receiver Subsystem block is based on a superheterodyne receiver architecture. This architecture models the effects of downconverting the NR waveform to an intermediate frequency by characterizing these RF components:

- Bandpass filters

- Low noise amplifiers
- Demodulator consisting of mixers, phase shifter, and local oscillator

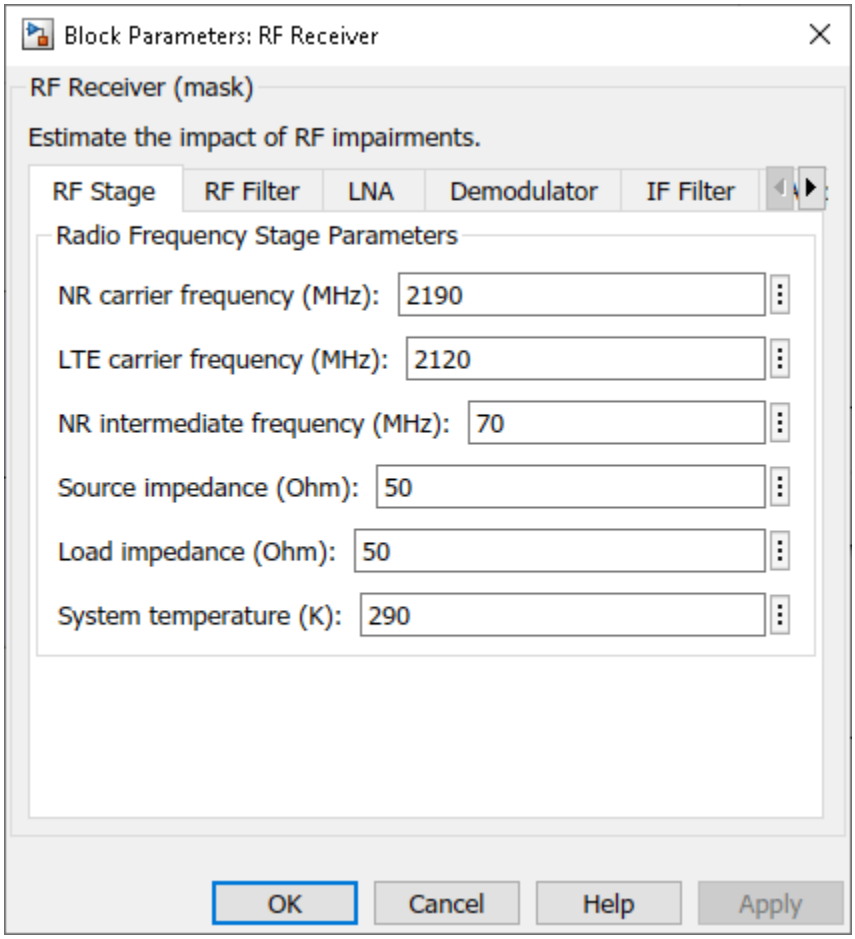
```
set_param(modelName,'Open','off');
set_param([modelName '/RF Receiver'],'Open','on');
```



Use an Input Buffer block to send one sample at a time to the RF Receiver Subsystem block.

The Inport block inside the RF Receiver Subsystem block converts the two concatenated Simulink complex baseband waveforms into the RF Blockset Circuit Envelope simulation environment. The **Carrier frequencies** parameter of the Inport block specifies the center frequency of the carriers in the RF Blockset domain. The Outport block converts the RF Blockset signals back into Simulink complex baseband.

You can configure the RF Receiver components using the RF Receiver Subsystem block mask.

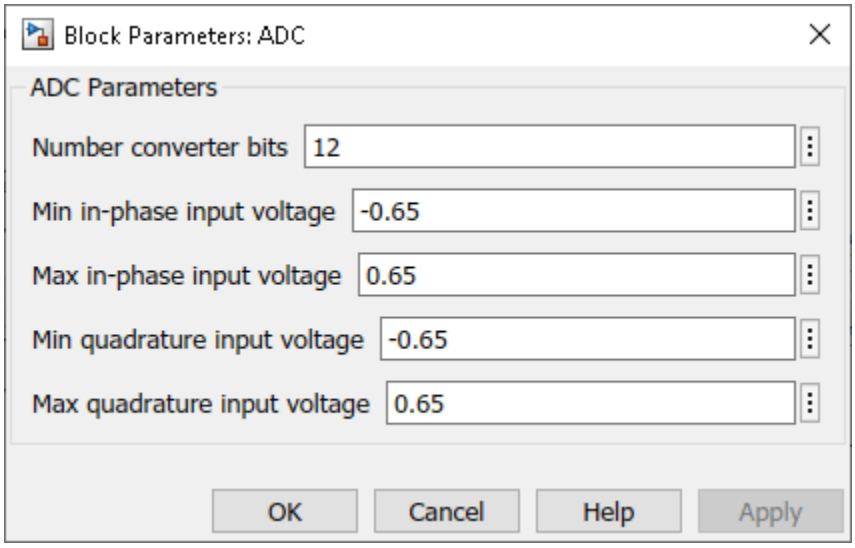


The RF Receiver Subsystem block models typical impairments, including:

- Phase noise as a secondary effect directly related to the thermal noise within the active devices of the oscillator.
- Amplifier nonlinearities due to DC power limitation when the amplifier works in saturation region.
- Impedance mismatch resulting in signal reflection or an inefficient power transfer

Before sending the samples onto the Decode Subframe block, the Output Buffer (after the RF Receiver) buffers all samples within a subframe. At the output of the Output Buffer block, the FIR Decimation and the FIR Rate Conversion blocks downsample the NR waveform back to the original sampling rate.

The ADC Subsystem block models the effect of digitizing the signal. You can modify the parameters of the blocks inside the ADC Subsystem block using its mask.



The use of buffers in the model generates time delays. As the duration of the delay is equivalent to the transmission of a subframe, the Decode Subframe block does not demodulate the first received subframe.

NR Baseband Reception and Measurements

The Decode Subframe block performs OFDM demodulation of the received subframe, channel estimation, and equalization to recover and plot the PDSCH symbols in the Constellation Diagram. This block also averages the EVM over time and frequency and plots these values:

- EVM per OFDM symbol: EVM averaged over each OFDM symbol.
- EVM per slot: EVM averaged over the allocated PDSCH symbols within a slot.
- EVM per subcarrier: EVM averaged over the allocated PDSCH symbols within a subcarrier.
- Overall EVM: EVM averaged over all the allocated PDSCH symbols transmitted.

Annex B and Annex C of TS 38.104 define an alternative method for computing the EVM in FR1 and FR2, respectively. You can enable this method by enabling **Perform 3GPP EVM measurement** parameter of the Multirate Parameters block.

The Spectrum Analyzer block provides frequency-domain measurements such as ACLR (referred to as ACPR) and occupied bandwidth. If you disable the **Perform 3GPP ACLR measurement** parameter of the Multirate Parameters block, you can select the oversampling factor and the Spectrum Analyzer block measures the occupied bandwidth.

The Decode Subframe block discards the first received subframe (1 ms) due to processing delays. Therefore, to receive one frame, you must simulate 11 ms for FDD (10 ms for the frame plus 1 ms for the initially discarded subframe period). If the simulation time is longer than 11 ms, the 5G NR Test Model block cyclically transmits the same NR frame. Similarly, the LTE Test Model block cyclically transmits the same LTE frame.

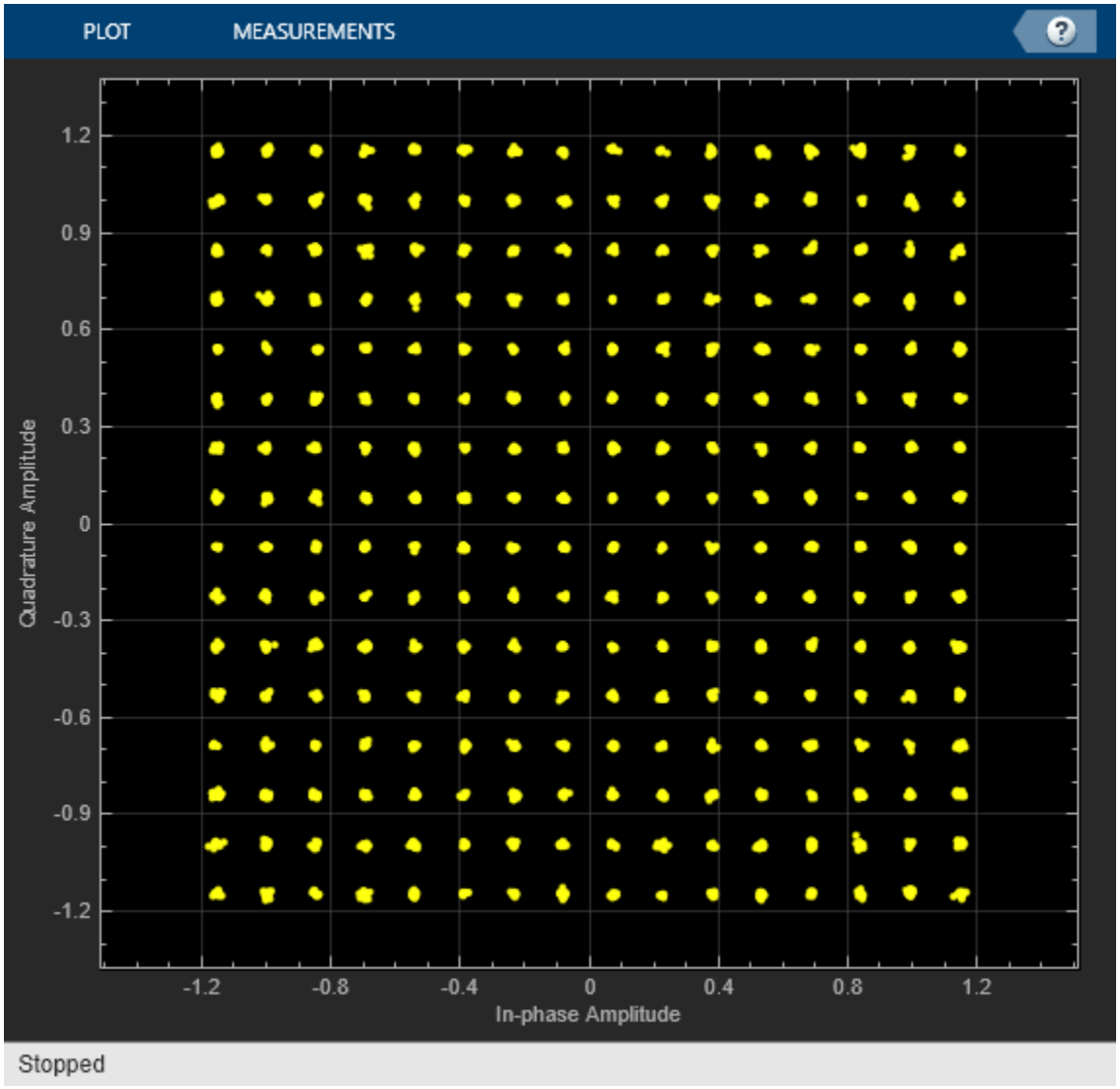
Model Performance

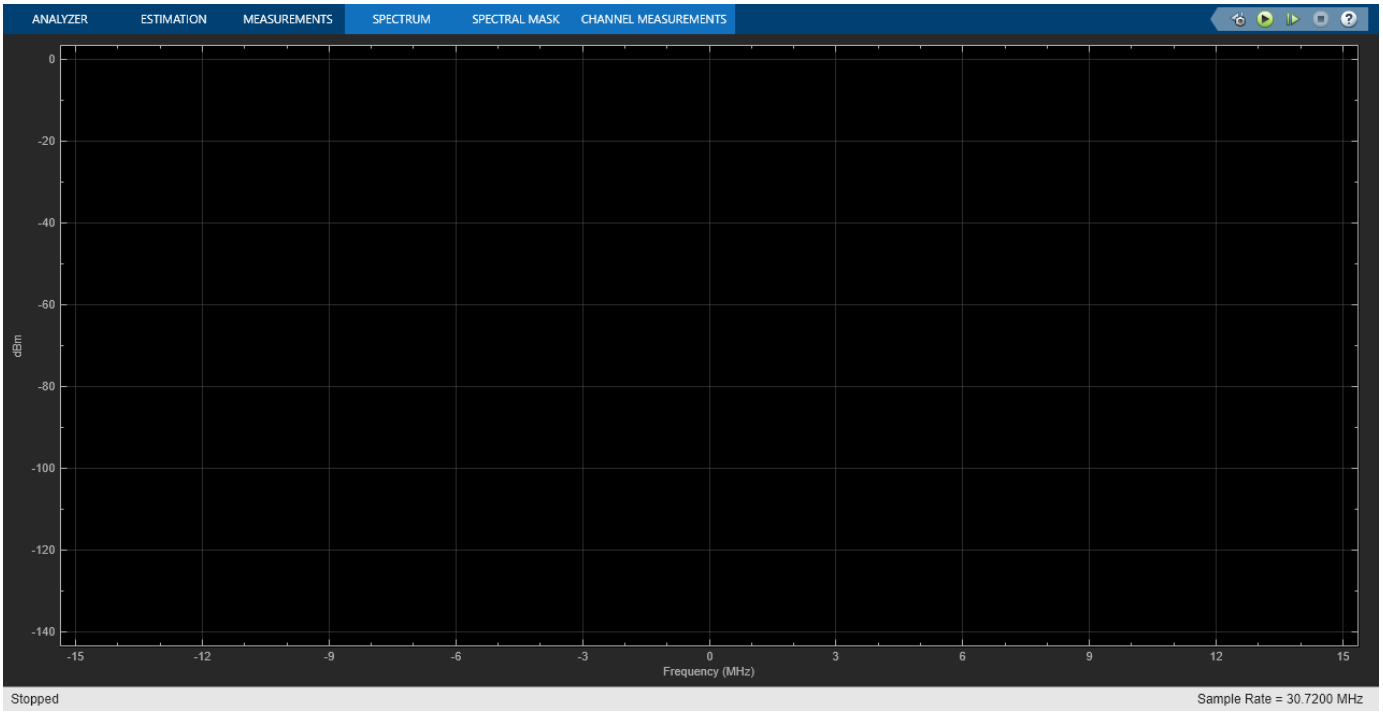
To characterize the impact of the LTE interference on the NR reception you can compare the EVM and ACLR results for two different cases: 1) NR transmission without LTE interference and 2) NR transmission with LTE interference.

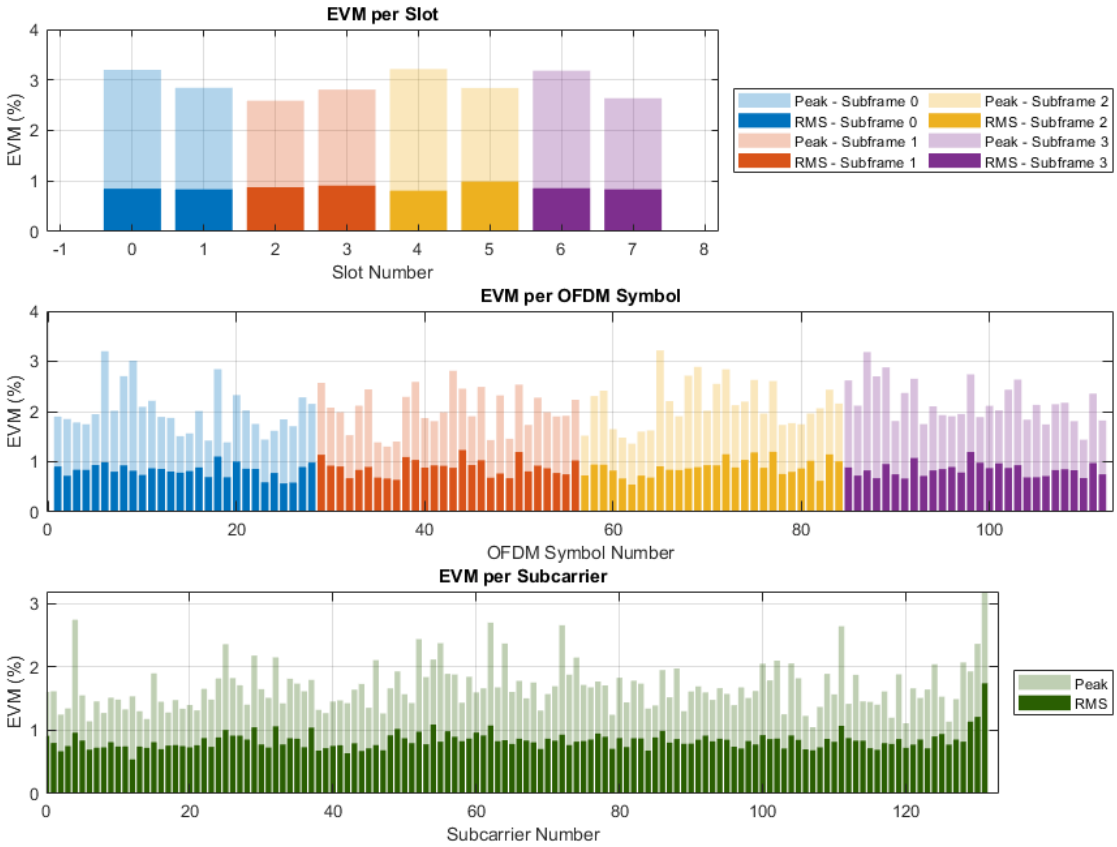
- *Without LTE interference (Interferer gain = 0)*. To eliminate the LTE interference, set the **Gain** parameter of the Interferer Gain block to 0. Run the simulation to capture, for instance, 4 subframes (5 ms). During simulation, the model displays the EVM and ACLR measurements and the constellation diagram.

```
set_param([modelName '/Interferer Gain'],'Gain','0');
sim(modelName);
```

```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 0.849 3.201%
PDSCH RMS EVM, Peak EVM, slot 1: 0.835 2.844%
Averaged overall PDSCH RMS EVM: 0.842%
Overall PDSCH Peak EVM = 3.2014%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 2: 0.878 2.589%
PDSCH RMS EVM, Peak EVM, slot 3: 0.909 2.811%
Averaged overall PDSCH RMS EVM: 0.894%
Overall PDSCH Peak EVM = 2.811%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 4: 0.809 3.218%
PDSCH RMS EVM, Peak EVM, slot 5: 0.989 2.840%
Averaged overall PDSCH RMS EVM: 0.904%
Overall PDSCH Peak EVM = 3.2177%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 6: 0.859 3.185%
PDSCH RMS EVM, Peak EVM, slot 7: 0.835 2.637%
Averaged overall PDSCH RMS EVM: 0.847%
Overall PDSCH Peak EVM = 3.1853%
```


PLOT
MEASUREMENTS
1.2
0.9
0.6
0.3
0
-0.3
-0.6
-0.9
-1.2
Quadrature Amplitude
-1.2
-0.8
-0.4
0
0.4
0.8
1.2
In-phase Amplitude
Stopped


ANALYZER
ESTIMATION
MEASUREMENTS
SPECTRUM
SPECTRAL MASK
CHANNEL MEASUREMENTS
0
-20
-40
-60
-80
-100
-120
-140
dBm
-15
-12
-9
-6
-3
0
3
6
9
12
15
Frequency (MHz)
Stopped
Sample Rate = 30.7200 MHz



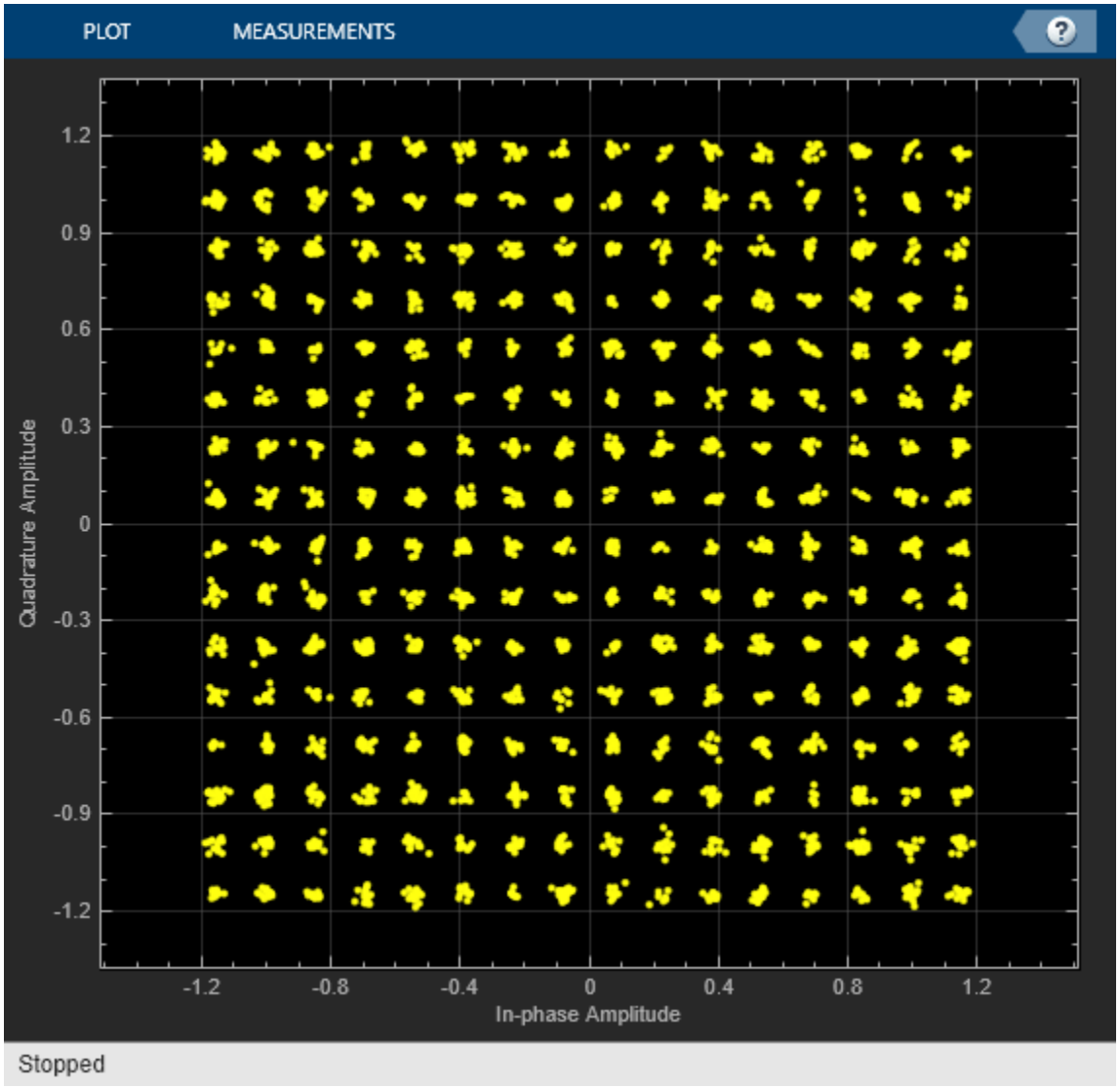
When there is no LTE interference, the ACLR values are around 50 and 87 dB and the overall EVM is around 0.9%.

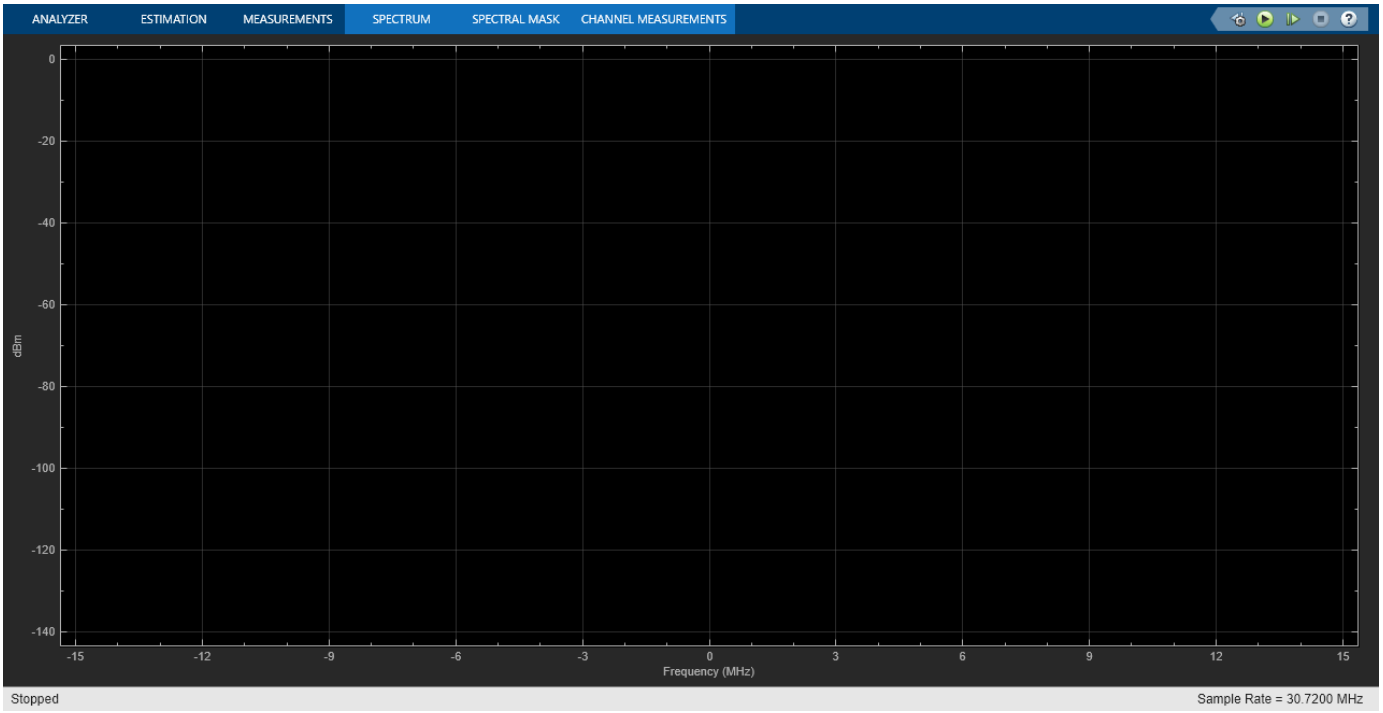
- *With LTE interference (Interferer gain = 1)*. To activate the LTE interference, set the **Gain** parameter of the Interferer Gain block to a value different from 0. For example, choose 1.

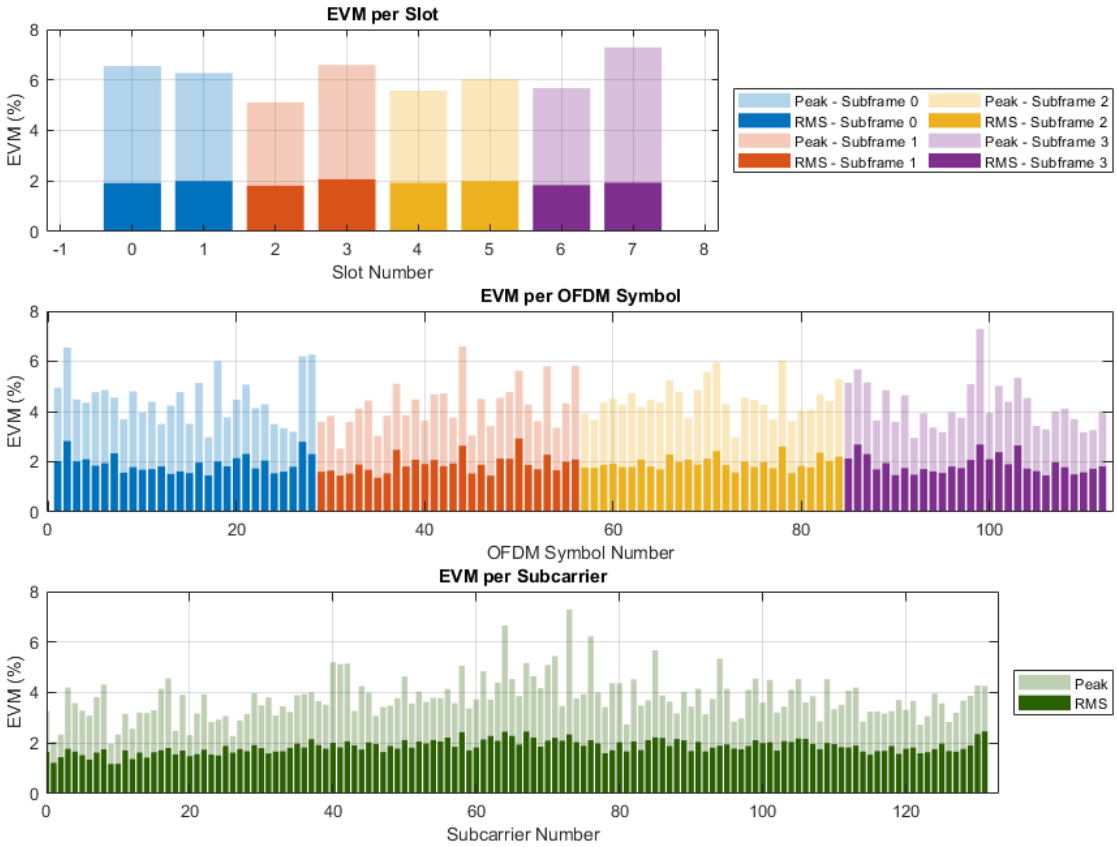
```
set_param([modelName '/Interferer Gain'],'Gain','1');
sim(modelName);
```

```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 1.917 6.547%
PDSCH RMS EVM, Peak EVM, slot 1: 1.999 6.271%
Averaged overall PDSCH RMS EVM: 1.958%
Overall PDSCH Peak EVM = 6.547%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 2: 1.816 5.113%
PDSCH RMS EVM, Peak EVM, slot 3: 2.072 6.591%
Averaged overall PDSCH RMS EVM: 1.948%
Overall PDSCH Peak EVM = 6.591%
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 4: 1.929 5.578%
PDSCH RMS EVM, Peak EVM, slot 5: 1.996 6.033%
Averaged overall PDSCH RMS EVM: 1.963%
Overall PDSCH Peak EVM = 6.0331%
```

```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 6: 1.841 5.680%
PDSCH RMS EVM, Peak EVM, slot 7: 1.934 7.290%
Averaged overall PDSCH RMS EVM: 1.888%
Overall PDSCH Peak EVM = 7.2901%
```




ANALYZER
ESTIMATION
MEASUREMENTS
SPECTRUM
SPECTRAL MASK
CHANNEL MEASUREMENTS
0
-20
-40
-60
-80
-100
-120
-140
dBm
-15
-12
-9
-6
-3
0
3
6
9
12
15
Frequency (MHz)
Stopped
Sample Rate = 30.7200 MHz



Compared to the previous case, the constellation diagram is more distorted and the spectral regrowth is higher. In terms of the measurements, the ACLR values are around 46 and 82 dB and the overall EVM is around 2%.

The RF Receiver is configured to work with the waveform configurations selected in the 5G NR Test Model and the LTE Test Model blocks and with the NR and LTE carriers centered at 2190 MHz and 2120 MHz, respectively. These carriers are within the NR operating band n65, TS 38.101-1, and the E-UTRA operating band 1, TS 36.101.

Summary and Further Exploration

This example demonstrates how to model and test the reception of an NR waveform when coexisting with an LTE waveform. The RF receiver consists of bandpass filters, amplifiers and a demodulator. To evaluate the impact of the LTE interference, the example modifies the gain of the LTE waveform and performs ACLR and EVM measurements. You can explore the impact of altering the RF impairments as well. For example:

- Increase the phase noise by using **Phase noise offset (Hz)** and **Phase noise level (dBc/Hz)** parameters on the **Demodulator** tab of the RF Receiver Subsystem block.
- Reduce the input back-off of the Amplifier block by increasing the **Gain (dB)** parameter on the **LNA** tab of the RF Receiver Subsystem block.

If you modify the carrier frequencies of the RF Receiver Subsystem block or the NR-TM and E-TM configurations, check if you need to update the parameters of the RF Receiver components as these parameters have been selected to work for the current example configuration. For instance, a change in the carrier frequency requires revising the **Passband frequencies** and **Stopband frequencies** parameters of the RF Bandpass Filter block inside the RF Receiver. If you select a bandwidth wider than 20 MHz, check if you need to update the **Impulse response duration** and **Phase noise frequency offset (Hz)** parameters of the Demodulator (RF Blockset) block. The phase noise offset determines the lower limit of the impulse response duration. If the phase noise frequency offset resolution is high for a given impulse response duration, a warning message appears, specifying the minimum duration suitable for the required resolution.

You can use this example as the basis for testing the coexistence between NR-TM and E-TM waveforms for different RF configurations. You can try replacing the RF Receiver Subsystem block by another RF subsystem and then configure the model accordingly.

To use a different NR-TM waveform, open the **5G Waveform Generator** app, select the NR-TM configuration, and export a new block. Similarly, to use a different E-TM waveform, open the **LTE Waveform Generator** app, select the E-TM configuration, and export a new block. For more information on how to generate and use such blocks, see “Generate Wireless Waveform in Simulink Using App-Generated Block”.

References

1. 3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
2. 3GPP TS 36.141 "E-UTRA; Base Station (BS) conformance testing" *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
3. 3GPP TS 38.101-1. "NR; User Equipment (UE) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
4. 3GPP TS 36.101. "E-UTRA; User Equipment (UE) radio transmission and reception." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

See Also

Related Examples

- “Modeling and Testing an NR RF Transmitter” on page 7-45
- “Generate Wireless Waveform in Simulink Using App-Generated Block”

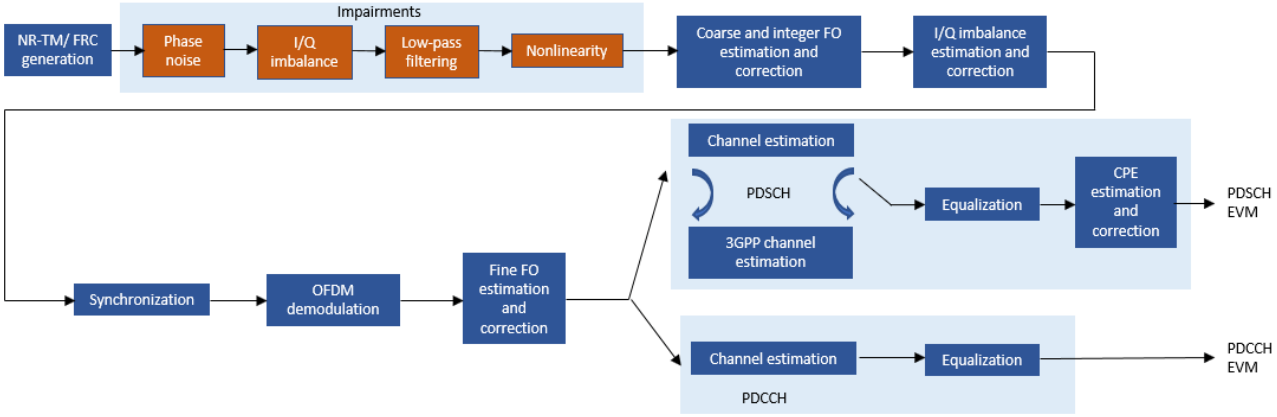
EVM Measurement of 5G NR Downlink Waveforms with RF Impairments

This example shows how to measure the error vector magnitude (EVM) of NR test model (NR-TM) or fixed reference channel (FRC) waveforms. The example also shows how to add RF impairments, including phase noise, in-phase and quadrature (I/Q) imbalance, filter effects, and memoryless nonlinearity.

Introduction

For base station RF testing, the 3GPP 5G NR standard defines a set of NR-TM waveforms. For user equipment (UE) testing, the standard defines a set of FRC waveforms. The NR-TMs and FRCs for frequency range 1 (FR1) are defined in TS 38.141-1 while the NR-TMs and FRCs for frequency range 2 (FR2) are defined in TS 38.141-2.

This example shows how to generate an NR waveform (TM or FRC), add RF impairments, such as, phase noise, I/Q imbalance, filter effects, and memoryless nonlinearities, and calculate the EVM of the resulting signal. The example plots the RMS and peak EVMs per orthogonal frequency division multiplexing (OFDM) symbol, slot, and subcarrier and also calculates the overall EVM (RMS EVM averaged over the complete waveform). Annex B and Annex C of TS 38.104 define an alternative method for computing the EVM in FR1 and FR2, respectively. This figure shows the processing chain implemented in this example.



Simulation parameters

Each NR-TM or FRC waveform is defined by a combination of:

- NR-TM/FRC name
- Channel bandwidth
- Subcarrier spacing
- Duplexing mode

```
% Select one of the Release 15 NR-TMs for FR1 and FR2 among:
% "NR-FR1-TM1.1","NR-FR1-TM1.2","NR-FR1-TM2",
% "NR-FR1-TM2a","NR-FR1-TM3.1","NR-FR1-TM3.1a",
% "NR-FR1-TM3.2","NR-FR1-TM3.3","NR-FR2-TM1.1",
% "NR-FR2-TM2","NR-FR2-TM2a","NR-FR2-TM3.1","NR-FR2-TM3.1a"
```

```
% or
% Select one of the Release 15 FRCs for FR1 and FR2 among:
% "DL-FRC-FR1-QPSK","DL-FRC-FR1-64QAM",
% "DL-FRC-FR1-256QAM","DL-FRC-FR2-QPSK",
% "DL-FRC-FR2-16QAM","DL-FRC-FR2-64QAM"

rc = "NR-FR1-TM3.2"; % Reference channel (NR-TM or FRC)

% Select the NR waveform parameters
bw = "10MHz"; % Channel bandwidth
scs = "30kHz"; % Subcarrier spacing
dm = "FDD"; % Duplexing mode
```

For TMs, the generated waveform may contain more than one physical data shared channel (PDSCH). The chosen PDSCH to analyse is based on the radio network temporary identifier (RNTI). By default, these RNTIs are considered for EVM calculation:

- NR-FR1-TM2: RNTI = 2 (64QAM EVM)
- NR-FR1-TM2a: RNTI = 2 (256QAM EVM)
- NR-FR1-TM3.1: RNTI = 0 and 2 (64QAM EVM)
- NR-FR1-TM3.1a: RNTI = 0 and 2 (256QAM EVM)
- NR-FR1-TM3.2: RNTI = 1 (16QAM EVM)
- NR-FR1-TM3.3: RNTI = 1 (QPSK EVM)
- NR-FR2-TM2: RNTI = 2 (64QAM EVM)
- NR-FR2-TM2a: RNTI = 2 (256QAM EVM)
- NR-FR2-TM3.1: RNTI = 0 and 2 (64QAM EVM)
- NR-FR2-TM3.1a: RNTI = 0 and 2 (256QAM EVM)

As per the specifications (TS 38.141-1, TS 38.141-2), these TMs are not designed to perform EVM measurements: NR-FR1-TM1.1, NR-FR1-TM1.2, NR-FR2-TM1.1. However, if you generate these TMs, the example measures the EVM for the following RNTIs.

- NR-FR1-TM1.1: RNTI = 0 (QPSK EVM)
- NR-FR1-TM1.2: RNTI = 2 (QPSK EVM)
- NR-FR2-TM1.1: RNTI = 0 (QPSK EVM)

For PDSCH FRCs and physical downlink control channel (PDCCH), by default, RNTI 0 is considered for EVM calculation.

The example calculates the PDSCH EVM for the RNTIs listed above. To override the default PDSCH RNTIs, specify the `targetRNTIs` vector

```
targetRNTIs = [];
```

To print EVM statistics, set `displayEVM` to `true`. To disable the prints, set `displayEVM` to `false`. To plot EVM statistics, set `plotEVM` to `true`. To disable the plots, set `plotEVM` to `false`

```
displayEVM = true;
plotEVM = true;
```

```
if displayEVM
    fprintf('Reference Channel = %s\n', rc);
end
```

```
Reference Channel = NR-FR1-TM3.2
```

To measure EVM as defined in TS 38.104, Annex B(FR1) / Annex C(FR2), set `evm3GPP` to `true`. `evm3GPP` is disabled by default. `evm3GPP` is disabled for PDCCH EVM measurement.

```
evm3GPP = false;
```

This example considers the most typical impairments that distort the waveform when passing through an RF transmitter or receiver: phase noise, I/Q imbalance, filter effects and memoryless nonlinearity. Enable or disable impairments by toggling the flags `phaseNoiseOn`, `IQImbalanceON`, `filterOn`, and `nonLinearityModelOn`.

```
phaseNoiseOn = true;
IQImbalanceON = true;
filterOn = true;
nonLinearityModelOn = true;
```

To model wideband filter effects, specify a higher waveform sample rate. You can increase the sample rate by multiplying the nominal sample rate with the oversampling factor, `OSR`. To use the nominal sample rate, set `OSR` to 1.

```
OSR = 5; % oversampling factor

% Create waveform generator object
tmwavegen = hNRReferenceWaveformGenerator(rc,bw,scs,dm);

% Waveform bandwidth
bandwidth = tmwavegen.Config.ChannelBandwidth*1e6;

if OSR > 1
    % The |Config| property in |tmwavegen| specifies the configuration of
    % the standard-defined reference waveform. It is a read-only property.
    % To customize the waveform, make the |Config| property writable.
    tmwavegen = makeConfigWritable(tmwavegen);

    % Increase the waveform sample rate by multiplying the nominal sample
    % rate with |OSR|
    nominalSampleRate = getNominalSampleRate(tmwavegen.Config);
    tmwavegen.Config.SampleRate = nominalSampleRate*OSR;
else
    filterOn = false;
end

% Generate the waveform and get the waveform sample rate
[txWaveform,tmwaveinfo,resourcesinfo] = generateWaveform(tmwavegen,tmwavegen.Config.NumSubframes)
sr = tmwaveinfo.Info.SamplingRate; % waveform sample rate
```

Normalize the waveform to fit the dynamic range of the nonlinearity.

```
txWaveform = txWaveform/max(abs(txWaveform),[],'all');
```

The waveform consists of one frame for frequency division duplex (FDD) and two for time division duplex (TDD). Repeat the signal twice. Remove the first half of the resulting waveform to avoid the transient introduced by the phase noise model.

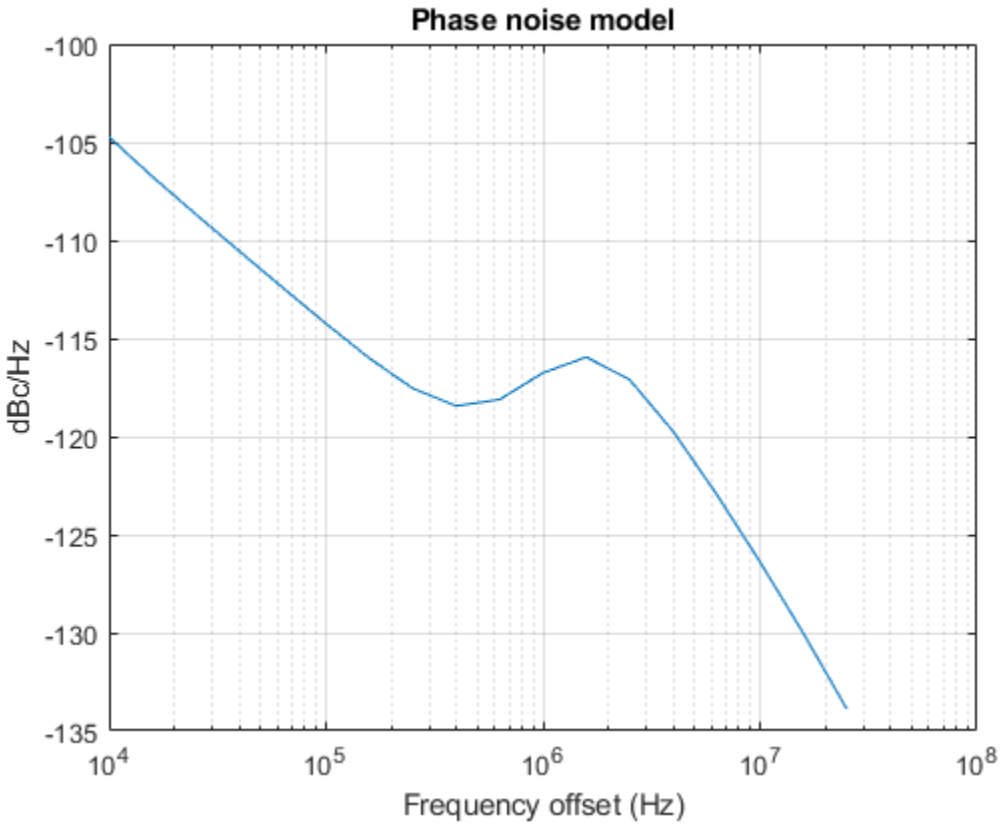
```
txWaveform = repmat(txWaveform,2,1);
```

RF Impairments

This section shows how to model these RF impairments: phase noise, I/Q imbalance, filter effects, and nonlinearity.

Introduce phase noise distortion. The figure shows the phase noise characteristic. The carrier frequency that is considered in the example depends on the frequency range. This example uses the center frequency values of 4 GHz and 30 GHz for FR1 and FR2, respectively. The phase noise characteristic is generated with the pole/zero model described in R1-163984, "Discussion on phase noise modeling".

```
if phaseNoiseOn
    % Carrier frequency
    if tmwavegen.Config.FrequencyRange == "FR1" % carrier frequency for FR1
        fc = 4e9;
    else % carrier frequency for FR2
        fc = 30e9;
    end

    % Calculate the phase noise level
    foffsetLog = (4:0.2:log10(sr/2)-0.001); % model offset from 10e3Hz to
                                            % almost sr/2. To avoid
                                            % aliasing, the sample rate
                                            % must be greater than twice
                                            % the largest value specified
                                            % by FrequencyOffset
    foffset = 10.^foffsetLog;    % linear frequency offset
    PN_dBc_Hz = hPhaseNoisePoleZeroModel(foffset,fc,'C');
    figure; semilogx(foffset,PN_dBc_Hz);
    xlabel('Frequency offset (Hz)');
    ylabel('dBc/Hz');
    title('Phase noise model'); grid on

    % Apply phase noise to the waveform
    pnoise = comm.PhaseNoise('FrequencyOffset',foffset,'Level',PN_dBc_Hz,'SampleRate',sr);
    pnoise.RandomStream = "mt19937ar with seed";
    rxWaveform = pnoise(txWaveform);
    release(pnoise);
else
    rxWaveform = txWaveform; %#ok<UNRCH>
end
```



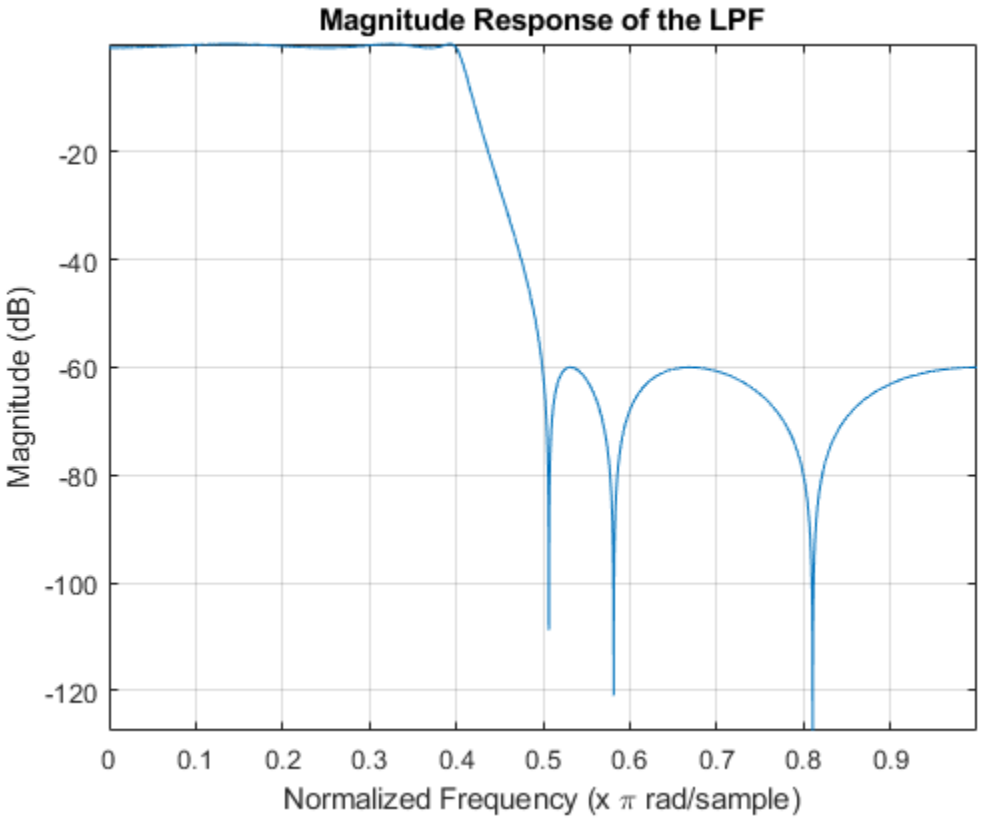
Introduce I/Q imbalance. Apply a 0.2 dB amplitude imbalance and a 0.5 degree phase imbalance to the waveform. You can also increase the amplitude and phase imbalances by setting `amplitudeImbalance` and `phaseImbalance` to higher values.

```
if IQImbalanceON
    amplitudeImbalance = 0.2;
    phaseImbalance = 0.5;
    rxWaveform = iqimbal(rxWaveform,amplitudeImbalance,phaseImbalance);
end
```

Introduce filter effects. To filter the baseband waveform, use a low-pass filter. If the use of the current passband and stopband frequencies, `PassbandFrequency` and `StopbandFrequency`, results in high EVM values for a certain waveform bandwidth and `OSR`, use a wider filter by increasing `PassbandFrequency` and `StopbandFrequency`. To use a narrower filter, reduce `PassbandFrequency` and `StopbandFrequency`. You can also modify the passband ripple and the stopband attenuation. This figure shows the magnitude response of the low-pass filter.

```
if filterOn
    % Create low-pass filter object
    LPF = dsp.LowpassFilter('SampleRate',sr, ...
                            'FilterType','IIR', ...
                            'PassbandFrequency',sr/2-(sr/2*0.6), ...
                            'StopbandFrequency',sr/2-(sr/2*0.5), ...
                            'PassbandRipple',0.7, ...
                            'StopbandAttenuation',60);

    % Plot the magnitude response of the low-pass filter
```

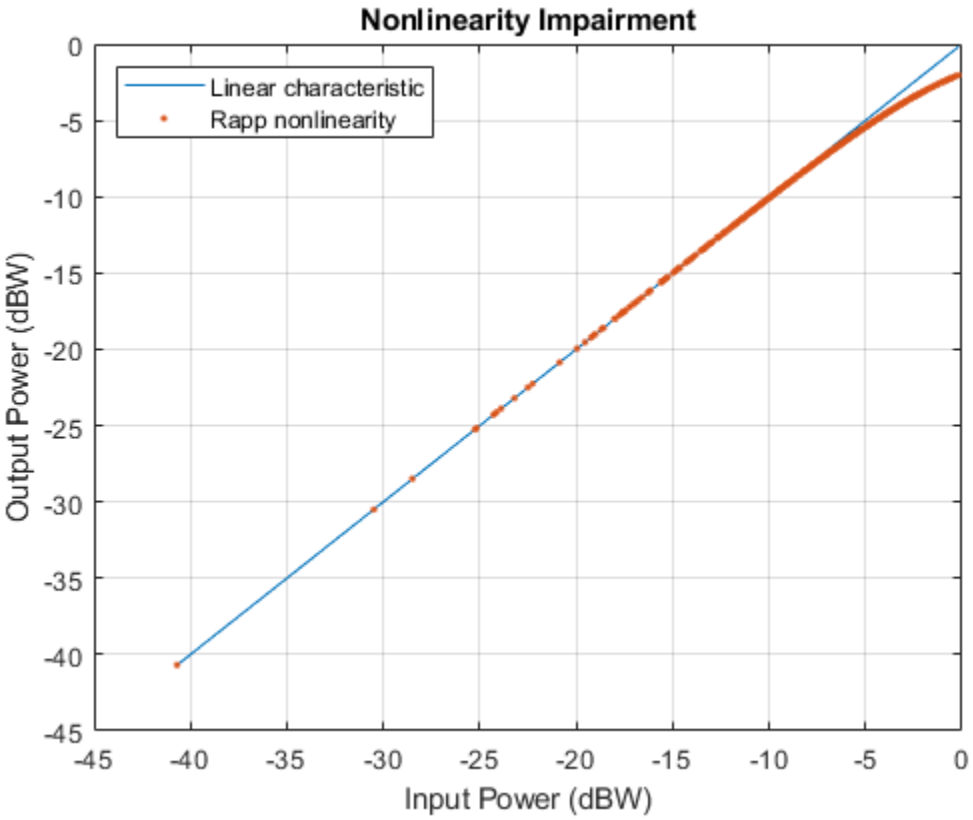
```
    [h,w] = freqz(LPF);
    figure
    plot(w/pi,mag2db(abs(h)));
    axis('tight');
    grid;
    title('Magnitude Response of the LPF')
    xlabel('Normalized Frequency (x \pi rad/sample)');
    ylabel('Magnitude (dB)');

    % Filter the waveform
    rxWaveform = LPF(rxWaveform);
    release(LPF);
end
```



Introduce nonlinear distortion. For this example, use the Rapp model. This figure shows the nonlinearity that is introduced by Rapp model. Set the parameters for the Rapp model to match the characteristics of the memoryless model from TR 38.803 Annex A.1.

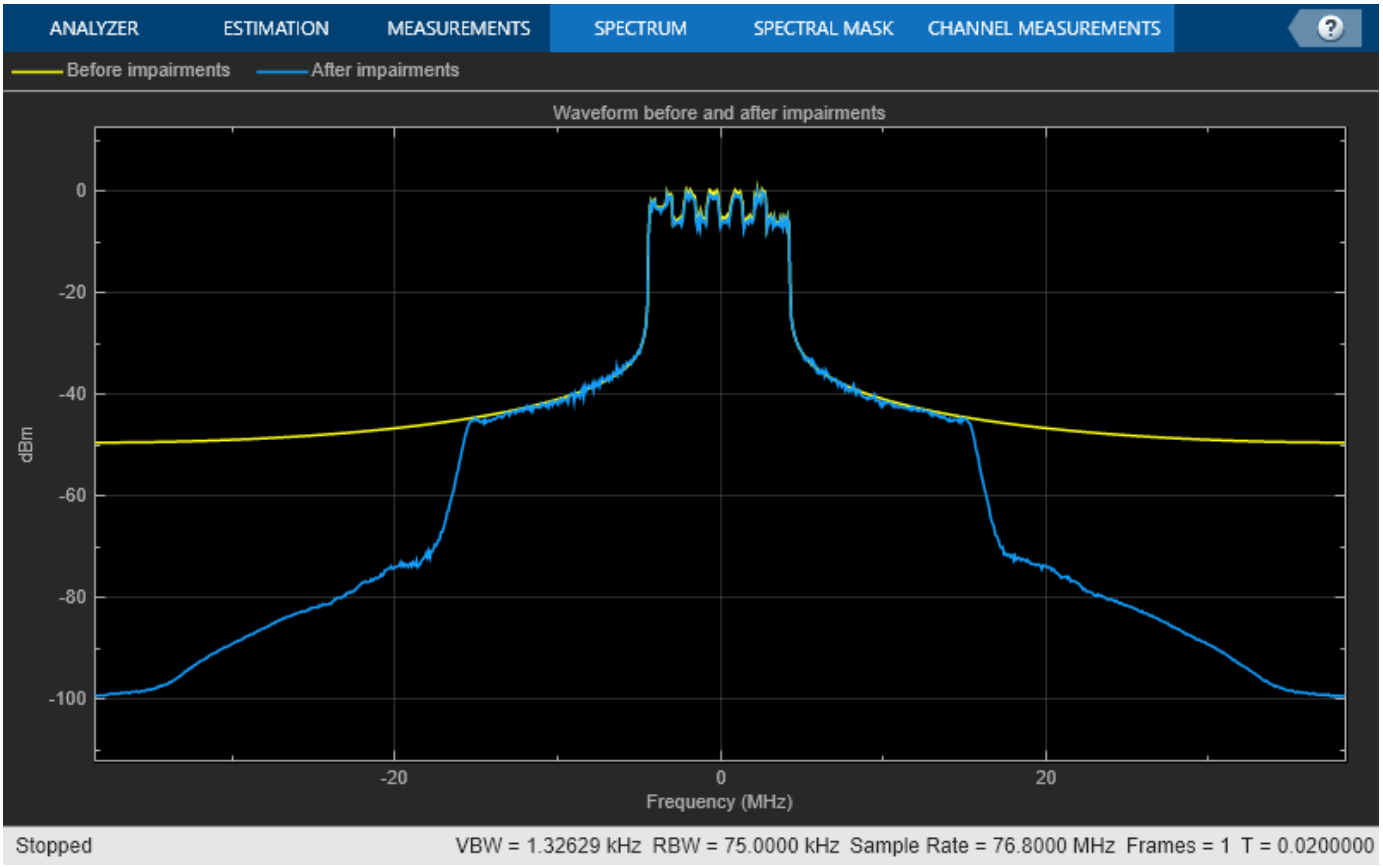
```
if nonLinearityModelOn
    % Generate Rapp model object
    rapp = comm.MemorylessNonlinearity('Method','Rapp model');
    rapp.Smoothness = 1.55;
    rapp.OutputSaturationLevel = 1;

    % Plot nonlinear characteristic
    plotNonLinearCharacteristic(rapp);
```

```
    % Apply nonlinearity
    rxWaveform = rapp(rxWaveform);
    release(rapp);
end
```



Plot the spectrum of the waveform before and after adding the RF impairments

```
scope = spectrumAnalyzer('SampleRate',sr,...
    'ChannelNames',{'Before impairments','After impairments'},...
    'Title', 'Waveform before and after impairments');
scope([txWaveform,rxWaveform]);
release(scope);
```



The signal was previously repeated twice. Remove the first half of this signal. This avoids any transient introduced by the impairment models.

```
if dm == "FDD"
    nFrames = 1;
else % TDD
    nFrames = 2;
end

rxWaveform(1:nFrames*tmwaveinfo.Info.SamplesPerSubframe*10,:) = [];
```

Measurements

The function, hNRDownlinkEVM, performs these steps to decode and analyze the waveform:

- Coarse frequency offset estimation and correction
- Integer frequency offset estimation and correction
- I/Q imbalance estimation and correction
- Synchronization using the demodulation reference signal (DM-RS) over one frame for FDD (two frames for TDD)
- OFDM demodulation of the received waveform
- Fine frequency offset estimation and correction
- Channel estimation

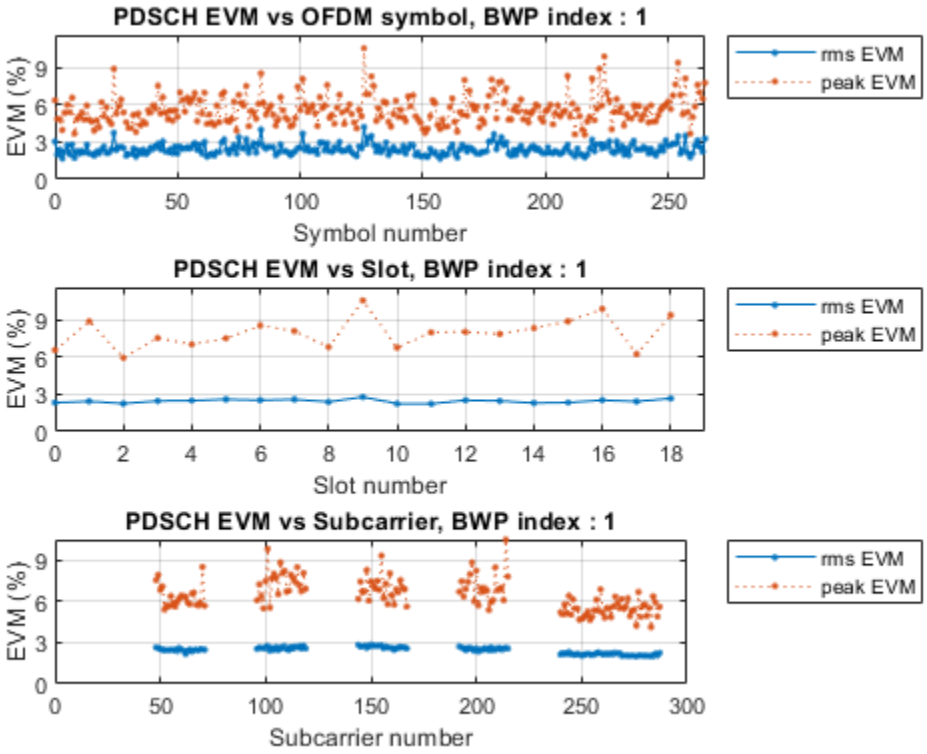
- Equalization
- Common Phase error (CPE) estimation and compensation
- PDSCH EVM computation (enable the switch `evm3GPP`, to process according to the EVM measurement requirements specified in TS 38.104, Annex B (FR1) / Annex C (FR2)).
- PDCCH EVM computation

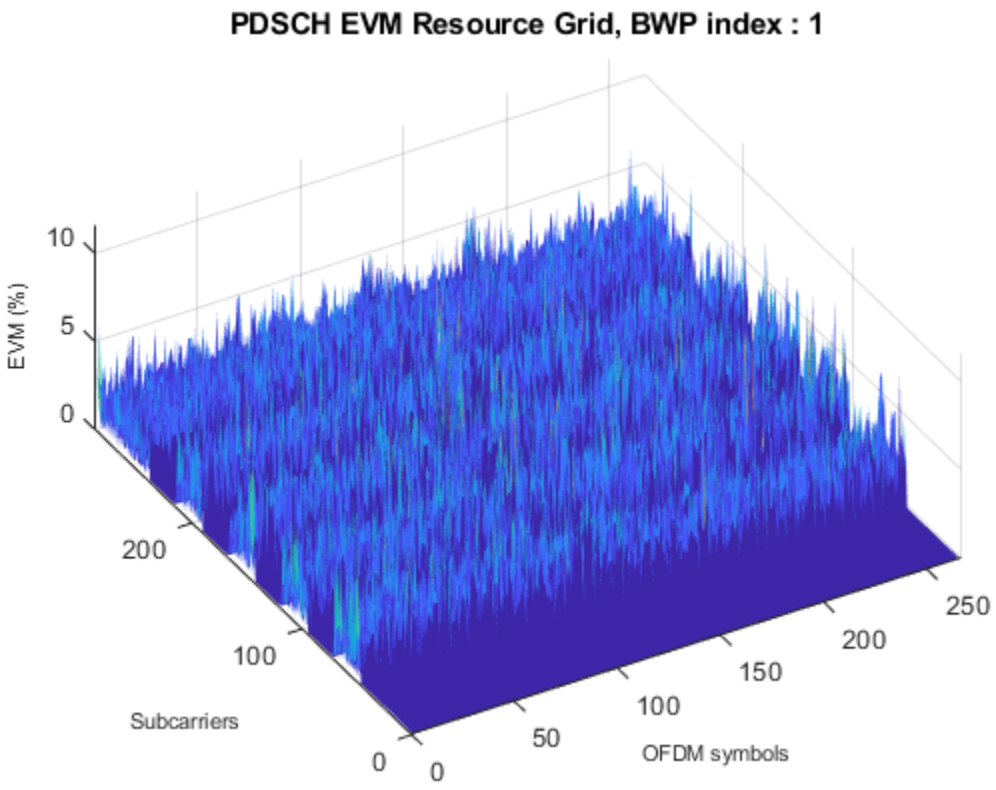
The example measures and outputs various EVM related statistics per symbol, per slot, and per frame peak EVM and RMS EVM. The example displays EVM for each slot and frame. It also displays the overall EVM averaged over the entire input waveform. The example produces a number of plots: EVM vs per OFDM symbol, slot, subcarrier, and overall EVM. Each plot displays the peak vs RMS EVM.

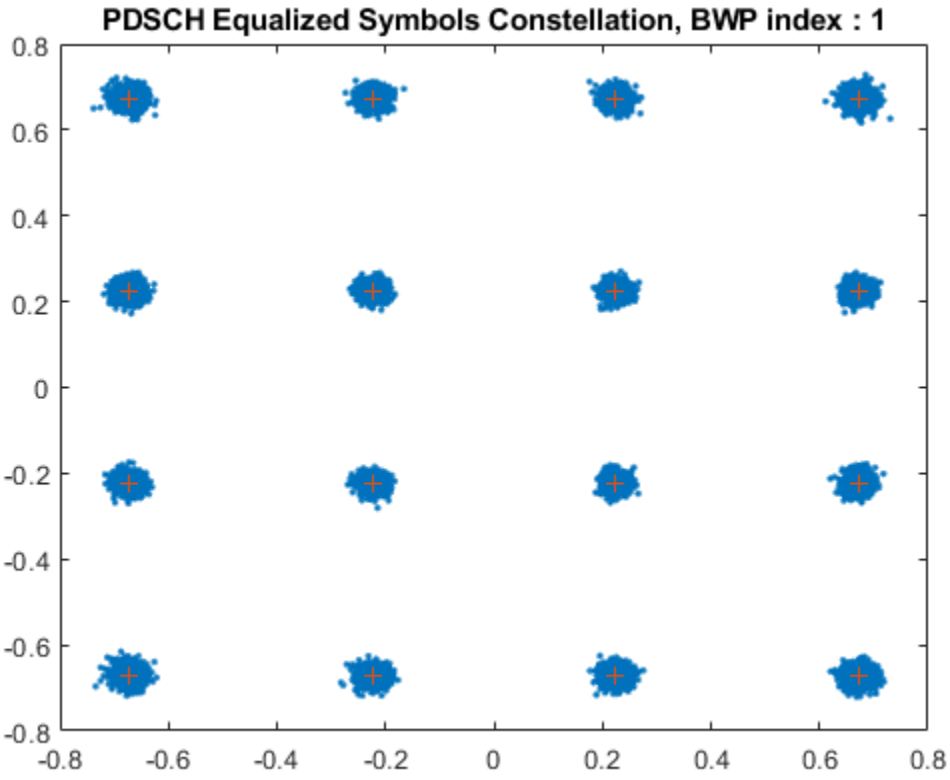
```
cfg = struct();
cfg.Evm3GPP = evm3GPP;
cfg.TargetRNTIs = targetRNTIs;
cfg.PlotEVM = plotEVM;
cfg.DisplayEVM = displayEVM;
cfg.IQImbalance = IQImbalanceON;

% Compute and display EVM measurements
[evmInfo,eqSym,refSym] = hNRDownlinkEVM(tmwavegen.Config,rxWaveform,cfg);
```

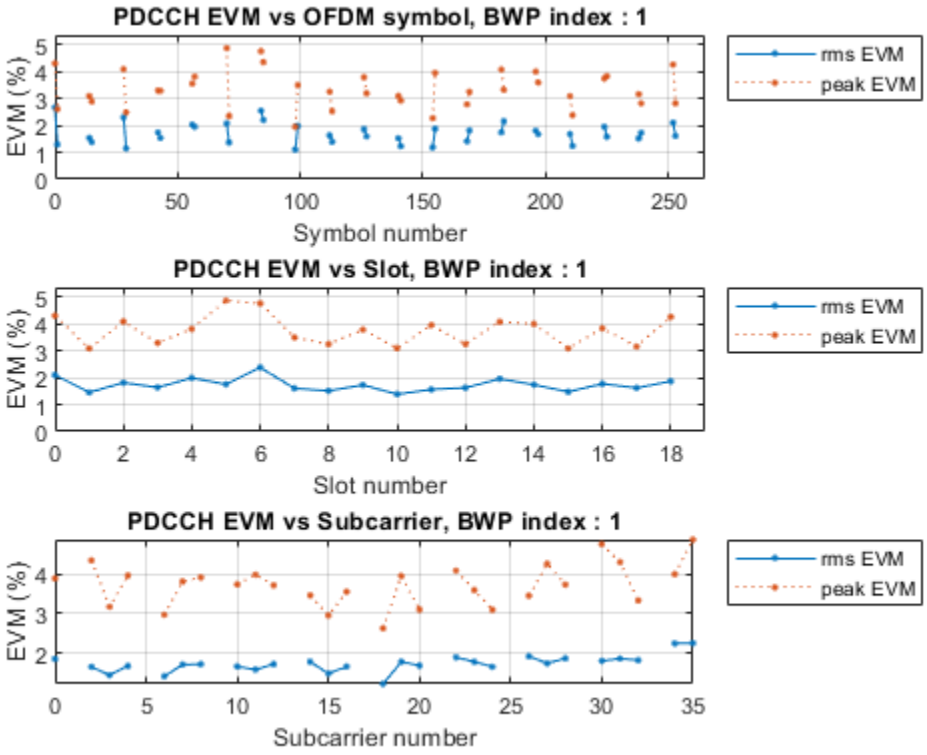
```
EVM stats for BWP idx : 1
PDSCH RMS EVM, Peak EVM, slot 0: 2.319 6.537%
PDSCH RMS EVM, Peak EVM, slot 1: 2.428 8.852%
PDSCH RMS EVM, Peak EVM, slot 2: 2.245 5.909%
PDSCH RMS EVM, Peak EVM, slot 3: 2.454 7.507%
PDSCH RMS EVM, Peak EVM, slot 4: 2.483 6.999%
PDSCH RMS EVM, Peak EVM, slot 5: 2.585 7.484%
PDSCH RMS EVM, Peak EVM, slot 6: 2.519 8.509%
PDSCH RMS EVM, Peak EVM, slot 7: 2.583 8.062%
PDSCH RMS EVM, Peak EVM, slot 8: 2.380 6.781%
PDSCH RMS EVM, Peak EVM, slot 9: 2.774 10.522%
PDSCH RMS EVM, Peak EVM, slot 10: 2.240 6.740%
PDSCH RMS EVM, Peak EVM, slot 11: 2.228 7.963%
PDSCH RMS EVM, Peak EVM, slot 12: 2.511 8.002%
PDSCH RMS EVM, Peak EVM, slot 13: 2.459 7.831%
PDSCH RMS EVM, Peak EVM, slot 14: 2.287 8.306%
PDSCH RMS EVM, Peak EVM, slot 15: 2.328 8.857%
PDSCH RMS EVM, Peak EVM, slot 16: 2.520 9.851%
PDSCH RMS EVM, Peak EVM, slot 17: 2.412 6.223%
PDSCH RMS EVM, Peak EVM, slot 18: 2.660 9.351%
PDCCH RMS EVM, Peak EVM, slot 0: 2.095 4.296%
PDCCH RMS EVM, Peak EVM, slot 1: 1.452 3.086%
PDCCH RMS EVM, Peak EVM, slot 2: 1.813 4.080%
PDCCH RMS EVM, Peak EVM, slot 3: 1.637 3.289%
PDCCH RMS EVM, Peak EVM, slot 4: 1.987 3.807%
PDCCH RMS EVM, Peak EVM, slot 5: 1.752 4.863%
PDCCH RMS EVM, Peak EVM, slot 6: 2.375 4.750%
PDCCH RMS EVM, Peak EVM, slot 7: 1.600 3.493%
PDCCH RMS EVM, Peak EVM, slot 8: 1.512 3.245%
PDCCH RMS EVM, Peak EVM, slot 9: 1.729 3.779%
PDCCH RMS EVM, Peak EVM, slot 10: 1.384 3.092%
PDCCH RMS EVM, Peak EVM, slot 11: 1.562 3.945%
PDCCH RMS EVM, Peak EVM, slot 12: 1.624 3.245%
PDCCH RMS EVM, Peak EVM, slot 13: 1.956 4.071%
PDCCH RMS EVM, Peak EVM, slot 14: 1.740 3.993%
```

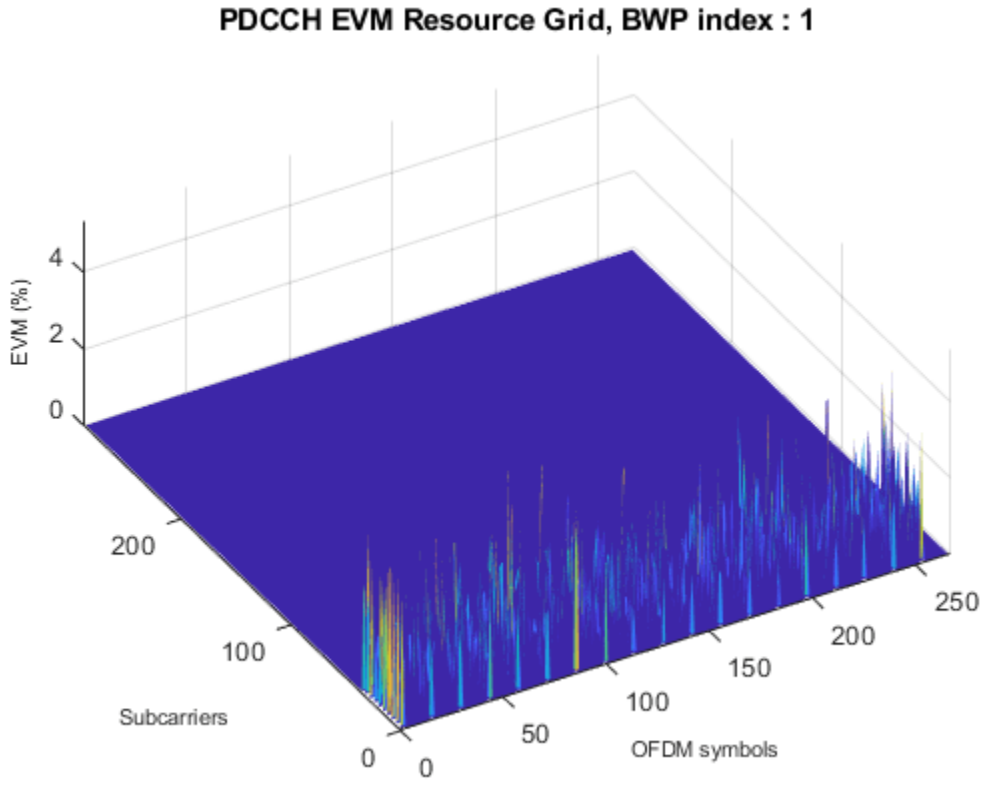
```
PDCCH RMS EVM, Peak EVM, slot 15: 1.474 3.085%
PDCCH RMS EVM, Peak EVM, slot 16: 1.772 3.829%
PDCCH RMS EVM, Peak EVM, slot 17: 1.619 3.151%
PDCCH RMS EVM, Peak EVM, slot 18: 1.867 4.254%
Averaged overall PDSCH RMS EVM: 2.447%
Overall PDSCH Peak EVM = 10.5224%
Averaged overall PDCCH RMS EVM: 1.750%
Overall PDCCH Peak EVM = 4.8629%
```

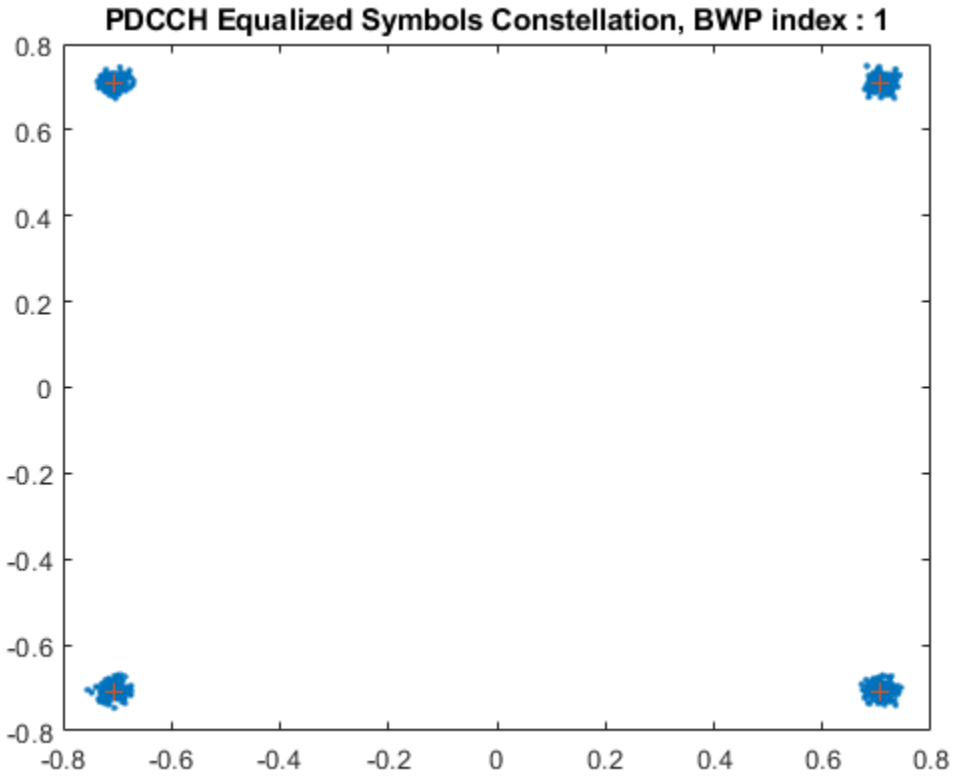



PDSCH EVM Resource Grid, BWP index : 1
EVM (%)
10
5
0
200
100
Subcarriers
0
0
50
100
150
200
250
OFDM symbols


PDSCH Equalized Symbols Constellation, BWP index : 1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


PDCCH EVM vs OFDM symbol, BWP index : 1
EVM (%)
Symbol number
rms EVM
peak EVM
PDCCH EVM vs Slot, BWP index : 1
EVM (%)
Slot number
rms EVM
peak EVM
PDCCH EVM vs Subcarrier, BWP index : 1
EVM (%)
Subcarrier number
rms EVM
peak EVM


PDCCH EVM Resource Grid, BWP index : 1
EVM (%)
4
2
0
200
100
0
Subcarriers
0
50
100
150
200
250
OFDM symbols



Local functions

```
function plotNonLinearCharacteristic(memoryLessNonlinearity)
    % Plot the nonlinear characteristic of the power amplifier (PA) impairment
    % represented by the input parameter memoryLessNonlinearity, which is a
    % comm.MemorylessNonlinearity Communications Toolbox(TM) System object.

    % Input samples
    x = complex((1/sqrt(2))*(-1+2*rand(1000,1)),(1/sqrt(2))*(-1+2*rand(1000,1)));

    % Nonlinearity
    yRapp = memoryLessNonlinearity(x);

    % Release object to feed it a different number of samples
    release(memoryLessNonlinearity);

    % Plot characteristic
    figure;
    plot(10*log10(abs(x).^2),10*log10(abs(x).^2));
    hold on;
    grid on
    plot(10*log10(abs(x).^2),10*log10(abs(yRapp).^2),'.');
    xlabel('Input Power (dBW)');
    ylabel('Output Power (dBW)');
    title('Nonlinearity Impairment')
    legend('Linear characteristic', 'Rapp nonlinearity','Location','Northwest');
end
```

```
function [sampleRate] = getNominalSampleRate(cfgObj)
    % Obtain nominal sample rate for the input parameter CFGOBJ. CFGOBJ is
    % an object of type 'nrDLCarrierConfig'
    sampleRate = nr5g.internal.wavegen.maxSampleRate(cfgObj);
end
```

See Also

Related Examples

- “5G NR-TM and FRC Waveform Generation” on page 7-12
- “NR Phase Noise Modeling and Compensation” on page 4-17

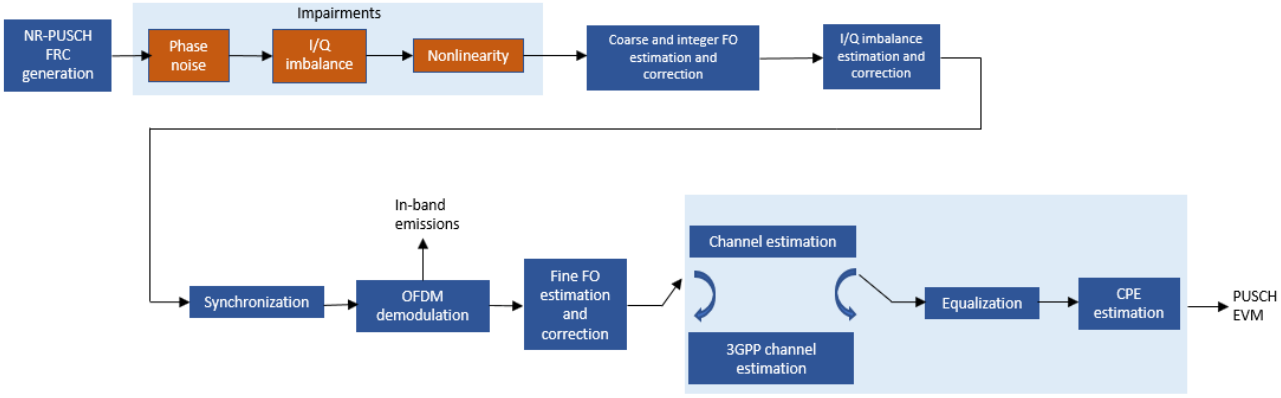
EVM Measurement of 5G NR PUSCH Waveforms

This example shows how to measure the error vector magnitude (EVM) of PUSCH fixed reference channel (FRC) waveforms. The example also shows how to add RF impairments, including in-phase and quadrature (I/Q) imbalance, phase noise and memoryless nonlinearity.

Introduction

For base station receiver RF testing, the 3GPP 5G NR standard defines a set of FRC waveforms. The FRCs for frequency range 1 (FR1) and frequency range 2 (FR2) are defined in TS 38.104, Annex A.

This example shows how to generate an NR waveform (FRC) and add impairments. The example uses in-phase and quadrature (I/Q) imbalance, phase noise and memoryless nonlinearities. It shows how to calculate the EVM of the resulting signal, and then plot the root mean square (RMS) and peak EVMs per orthogonal frequency division multiplexing (OFDM) symbol, slot, and subcarrier. Then calculate the overall EVM (RMS EVM averaged over the complete waveform). Annex F of TS 38.101-1 and TS 38.101-2 define an alternative method for computing the EVM in FR1 and FR2, respectively. The figure below shows the processing chain implemented in this example.



Simulation parameters

Each FRC waveform is defined by a combination of these parameters:

- Frequency range number
- Order of the modulation and coding scheme (MCS) used
- Index of the FRC for the given MCS

Select one of the Release 15 FRCs for FR1 and FR2.

The selected reference channel (rc) must follow a G-FR *X* -A *Y* - *Z* format, where *X* is the frequency-range number (1 or 2), *Y* is the order of the MCS used in Annex A of TS 38.104 (i.e., a value from 1 to 5) and *Z* is the index of the FRC for the given MCS. The range of valid *Z* values depends on the frequency range and the MCS (*X* and *Y*) In case the input waveform is not a FRC waveform, atmost one layer is supported for EVM measurement.

```
rc = "G-FR1-A1-7"; % FRC
```

To print EVM statistics, set `displayEVM` to `true`. To disable the prints, set `displayEVM` to `false`. To plot EVM statistics, set `plotEVM` to `true`. To disable the plots, set `plotEVM` to `false`

```
displayEVM = true;
plotEVM = true;

if displayEVM
    fprintf('Reference Channel = %s\n',rc);
end
```

```
Reference Channel = G-FR1-A1-7
```

To measure EVM as defined in TS 38.101-1 (FR1) or TS 38.101-2 (FR2), Annex F respectively, set `evm3GPP` to `true`. `evm3GPP` is disabled by default.

```
evm3GPP = false;
```

Create a waveform generator object,then and generate the waveform.

```
wavegen = hNRReferenceWaveformGenerator(rc);
[txWaveform,tmwaveinfo,resourcesinfo] = generateWaveform(wavegen,wavegen.Config.NumSubframes);
```

Impairment: In-phase and Quadrature (I/Q) Imbalance, Phase Noise and Memoryless Nonlinearity

This example considers the most typical impairments that distort the waveform when passing through an RF transmitter or receiver: phase noise, I/Q imbalance, and memoryless nonlinearity. Enable or disable impairments by toggling the flags `phaseNoiseOn`, `IQImbalanceON`, and `nonLinearityModelOn`.

```
phaseNoiseOn = true;
IQImbalanceON = true;
nonLinearityModelOn = true;
```

Normalize the waveform to fit the dynamic range of the nonlinearity.

```
txWaveform = txWaveform/max(abs(txWaveform),[],'all');
```

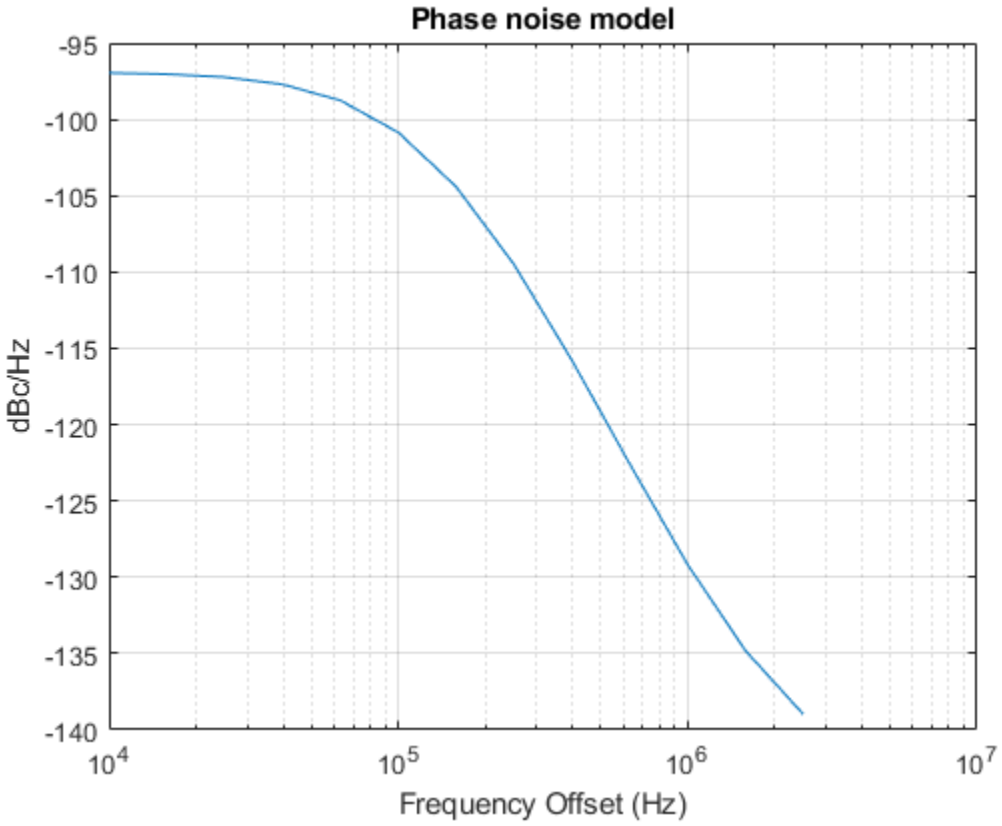
The waveform consists of one frame for frequency division duplexing (FDD) and two for time division duplexing (TDD). Repeat the signal twice. For this example, remove the first half of the resulting waveform to avoid the transient introduced by the phase noise model.

```
txWaveform = repmat(txWaveform,2,1);
```

Introduce phase noise distortion. The figure shows the phase noise characteristic. The carrier frequency considered depends on the frequency range. We use values of 4 GHz and 28 GHz for FR1 and FR2, respectively. The phase noise characteristic is generated with the pole or zero model described in R1-163984, "Discussion on phase noise modeling".

```
if phaseNoiseOn
    sr = tmwaveinfo.Info.SamplingRate;

    % Carrier frequency
    if wavegen.Config.FrequencyRange == "FR1" % Carrier frequency for FR1
        fc = 4e9;
    else % Carrier frequency for FR2
        fc = 28e9;
    end

    % Calculate the phase noise level.
    foffsetLog = (4:0.2:log10(sr/2)); % Model offset from 1e4Hz to sr/2
```

```
    foffset = 10.^foffsetLog;    % Linear frequency offset
    PN_dBc_Hz = hPhaseNoisePoleZeroModel(foffset,fc,'A');
    figure;
    semilogx(foffset,PN_dBc_Hz);
    xlabel('Frequency Offset (Hz)');
    ylabel('dBc/Hz');
    title('Phase noise model'); grid on

    % Apply phase noise to waveform
    pnoise = comm.PhaseNoise('FrequencyOffset',foffset,'Level',PN_dBc_Hz,'SampleRate',sr);
    rxWaveform = zeros(size(txWaveform),'like',txWaveform);
    for i = 1:size(txWaveform,2)
        rxWaveform(:,i) = pnoise(txWaveform(:,i));
        release(pnoise)
    end
else
    rxWaveform = txWaveform; %#ok<UNRCH>
end
```



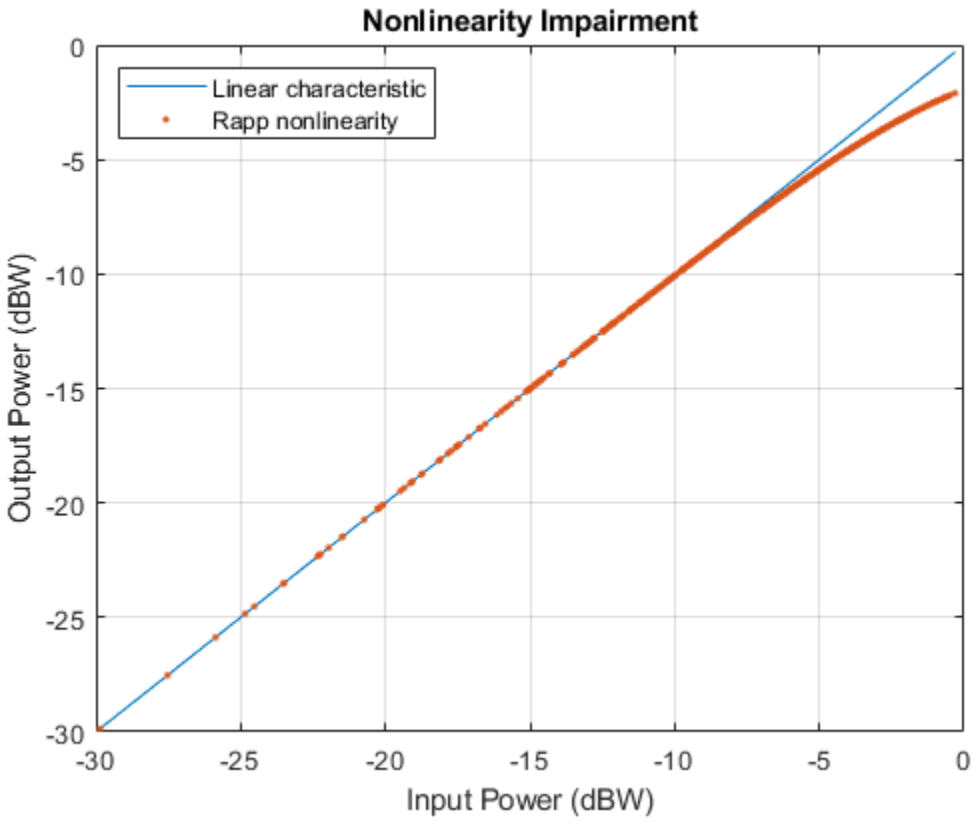
Introduce I/Q imbalance. Apply a 0.2 dB amplitude imbalance and a 0.5 degree phase imbalance to the waveform. You can also increase the amplitude and phase imbalances by setting `amplitudeImbalance` and `phaseImbalance` to higher values.

```
if IQImbalanceON
    amplitudeImbalance = 0.2;
    phaseImbalance = 0.5;
```

```
    rxWaveform = iqimbal(rxWaveform,amplitudeImbalance,phaseImbalance);
end
```

Introduce nonlinear distortion. For this example, use the Rapp model. The figure shows the introduced nonlinearity. Set the parameters for the Rapp model to match the characteristics of the memoryless model from TR 38.803 "Memoryless polynomial model - Annex A.1".

```
if nonLinearityModelOn
    rapp = comm.MemorylessNonlinearity('Method','Rapp model');
    rapp.Smoothness = 1.55;
    rapp.OutputSaturationLevel = 1;

    % Plot nonlinear characteristic
     plotNonLinearCharacteristic(rapp);

    % Apply nonlinearity
    for i = 1:size(rxWaveform,2)
        rxWaveform(:,i) = rapp(rxWaveform(:,i));
        release(rapp)
    end
end
```



The signal was previously repeated twice. Remove the first half of this signal to avoid any transient introduced by the impairment models.

```
dm = wavegen.ConfiguredModel{4};
if dm == "FDD"
    nFrames = 1;
```

```
else % TDD
    nFrames = 2;
end
rxWaveform(1:nFrames*tmwaveinfo.Info.SamplesPerSubframe*10,:) = [];
```

Measurements

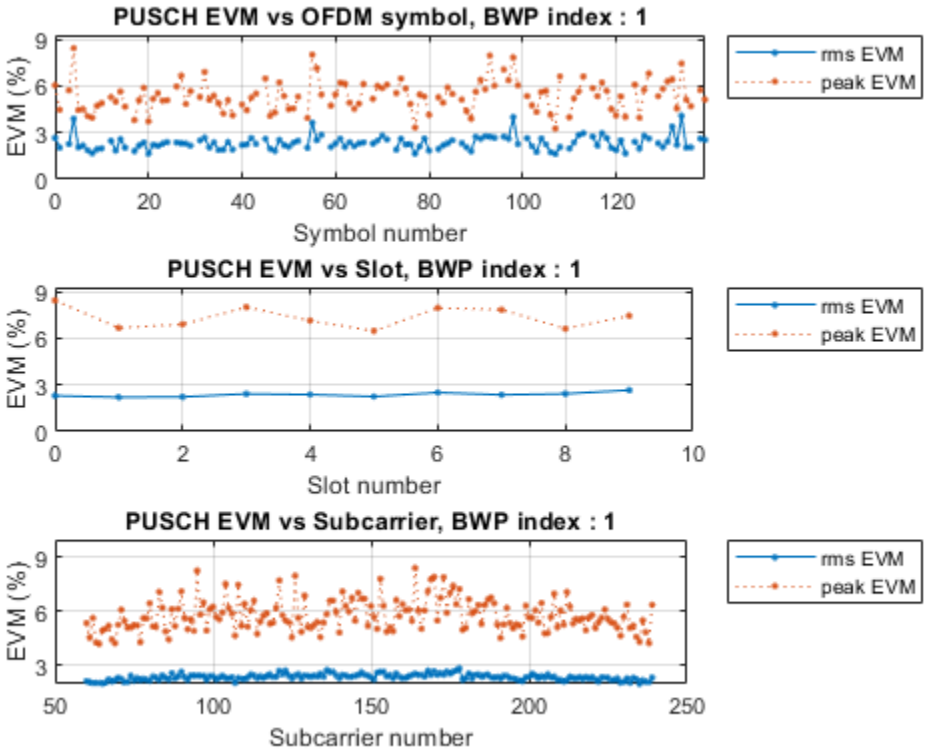
The function, hNRPUSCHEVM, performs these steps to decode and analyze the waveform:

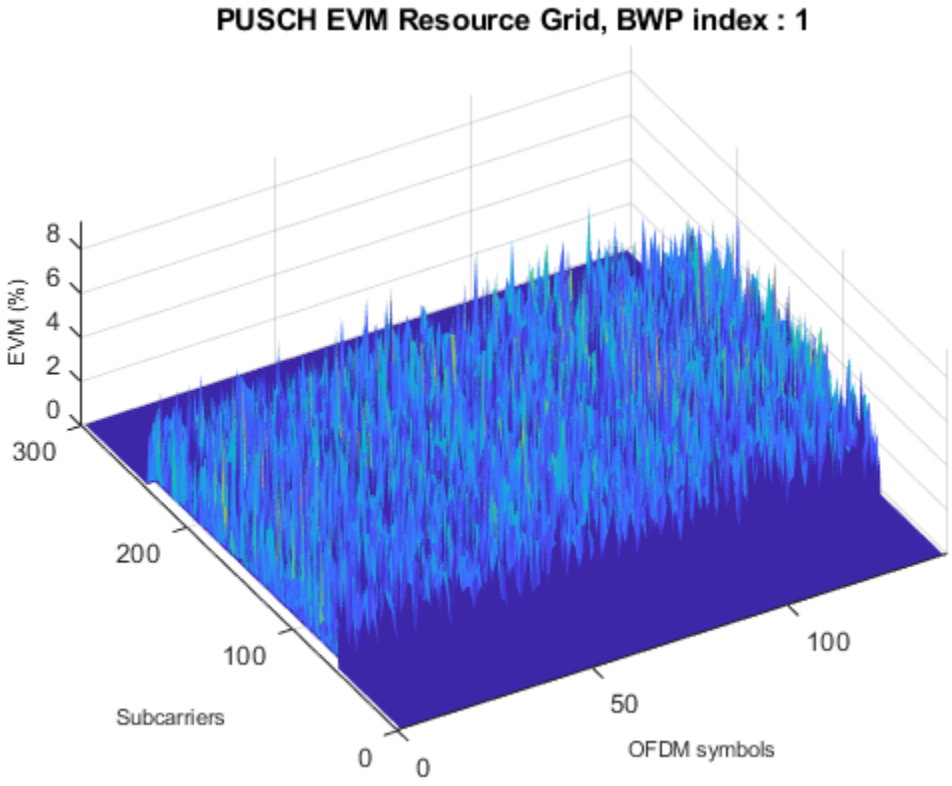
- Coarse frequency offset estimation and correction
- Integer frequency offset estimation and correction
- I/Q imbalance estimation and correction
- Synchronization using the demodulation reference signal (DM-RS) over one frame for FDD (two frames for TDD)
- OFDM demodulation of the received waveform
- Fine frequency offset estimation and correction
- Channel estimation
- Equalization
- Common Phase error (CPE) estimation and compensation
- PUSCH EVM computation (enable the switch `evm3GPP` to process according to the EVM measurement requirements specified in TS 38.101-1(FR1) or TS 38.101-2 (FR2), Annex F.

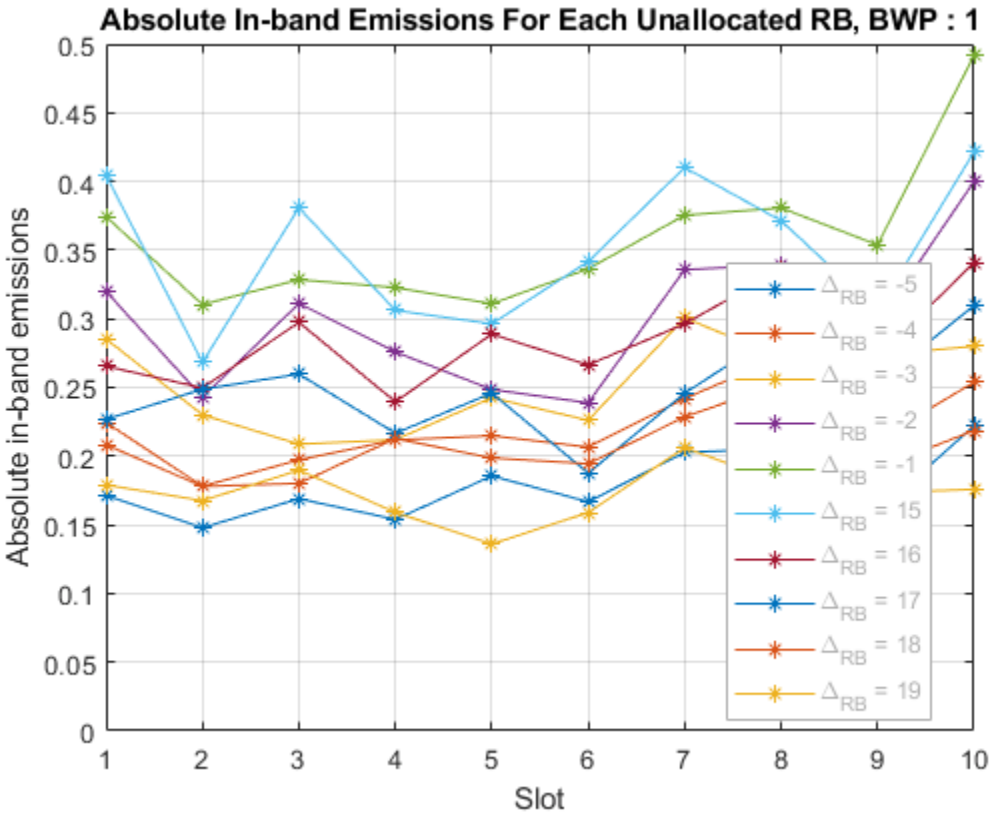
The example measures and outputs various EVM related statistics per symbol, per slot, and per frame peak EVM and RMS EVM. The example displays the EVM for each slot and frame, and displays the overall EVM averaged over the entire input waveform. The example produces these plots: EVM versus per OFDM symbol, slot, subcarrier, overall EVM and in-band emissions. Each EVM related plot displays the peak versus RMS EVM.

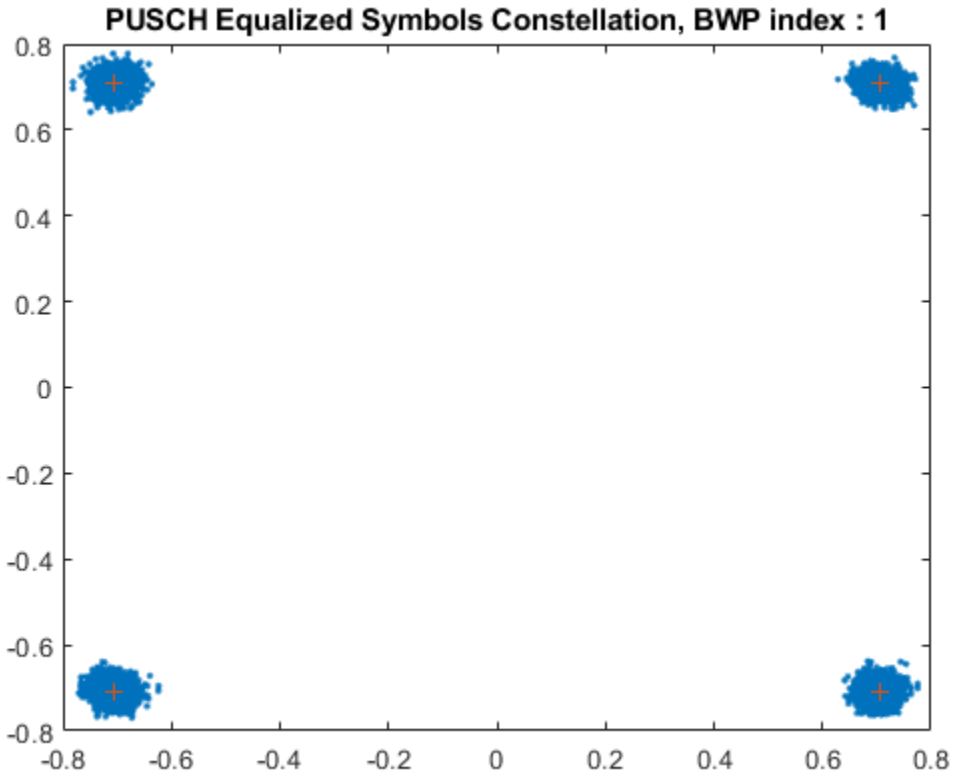
```
% Compute and display EVM measurements
cfg = struct();
cfg.Evm3GPP = evm3GPP;
cfg.PlotEVM = plotEVM;
cfg.DisplayEVM = displayEVM;
cfg.IQImbalance = IQImbalanceON;
[evmInfo,eqSym,refSym] = hNRPUSCHEVM(wavegen.Config,rxWaveform,cfg);
```

```
EVM stats for BWP idx : 1
 RMS EVM, Peak EVM, slot 0: 2.305 8.423%
 RMS EVM, Peak EVM, slot 1: 2.201 6.659%
 RMS EVM, Peak EVM, slot 2: 2.225 6.888%
 RMS EVM, Peak EVM, slot 3: 2.424 7.995%
 RMS EVM, Peak EVM, slot 4: 2.373 7.129%
 RMS EVM, Peak EVM, slot 5: 2.250 6.467%
 RMS EVM, Peak EVM, slot 6: 2.503 7.943%
 RMS EVM, Peak EVM, slot 7: 2.360 7.830%
 RMS EVM, Peak EVM, slot 8: 2.429 6.609%
 RMS EVM, Peak EVM, slot 9: 2.656 7.438%
Averaged RMS EVM frame 0: 2.376%
Averaged overall RMS EVM: 2.376%
Peak EVM : 8.4234%
```


PUSCH EVM vs OFDM symbol, BWP index : 1
EVM (%)
Symbol number
rms EVM
peak EVM
PUSCH EVM vs Slot, BWP index : 1
EVM (%)
Slot number
rms EVM
peak EVM
PUSCH EVM vs Subcarrier, BWP index : 1
EVM (%)
Subcarrier number
rms EVM
peak EVM


PUSCH EVM Resource Grid, BWP index : 1
EVM (%)
8
6
4
2
0
300
200
100
Subcarriers
0
0
50
100
OFDM symbols


Absolute In-band Emissions For Each Unallocated RB, BWP : 1
Absolute in-band emissions
Slot
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
1
2
3
4
5
6
7
8
9
10
Δ_RB = -5
Δ_RB = -4
Δ_RB = -3
Δ_RB = -2
Δ_RB = -1
Δ_RB = 15
Δ_RB = 16
Δ_RB = 17
Δ_RB = 18
Δ_RB = 19



Local functions

```
function plotNonLinearCharacteristic(memoryLessNonlinearity)
    % Plot the nonlinear characteristic of the power amplifier (PA) impairment
    % represented by the input parameter memoryLessNonlinearity, which is a
    % comm.MemorylessNonlinearity Communications Toolbox(TM) System object.

    % Input samples
    x = complex((1/sqrt(2))*(-1+2*rand(1000,1)),(1/sqrt(2))*(-1+2*rand(1000,1)));

    % Nonlinearity
    yRapp = memoryLessNonlinearity(x);
    % Release object to feed it a different number of samples
    release(memoryLessNonlinearity);

    % Plot characteristic
    figure;
    plot(10*log10(abs(x).^2),10*log10(abs(x).^2));
    hold on;
    grid on
    plot(10*log10(abs(x).^2),10*log10(abs(yRapp).^2),'.');
    xlabel('Input Power (dBW)');
    ylabel('Output Power (dBW)');
    title('Nonlinearity Impairment')
```

```
    legend('Linear characteristic', 'Rapp nonlinearity','Location','Northwest');
end
```

See Also

Related Examples

- “5G NR-TM and FRC Waveform Generation” on page 7-12
- “NR Phase Noise Modeling and Compensation” on page 4-17

5G NR Downlink Carrier Aggregation, Demodulation, and Analysis

This example shows how to generate, aggregate, and demodulate multiple downlink carriers using 5G Toolbox™ features.

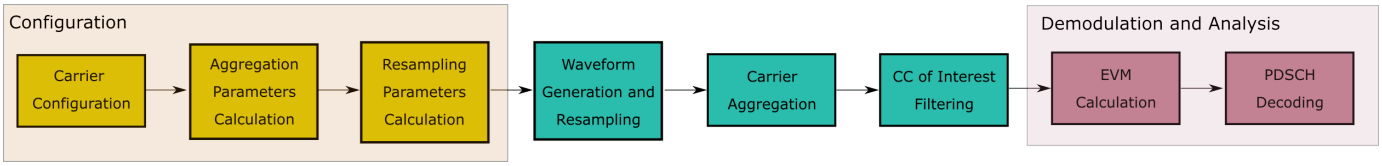
Introduction

This example generates an NR waveform with carrier aggregation (CA). To perform carrier aggregation, the example calculates the frequency offsets for the intraband contiguous CA case, as described in TS 38.104 Section 5.3A. The example also supports customized intraband noncontiguous and interband CA scenarios.

To generate an aggregated downlink waveform, the example configures a standard-compliant downlink fixed reference channel (FRC) for each component carrier. TS 38.101-1 Annex A.3 defines the physical downlink shared channel (PDSCH) FRCs for FR1 and TS 38.101-2 Annex A.3 defines the PDSCH FRCs for FR2. For the FRC waveform generation, you can specify the channel bandwidth, the subcarrier spacing, the modulation, the duplexing mode, and the cell ID. For more information on how to generate DL FRCs, see “5G NR-TM and FRC Waveform Generation” on page 7-12.

After generating the component carriers (CCs), the example resamples the waveforms to a common sample rate and combines the CCs to generate the aggregated waveform.

Finally, the example filters and downsamples a selected CC to perform EVM measurements and PDSCH decoding.



Selection of Component Carrier Parameters

The vector `ChannelBandwidths` specifies the bandwidth for each CC. The length of this vector corresponds to the number of CCs. The elements of `ChannelBandwidths` must be in the set {5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90 100} MHz for FR1 and in the set {50, 100, 200, 400} MHz for FR2. TS 38.101-1 Section 5.5A.1 Table 5.5A.1-1 and TS 38.101-2 Section 5.5A.1 Table 5.5A.1-1 list the valid combinations of bandwidths for FR1 and FR2 carrier aggregation, respectively.

If the `ccSpacings` vector is empty, the example calculates the spacings between consecutive carriers, `ccSpacings`, as described in TS 38.104. To select the spacings of your choice, add the spacings to the `ccSpacings` vector.

```
% Configure three carriers for the aggregation. You can select a different
% number of carriers by modifying the number of elements in the
% |channelBandwidths|, |SCSs|, |modulations|, and |nCellIDs| vectors.
frequencyRange = 'FR1'; % (FR1 or FR2)
channelRaster = 100; % Channel raster in kHz (15, 60 or 100)
channelBandwidths = [60 40 40]; % Channel bandwidths in MHz
                                % (5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80,
                                % 90 or 100 for FR1)
                                % (50, 100, 200 or 400 for FR2)
```

```
SCSs = [30 30 30]; % Subcarrier spacings in kHz
                   % (15, 30 or 60 for FR1)
                   % (60 or 120 for FR2)
modulations = {'QPSK' '64QAM' '256QAM'}; % ('QPSK', '64QAM' or '256QAM' for FR1)
                                         % ('QPSK', '16QAM' or '64QAM' for FR2)
nCellIDs = [1 2 3]; % Cell IDs
duplexMode = 'TDD'; % Duplex mode ('TDD' or 'FDD')
sv = '15.2.0'; % Standard version ('15.1.0', '15.2.0' or '15.7.0')
numSubframes = 10; % Number of subframes to generate per carrier

% Choose the spacings between consecutive carriers, |ccSpacings|, or leave
% the vector empty for calculating the spacings, as described in TS 38.104
% Section 5.3A
ccSpacings = []; % MHz

% Select the CC to demodulate
CCofInterest = 2;

% Verify the number of CCs, channel bandwidths, subcarrier spacings,
% modulations, and spacings. Additionally, check that the spacings are
% greater than 0.
hNRVerifyCarrierParameters(length(channelBandwidths),length(SCSs),...
    length(modulations),length(nCellIDs),ccSpacings);
```

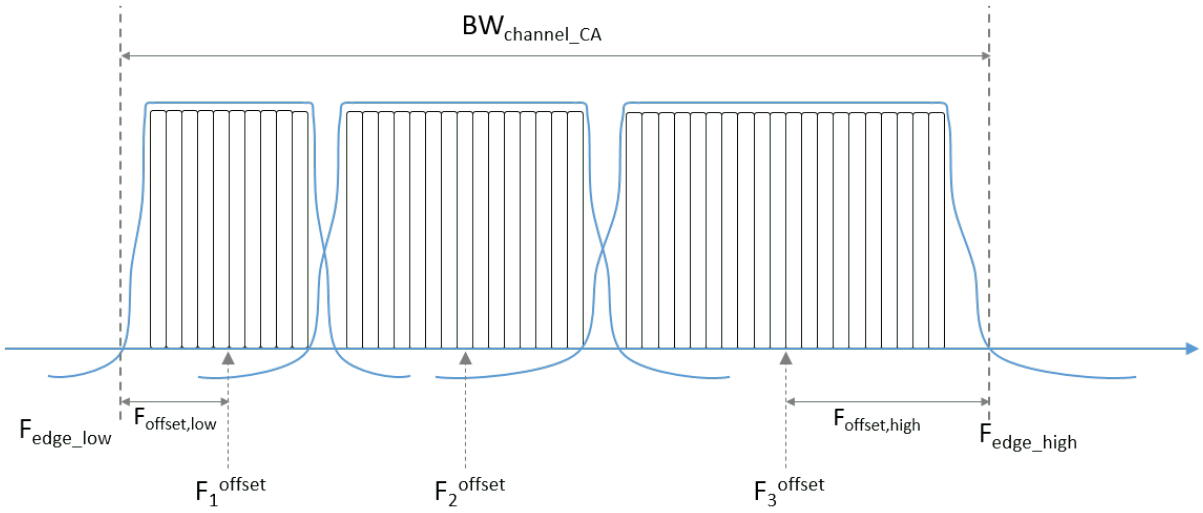
Component Carrier Configuration

Generate a configuration structure for each CC using `hNRReferenceWaveformGenerator`. Store the configuration structures for all CCs in a cell array.

```
% Establish the number of component carriers
numCC = length(channelBandwidths);

% CC configuration
referenceChannel = cell(1,numCC);
wavegen = cell(1,numCC);
for i = 1:numCC
    % Select a reference channel, FRC, based on the chosen modulation and
    % frequency range
    referenceChannel{i} = strcat('DL-FRC-',frequencyRange,'-',modulations{i});

    % Create a generator object for the above PDSCH FRC reference channel
    wavegen{i} = hNRReferenceWaveformGenerator(referenceChannel{i},...
        channelBandwidths(i),SCSs(i),duplexMode,nCellIDs(i),sv);
end
```

Carrier Aggregation Parameters Calculation

To perform the carrier aggregation, you must calculate the frequency parameters described in TS 38.104, Sections 5.3 and 5.4.



- `Fc_offset` is a vector containing the center frequency of each CC at baseband
- `F_offset_low` is the frequency offset from the lower `Fc_offset` to the lower aggregated bandwidth edge
- `F_offset_high` is the frequency offset from the upper `Fc_offset` to the upper aggregated channel bandwidth edge
- `F_edge_low` is the lower edge of the aggregated channel bandwidth
- `F_edge_high` is the upper edge of the aggregated channel bandwidth
- `BW_channel_CA` is the aggregated channel bandwidth of all CCs

Center the lower carrier component at baseband (`Fc_offset(1)` = 0 Hz) and compute the center frequency for the rest of the CCs. To determine the center frequencies by following the method described in TS 38.104 Section 5.3A, you must calculate the minimum guardbands and the minimum CC spacings. Alternatively, to select your own center frequencies, choose a nonempty `ccSpacings` vector.

```
% Get the largest SCS configuration, |SCSConfig|, among the SCS
% configurations supported for each two consecutive channel bandwidths to
% obtain the minimum guardband, as described in TS 38.104 Section 5.4.1.2.
table = hGetGBTable(frequencyRange); % Minimum guardband table
if isequal(frequencyRange,'FR1')
    if any(channelBandwidths(:) == 5)
        SCSConfig = 1;
    else
        SCSConfig = 2;
    end
else
    SCSConfig = 3;
end

% Calculate the minimum guardband, as described in TS 38.104 Section 5.3.3
% Table 5.3.3-1 for FR1 and Table 5.3.3-1 for FR2, to obtain the minimum CC
% spacings.
minimumGuardBand = zeros(1,numCC);
```

```
NDLRB = zeros(1,numCC);
scs = strcat(num2str(2^SCSConfig*15),'kHz'); % Common SCS in kHz to extract
                                             % minimum guardband
for i = 1:numCC
    NDLRB(i) = wavegen{i}.Config.SCSCarriers{1}.NSizeGrid;
    minimumGuardBand(i) = table{{scs},{strcat(num2str(channelBandwidths(i)),...
        'MHz');}}; % kHz
    minimumGuardBand(i) = minimumGuardBand(i)*1e-3; % MHz
end

% Compute the minimum CC spacings, as defined in TS 38.104 Section
% 5.4.1.2. Use these spacings to calculate the center frequencies and for
% filtering each CC.
minCCSpacings = zeros(1,numCC-1); % Minimum CC spacing
for k = 2:numCC
    minCCSpacings(k-1) = hNRCarrierAggregationChannelSpacing( ...
        channelBandwidths(k-1), channelBandwidths(k), minimumGuardBand(k-1), ...
        minimumGuardBand(k),channelRaster,SCSConfig); % MHz
end

% Determine the center frequency for each CC with respect to 0 Hz.
% Initially, the lower carrier frequency is at baseband (Fc_offset(1) = 0 Hz).
Fc_offset = zeros(1,numCC);
if isempty(ccSpacings)
    ccSpacings = minCCSpacings;
end
for k = 2:numCC
    Fc_offset(k) = Fc_offset(k-1) + ccSpacings(k-1); % MHz
end
```

Calculate the frequency offsets from the lower and upper center frequencies to the lower and upper aggregated bandwidth edges, respectively, as described in TS 38.104 Section 5.3A.

```
F_offset_low = (NDLRB(1)*12+1)*(SCSs(1)*1e-3)/2 + minimumGuardBand(1); % MHz
F_offset_high = (NDLRB(end)*12-1)*(SCSs(end)*1e-3)/2 + minimumGuardBand(end); %MHz
```

Compute the lower and upper edges of the aggregated channel bandwidth, TS 38.104 Section 5.3A.

```
F_edge_low = Fc_offset(1) - F_offset_low; % MHz
F_edge_high = Fc_offset(end) + F_offset_high; % MHz
```

Calculate the aggregated channel bandwidth, TS 38.104 Section 5.3A

```
BW_channel_CA = F_edge_high - F_edge_low; % MHz
fprintf('BW_channel_CA: %0.4f MHz\n',BW_channel_CA);
```

```
BW_channel_CA: 141.0800 MHz
```

Determine the frequency shift to center the aggregated channel bandwidth at baseband (0 Hz).

```
shiftToCenter = -1*(BW_channel_CA/2 + F_edge_low);

% Center aggregated bandwidth at baseband
Fc_offset = Fc_offset + shiftToCenter;
F_edge_low = Fc_offset(1) - F_offset_low;
F_edge_high = Fc_offset(end) + F_offset_high;

% Display frequency band edges
fprintf('F_edge_low:  %0.4f MHz\n',F_edge_low);
```

```
F_edge_low:  -70.5400 MHz
```

```
fprintf('F_edge_high: %0.4f MHz\n',F_edge_high);
```

```
F_edge_high: 70.5400 MHz
```

```
fprintf('F_offset_low:  %0.4f MHz\n',F_offset_low);
```

```
F_offset_low:  30.7050 MHz
```

```
fprintf('F_offset_high: %0.4f MHz\n',F_offset_high);
```

```
F_offset_high: 20.6750 MHz
```

```
% Display carrier frequencies
fprintf('\n');
for i = 1:numCC
    fprintf('Component Carrier %d:\n',i);
    fprintf('   Fc: %0.4f MHz\n', Fc_offset(i));
end
```

```
Component Carrier 1:

   Fc: -39.8350 MHz

Component Carrier 2:

   Fc: 9.9650 MHz

Component Carrier 3:

   Fc: 49.8650 MHz
```

Oversampling Rate Calculation

Calculate the required oversampling factors for each component carrier, OSRs, to use a common sampling rate for the aggregated waveform.

```
% Obtain sample rates of the component carriers
CCSR = zeros(1,numCC);
carriers = cell(1,numCC);
for i = 1:numCC
    carriers{i} = nrCarrierConfig;
    carriers{i}.NCellID = nCellIDs(i);
    carriers{i}.NSizeGrid = NDLRB(i);
    carriers{i}.SubcarrierSpacing = SCSs(i);
    carriers{i}.CyclicPrefix = wavegen{i}.Config.BandwidthParts{1}.CyclicPrefix;
    info = nrOFDMInfo(carriers{i});
    CCSR(i) = info.SampleRate; % Hz
end

% Calculate the oversampling ratio for the largest BW CC to ensure the
% waveform occupies a maximum of 85% of the total bandwidth, |bwfraction|
bwfraction = 0.85; % Bandwidth utilization of 85%
OSR = (BW_channel_CA/bwfraction)/(max(CCSR)/1e6);

% To simplify the resampling operation, choose an oversampling ratio which
% is a power of 2: calculate the next power of two above OSR
OSR = 2^ceil(log2(OSR));
```

```
% Calculate the overall sample rate for the aggregated waveform
SR = OSR*max(CCSR); % Hz
fprintf('\nOutput sample rate: %0.4f Ms/s\n\n',SR/1e6);
```

```
Output sample rate: 245.7600 Ms/s
```

```
% Calculate the individual oversampling factors for the component carriers
OSRs = SR./CCSR;
```

Waveform Generation and Carrier Aggregation

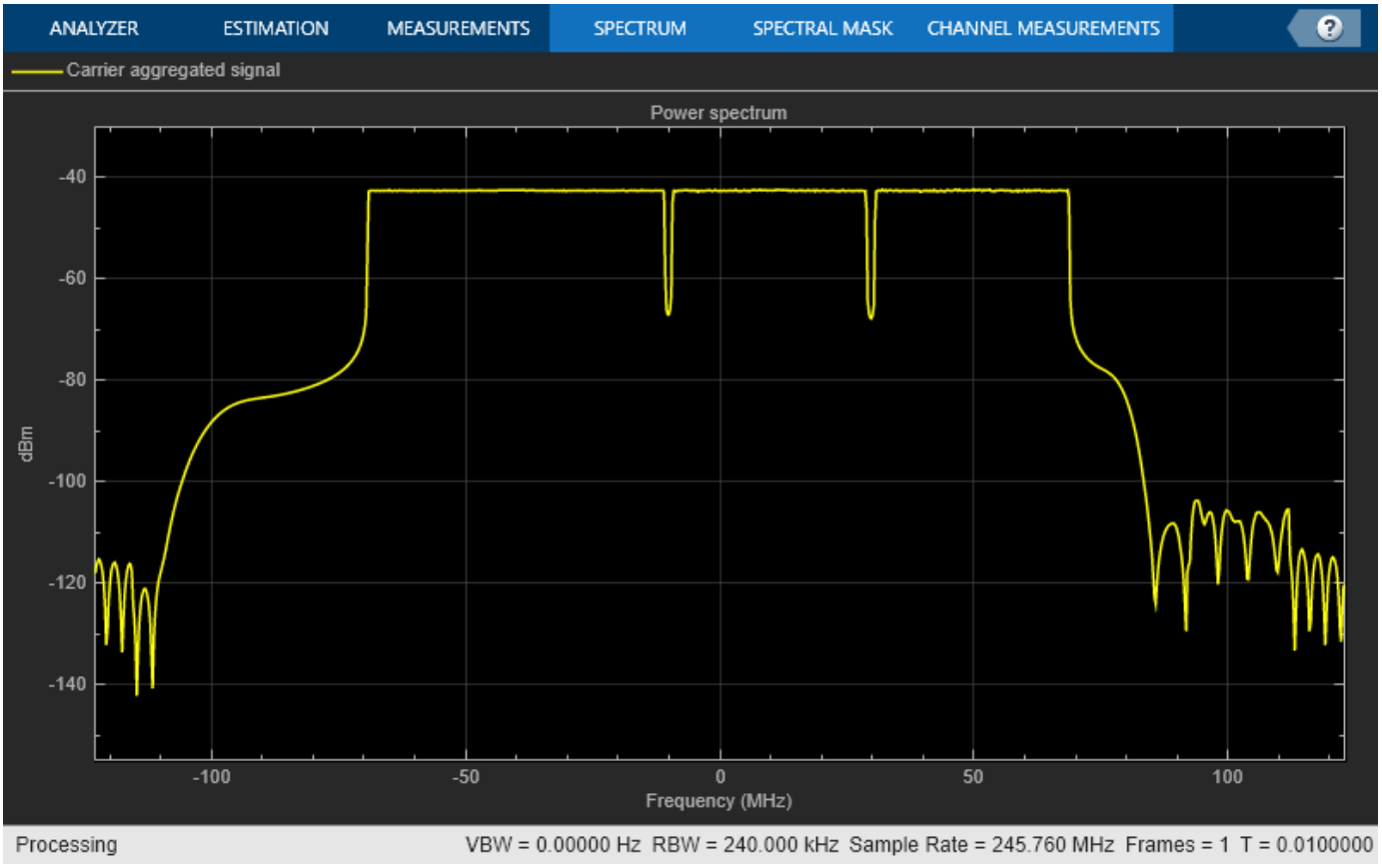
Call the `generateWaveform` function from the `hNRReferenceWaveformGenerator` helper file to generate the waveform for each CC. Resample each carrier to a common sample rate. Finally, use `comm.MultibandCombiner` to frequency modulate the carriers to the appropriate center frequency, and aggregate the CCs to form the combined waveform.

```
% Generate and aggregate the component carriers
tmwaveinfo = cell(1,numCC);
waveInfo = cell(1,numCC);
resourcesInfo = cell(1,numCC);
clear waveform
for i = numCC:-1:1
    % Generate each CC
    [wf,waveInfo{i},resourcesInfo{i}] = generateWaveform(wavegen{i},...
        numSubframes);

    % Resample the CCs so that they have the same sample rate
    waveform(:,i) = resample(wf,OSRs(i),1)/OSRs(i);
end

% Aggregate all CC. comm.MultibandCombiner applies the frequency offsets
% and combines the resulting signals together.
carrierAggregator = comm.MultibandCombiner(InputSampleRate = SR,...
    FrequencyOffsets = Fc_offset*1e6,...
    OutputSampleRateSource = 'Property',...
    OutputSampleRate = SR);

waveform = carrierAggregator(waveform);
```

Plot Carrier Aggregated Spectrum

Display the spectrum of the carrier aggregated signal. This example does not upconvert the waveform to radio frequency (RF), the center of the aggregated bandwidth is at baseband (0 Hz).

```
% Create spectrum analyzer object
specPlot = spectrumAnalyzer("SampleRate",SR,...
                            "AveragingMethod","exponential",...
                            "ForgettingFactor",1, ...
                            "ChannelNames","Carrier aggregated signal",...
                            "ShowLegend",true,...
                            "Title","Power spectrum" );
% Plot spectrum
specPlot(waveform);
```



CC Demodulation and Filtering

Choose a CC, then demodulate, filter and downsample the CC of your choice, following these steps.

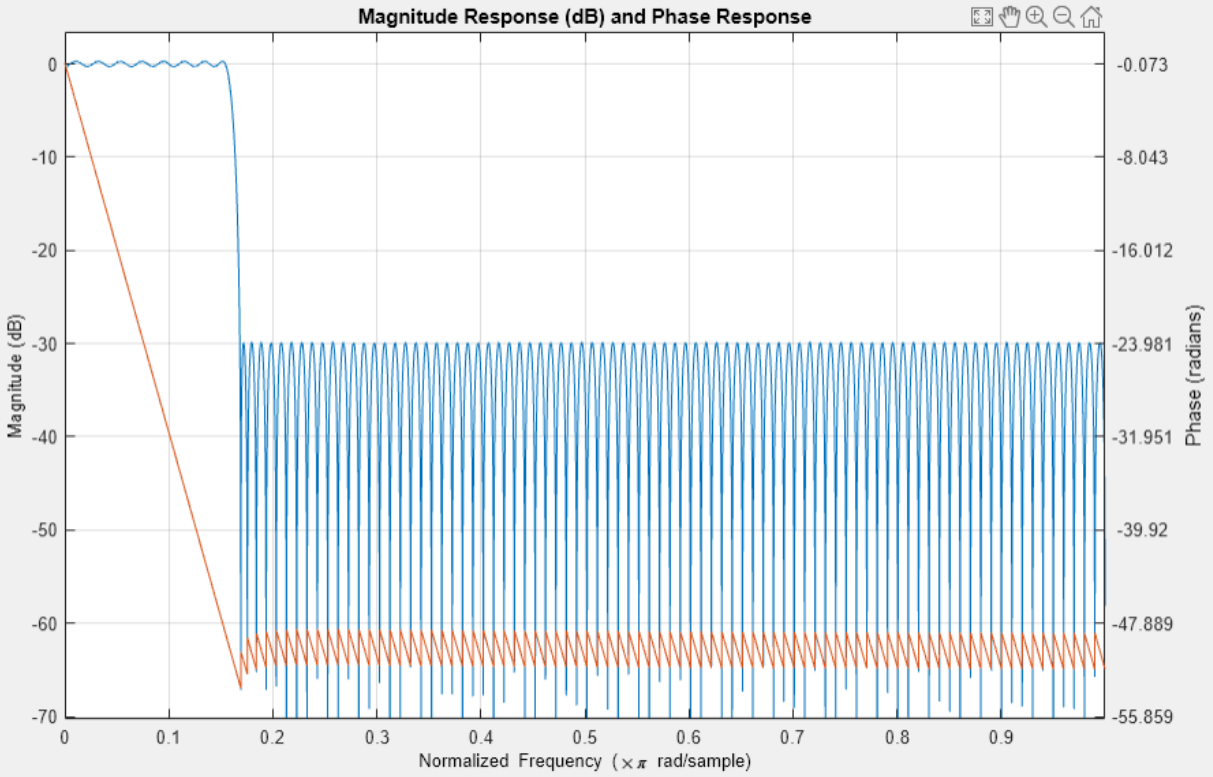
- Bring the CC to baseband (0 Hz)
- Calculate the passband and stopband frequencies of the filter
- Filter out the neighboring CCs by using the designed filter and downsample the CC.

```
% Verify carrier of interest ID
if CCofInterest > numCC || CCofInterest <= 0 || mod(CCofInterest,1) ~= 0
    error('nr5g:NRDownlinkCarrierAggregationExample:CCOutOfRange',...
        'Cannot demodulate CC number %d, choose an integer number that falls between 1 and %d\n'
        CCofInterest, numCC) ;
end
fprintf(1,'Extracting CC number %d: \n', CCofInterest);
```

```
Extracting CC number 2:
```

```
% Define downsampling filter order
filterOrder = 201;

% Precalculate the filter passband and stopband values for all CC
firPassbandVec = (NDLRB*12-1).*(SCSs*1e-3)/2 / (SR/1e6/2);
firStopbandVec = hNRCarrierAggregationStopband(minCCSpacings,NDLRB,SR,SCSs);
```

```
% Choose the passband and stopband values for the carrier of interest
firPassband = firPassbandVec(CCofInterest);
firStopband = firStopbandVec(CCofInterest);

% Pad signal with zeros to consider filter transient response length
waveform = [waveform; zeros(filterOrder*2,size(waveform,2))];

% Center the carrier of interest at 0 Hz
frequencyShifter = comm.PhaseFrequencyOffset(SampleRate = SR,...
    FrequencyOffset = -Fc_offset(CCofInterest)*1e6);
demodulatedCC = frequencyShifter(waveform);

% To ease the filter design requirements, apply the downsampling in two
% stages if necessary. Use a downsampling factor of 4 in the initial stage.
% If the quality of the resulting signal is not as required, consider a
% different filter design in this initial stage.
if (firStopband < 0.1)
    % Downsample by 4 in the initial stage
    SRC = 4;
    demodulatedCC = resample(demodulatedCC,1,SRC);
    % Update passband and stopband values
    firPassband = firPassband * SRC;
    firStopband = firStopband * SRC;
else
    % Do not apply an extra downsampling
    SRC = 1;
end

% Design the lowpass filter to filter out the CC of your choice
frEdges = [0 firPassband firStopband 1];
firFilter = dsp.FIRFilter;
firFilter.Numerator = firpm(filterOrder,frEdges,[1 1 0 0]);

% Display the response of the designed filter
fvtool(firFilter,'Analysis','freq');
```




```
% Filter the signal to extract the component of interest
rxWaveform  = firFilter(demodulatedCC);

filteredSpecPlot = spectrumAnalyzer("SampleRate",SR,...
                                    "AveragingMethod","exponential",...
                                    "ForgettingFactor",1,...
                                    "ChannelNames",{"Carrier aggregated signal", "Filtered signa
                                    "ShowLegend",true,...
                                    "Title","Power spectrum");
filteredSpecPlot([demodulatedCC, rxWaveform]);
```


```
% Downsample the filtered carrier to its baseband rate
rxWaveform = downsample(rxWaveform,OSRs(CCofInterest)/SRC);
```

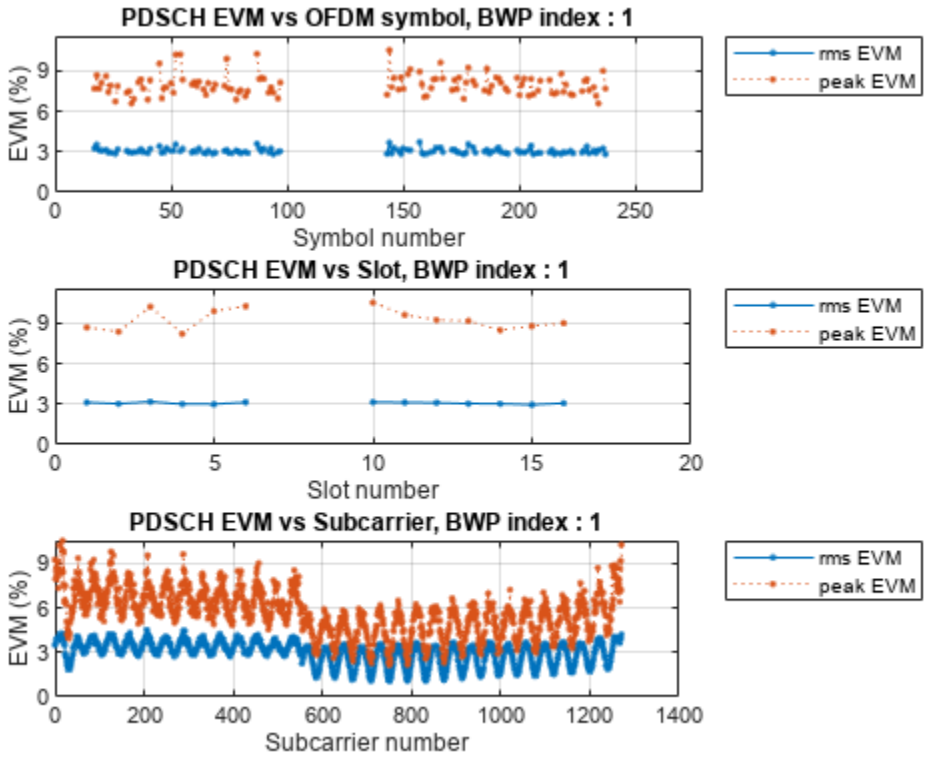
EVM Measurements

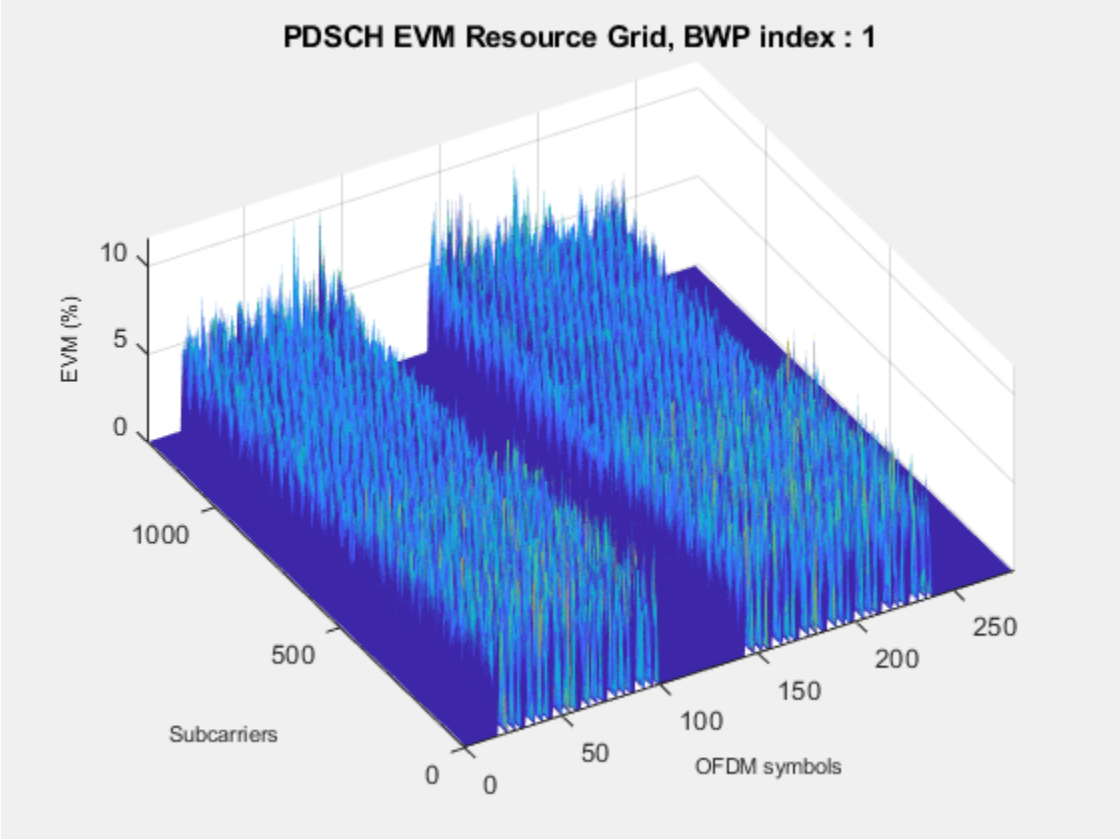
The `hNRPDSCHEVM` helper function returns the PDSCH EVM by performing synchronization, OFDM demodulation, channel estimation, and equalization. The function displays the EVM for each slot and frame and the overall EVM averaged over the entire input waveform. The function also plots these graphs: EVM per OFDM symbol, slot, subcarrier, and overall EVM.

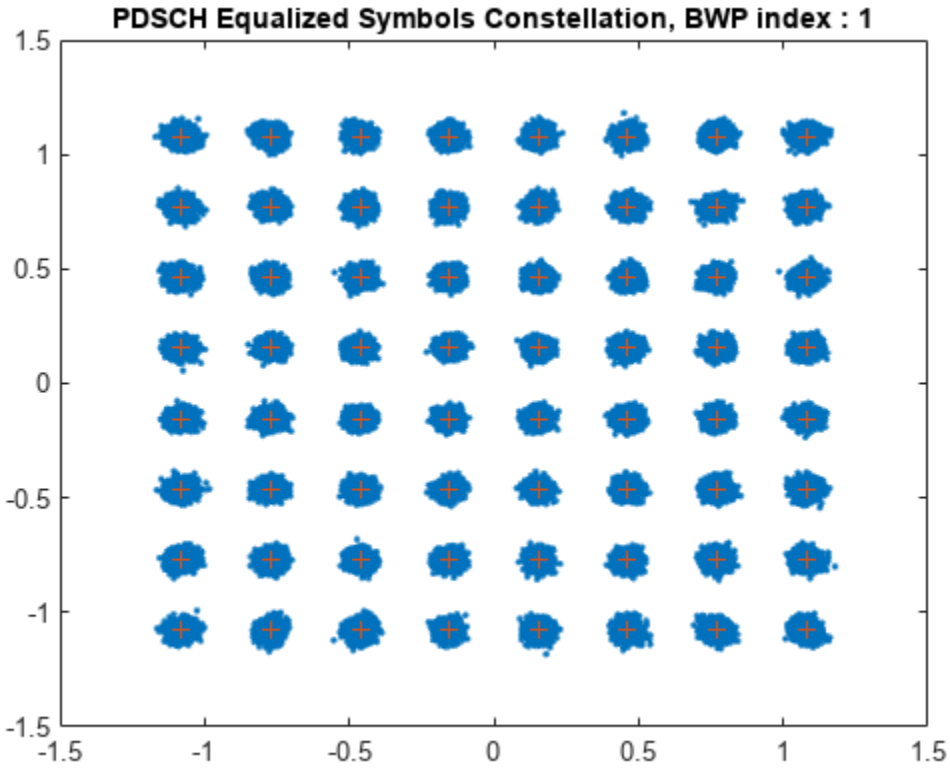
```
% Parameterize the channel estimator configuration using the structure |cfg|
cfg = struct();
cfg.Evm3GPP = true; % To measure EVM as defined in TS 38.104, Annex B(FR1)
                    % / Annex C(FR2) set |Evm3GPP| to |true|.
cfg.TargetRNTIs = [];
cfg.PlotEVM = true;
cfg.DisplayEVM = true;
cfg.Label = wavegen{CCofInterest}.ConfiguredModel{1};

% Perform EVM measurements and plot results
[evmInfo,eqSym,refSym] = hNRDownlinkEVM(wavegen{CCofInterest}.Config,...
    rxWaveform,cfg);
```

```
EVM stats for BWP idx : 1
Low edge PDSCH RMS EVM, Peak EVM, slot 1: 3.130 8.695%
High edge PDSCH RMS EVM, Peak EVM, slot 1: 2.807 7.912%
```

```
Low edge PDSCH RMS EVM, Peak EVM, slot 2: 3.035 8.388%
High edge PDSCH RMS EVM, Peak EVM, slot 2: 2.797 7.472%
Low edge PDSCH RMS EVM, Peak EVM, slot 3: 3.181 10.224%
High edge PDSCH RMS EVM, Peak EVM, slot 3: 2.824 7.939%
Low edge PDSCH RMS EVM, Peak EVM, slot 4: 3.004 8.220%
High edge PDSCH RMS EVM, Peak EVM, slot 4: 2.804 7.551%
Low edge PDSCH RMS EVM, Peak EVM, slot 5: 3.000 9.909%
High edge PDSCH RMS EVM, Peak EVM, slot 5: 2.802 7.593%
Low edge PDSCH RMS EVM, Peak EVM, slot 6: 3.140 10.270%
High edge PDSCH RMS EVM, Peak EVM, slot 6: 2.822 7.948%
Low edge PDSCH RMS EVM, Peak EVM, slot 10: 3.146 10.539%
High edge PDSCH RMS EVM, Peak EVM, slot 10: 2.796 8.589%
Low edge PDSCH RMS EVM, Peak EVM, slot 11: 3.121 9.624%
High edge PDSCH RMS EVM, Peak EVM, slot 11: 2.788 7.994%
Low edge PDSCH RMS EVM, Peak EVM, slot 12: 3.103 9.248%
High edge PDSCH RMS EVM, Peak EVM, slot 12: 2.816 7.963%
Low edge PDSCH RMS EVM, Peak EVM, slot 13: 3.042 9.182%
High edge PDSCH RMS EVM, Peak EVM, slot 13: 2.819 7.734%
Low edge PDSCH RMS EVM, Peak EVM, slot 14: 3.030 8.492%
High edge PDSCH RMS EVM, Peak EVM, slot 14: 2.815 7.428%
Low edge PDSCH RMS EVM, Peak EVM, slot 15: 2.955 8.799%
High edge PDSCH RMS EVM, Peak EVM, slot 15: 2.801 8.219%
Low edge PDSCH RMS EVM, Peak EVM, slot 16: 3.067 9.011%
High edge PDSCH RMS EVM, Peak EVM, slot 16: 2.801 7.699%
Averaged low edge RMS EVM, frame 0: 3.074%
Averaged high edge RMS EVM, frame 0: 2.807%
Averaged RMS EVM frame 0: 3.074%
```




PDSCH EVM Resource Grid, BWP index : 1
EVM (%)
10
5
0
1000
500
0
Subcarriers
0
50
100
150
200
250
OFDM symbols




```
Averaged overall PDSCH RMS EVM: 3.074%
Overall PDSCH Peak EVM = 10.5387%
```

PDSCH Decoding

Decode the PDSCH of the recovered signal and check the resulting CRC for errors.

```
% Perform time synchronization on the input waveform
offset = nrTimingEstimate(carriers{CCofInterest},rxWaveform,...
    waveInfo{CCofInterest}.ResourceGridBWP);
rxWaveform = rxWaveform(1+offset:end,:);

% Perform OFDM demodulation
rxGrid = nrOFDMDemodulate(carriers{CCofInterest},rxWaveform);

% Get the allocated slots and OFDM symbols per slot
allocatedSlots = zeros(1,size(resourcesInfo{CCofInterest}.WaveformResources.PDSCH.Resources,2));
for i=1:length(allocatedSlots)
    allocatedSlots(i) = resourcesInfo{CCofInterest}.WaveformResources.PDSCH.Resources(i).NSlot;
end
L = carriers{CCofInterest}.SymbolsPerSlot; % OFDM symbols per slot

% Create a DLSCH decoder System object
decodeDLSCH = nrDLSCHDecoder;
decodeDLSCH.MultipleHARQProcesses = false;
decodeDLSCH.TargetCodeRate = wavegen{CCofInterest}.Config.PDSCH{1}.TargetCodeRate;
decodeDLSCH.LDPCDecodingAlgorithm = 'Normalized min-sum';
decodeDLSCH.MaximumLDPCIterationCount = 6;
```

```
for NSlot=1:length(allocatedSlots)
    % Extract slot
    SlotID = allocatedSlots(NSlot);
    rxSlot = rxGrid(:,(1:L)+(SlotID*L),:);
    refSlot = waveInfo{CCofInterest}.ResourceGridBWP(:,(1:L)+(SlotID*L),:);

    % Perform channel estimation
    [estChannelGrid,noiseEst] = nrChannelEstimate(carriers{CCofInterest},...
        rxSlot,refSlot);

    % Get PDSCH resource elements from the received grid and channel estimate
    pdschIndices = resourcesInfo{CCofInterest}.WaveformResources.PDSCH.Resources(NSlot).ChannelIn
    [pdschRx,pdschHest] = nrExtractResources(pdschIndices,rxSlot,estChannelGrid);

    % Perform equalization
    [pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseEst);

    % Perform layer demapping, symbol demodulation, and descrambling
    modulation = wavegen{CCofInterest}.Config.PDSCH{1}.Modulation;
    RNTI = wavegen{CCofInterest}.Config.PDSCH{1}.RNTI;
    [dlschLLRs,rxSymbols] = nrPDSCHDecode(pdschEq,modulation,...
        carriers{CCofInterest}.NCellID,RNTI,noiseEst);

    % Scale LLRs by CSI
    csi = nrLayerDemap(csi); % CSI layer demapping
    NumCW = size(resourcesInfo{CCofInterest}.WaveformResources.PDSCH.Resources(NSlot).Codeword,2)
    for cwIdx = 1:NumCW
        Qm = length(dlschLLRs{cwIdx})/length(rxSymbols{cwIdx}); % Bits per symbol
        csi{cwIdx} = repmat(csi{cwIdx}.',Qm,1);   % Expand by each bit per symbol
        dlschLLRs{cwIdx} = dlschLLRs{cwIdx} .* csi{cwIdx}(:);   % Scaled symbols
    end

    % Calculate the transport block sizes for the codewords in the slot
    trBlkSize = resourcesInfo{CCofInterest}.WaveformResources.PDSCH.Resources(NSlot).TransportBlo

    % Decode the DL-SCH transport channel
    decodeDLSCH.TransportBlockLength = trBlkSize;
    NLayers = wavegen{CCofInterest}.Config.PDSCH{1}.NumLayers;
    RVSequence = wavegen{CCofInterest}.Config.PDSCH{1}.RVSequence;
    [decbits,crc] = decodeDLSCH(dlschLLRs,modulation,NLayers,RVSequence);

    % Display the CRC status
    if crc
        disp(['Slot ' num2str(SlotID) ': CRC failed']);
    else
        disp(['Slot ' num2str(SlotID) ': CRC passed']);
    end
end
```

```
Slot 1: CRC passed
Slot 2: CRC passed
Slot 3: CRC passed
Slot 4: CRC passed
Slot 5: CRC passed
Slot 6: CRC passed
Slot 10: CRC passed
Slot 11: CRC passed
```

```
Slot 12: CRC passed
Slot 13: CRC passed
Slot 14: CRC passed
Slot 15: CRC passed
Slot 16: CRC passed
```

Bibliography

1. 3GPP TS 38.104 "NR; Base Station (BS) radio transmission and reception" 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
2. 3GPP TS 38.101-1 "NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.
3. 3GPP TS 38.101-2 "NR; User Equipment (UE) radio transmission and reception: Part 2: Range 2 Standalone." 3rd Generation Partnership Project; Technical Specification Group Radio Access Network.

Local Functions

```
function table = hGetGBTable(frequencyRange)
%hGetGBTable Minimum guardband table
%   GB = hGetGBTable(FREQUENCYRANGE) extracts the minimum guardband, GB, in
%   kHz for the frequency range, FREQUENCYRANGE. For FR1, the table is
%   described in TS 38.104 Table 5.3.3-1 and for FR2, the table is
%   described in TS 38.104 Table 5.3.3-2.

    if isequal(frequencyRange,'FR1')
        % GB MHz    5       10      15      20      25      30      35      40      45      50     60     7
        gbTable = [242.5   312.5  382.5     452.5  522.5  592.5  572.5  552.5  712.5  692.5  NaN
                   505       665      645      805    785    945    925    905    1065   1045   82
                   NaN     1010      990     1330    1310   1290   1630   1610   1590   1570   15

         % Package GB array into a table
         table = array2table(gbTable,"RowNames",["15kHz","30kHz","60kHz"],"VariableNames",["5MHz"
         table.Properties.Description = 'TS 38.104 Table 5.3.3-1: Minimum guardband (kHz) FR1';
    else
         % GB MHz   50    100    200    400
        gbTable = [1210      2450     4930    NaN;      % 60 kHz
                   1900      2420     4900    9860];    % 120 kHz

         % Package GB array into a table
         table = array2table(gbTable,"RowNames",["60kHz","120kHz"],"VariableNames",["50MHz", "10
         table.Properties.Description = 'TS 38.104 Table 5.3.3-2: Minimum guardband (kHz) FR2';
    end
end

function spacing = hNRCarrierAggregationChannelSpacing(BW1,BW2,GB1,GB2,channelRaster,mu)
%hNRCarrierAggregationChannelSpacing NR nominal channel spacing
%   SPACING = hNRCarrierAggregationChannelSpacing(BW1,BW2,GB1,GB2,CHANNELRASTER,MU)
%   is the NR nominal channel spacing in MHz between two adjacent carriers
%   with bandwidths BW1 and BW2, also expressed in MHz. GB1 and GB2 are the
%   minimum guardbands for bandwidth BW1 and bandwidth BW2, respectively.
%   The NR nominal channel calculation depends on the channel raster,
%   CHANNELRASTER, and the subcarrier spacing (SCS) configuration, MU,
%   which is the largest SCS value among the SCS configurations supported
%   for both of the channel bandwidths.
```

```
        switch channelRaster
            case 15
                % Nominal channel spacing (MHz) as defined in TS 38.104 5.4.1.2
                % for NR operating bands with a 15 kHz channel raster
                n = mu;
                spacing = floor((BW1+BW2-2*abs(GB1-GB2))/(0.015*2^(n+1)))*(0.015*2^(n));
            case 60
                % Nominal channel spacing (MHz) as defined in TS 38.104 5.4.1.2
                % for NR operating bands with a 60 kHz channel raster
                n = mu-2;
                spacing = floor((BW1+BW2-2*abs(GB1-GB2))/(0.06*2^(n+1)))*(0.06*2^(n));
            case 100
                % Nominal channel spacing (MHz) as defined in TS 38.104 5.4.1.2
                % for NR operating bands with a 100 kHz channel raster
                spacing = floor((BW1+BW2-2*abs(GB1-GB2))/0.6)*0.3;
            otherwise
                error('nr5g:hNRCarrierAggregationChannelSpacing:IncorrectRaster','The channel raster
        end
    end

    function stopband = hNRCarrierAggregationStopband(ccSpacing,NDLRB,SR,SCS)
    %hNRCarrierAggregationStopband Stopband values for NR carrier selection
    %   STOPBAND = hNRCarrierAggregationStopband(CCSPACING,NDLRB,SR,SCS)
    %   calculates the stopband values, STOPBAND, in a 5G carrier aggregation
    %   scenario given the component carrier spacing vector CCSPACING, the
    %   vector of bandwidths for each CC, NDLRB, the sampling rate of the
    %   aggregated waveform, SR, and the subcarrier spacing of each component
    %   carrier, SCS.

        numCC = length(NDLRB);
        stopband = zeros(numCC,1);

        for i = 1:numCC
            % Compute two possible stopbands based on spacing with component
            % carrier before and after
            if (i > 1)
                % There is a carrier component before the one being considered
                frStopband1 = ccSpacing(i-1)-(NDLRB(i-1)*12-2)*(SCS(i-1)*1e-3)/2;
            else
                frStopband1 = NaN;
            end
            if (i < numCC)
                % There is a carrier component after the one being considered
                frStopband2 = ccSpacing(i)-(NDLRB(i+1)*12-2)*(SCS(i+1)*1e-3)/2;
            else
                frStopband2 = NaN;
            end
            % Take the minimum value
            stopband(i) = min(frStopband1,frStopband2) / (SR/1e6/2);
        end
    end

    function hNRVerifyCarrierParameters(numberBWs,numberSCSs,numberModulations,numberCellIDs,spacings
    %hNRVerifyCarrierParameters Carrier aggregation parameters verification
    %   hNRVerifyCarrierParameters(NUMBERBWS,NUMBERSCSS,NUMBERMODULATIONS,NUMBERCELLIDS,SPACINGS)
    %   verifies if the number of channel bandwidths, NUMBERBWS, subcarrier
    %   spacings, NUMBERSCSS, modulations, NUMBERMODULATIONS, and cell IDs,
    %   NUMBERCELLIDS, are the same. This function also checks the spacing
```

```
%   between two consecutive carriers, SPACINGS.

    % Verify that there is a channel bandwidth, a subcarrier spacing and a
    % modulation per carrier
    if ~(numberBWs == numberSCSs && numberSCSs == numberModulations && numberModulations == numbe
        error('nr5g:hNRVerifyCarrierParameters:NumberCCConfigParams','Please specify the same nu
    end

    % Check if there are at least two component carriers
    numCC = numberBWs;
    if numCC<2
        error('nr5g:hNRVerifyCarrierParameters:OneCarrier','Please specify more than one channel
    end

    % Check the number of spacings and their values
    if ~isempty(spacings)
        if length(spacings) ~= numCC-1
            error('nr5g:hNRVerifyCarrierParameters:NumberSpacings','The number of CC spacings (%d
        end
        if any(spacings<=0)
            error('nr5g:hNRVerifyCarrierParameters:SpacingsLowerEqualZero','The CC spacings shou
        end
    end
end
```

See Also

Functions

`nrCarrierConfig`

Related Examples

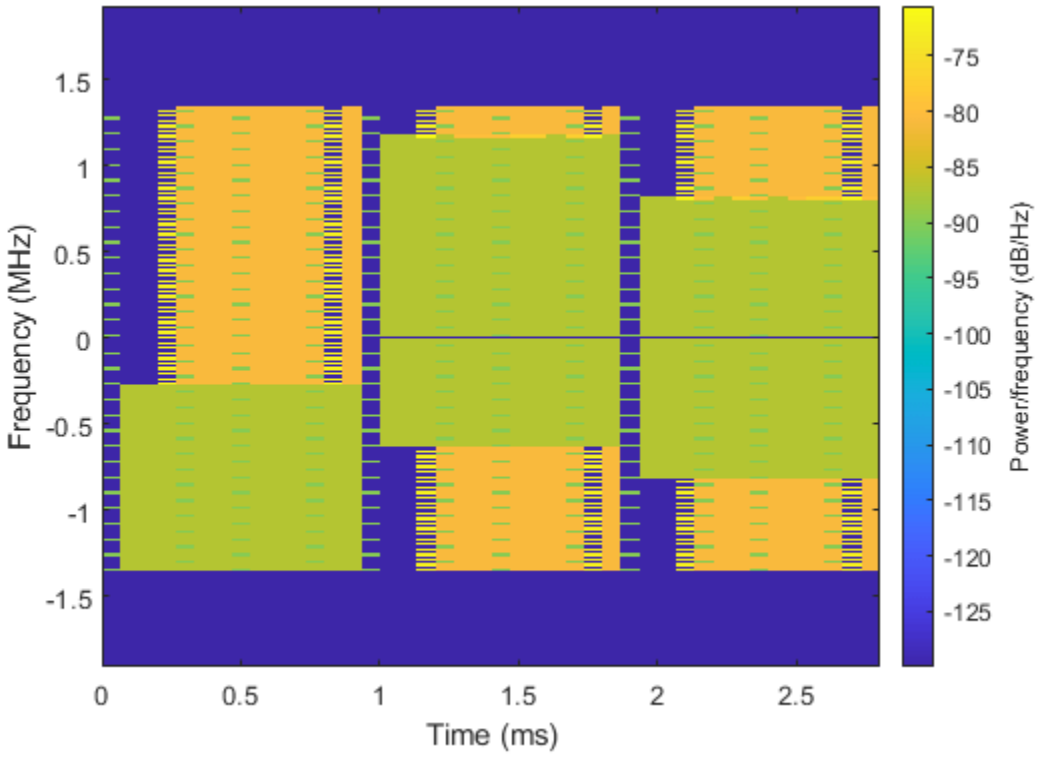
- “EVM Measurement of 5G NR Downlink Waveforms with RF Impairments” on page 7-79
- “5G NR-TM and FRC Waveform Generation” on page 7-12

Dynamic Spectrum Sharing for 5G NR and LTE Coexistence

This example shows how to generate a waveform containing LTE and 5G NR waveforms for dynamic spectrum sharing (DSS) by using LTE Toolbox™ and 5G Toolbox™.

Introduction

Dynamic spectrum sharing allows 5G and LTE to share resources dynamically in the same spectrum. Therefore, 5G can use the existing LTE spectrum without the need for re-farming existing bands. 5G and LTE coexistence also maximizes system efficiency because the LTE and 5G base stations, eNodeB and gNodeB, respectively, can use all available resources at all times. This figure shows an example in which the gNodeB scheduler allocates 5G resources dynamically every millisecond, filling all the resources unused by LTE. Specifically, the figure shows the spectrogram of the combined LTE and 5G waveform, in which the LTE waveform is in shades of green and the 5G waveform is in shades of yellow.



The main requirement of DSS is backward compatibility. That is, LTE devices must function as normal regardless of whether DSS is implemented. Specifically, 5G waveforms in DSS must not interfere with LTE always-on signals and channels, such as the cell reference signal (CRS), primary synchronization signal (PSS), secondary synchronization signal (SSS), and physical broadcast channel (PBCH). The 5G and LTE scheduling of resources can change dynamically every subframe, which is the lowest common time unit between the two technologies.

This example shows how to generate a waveform containing:

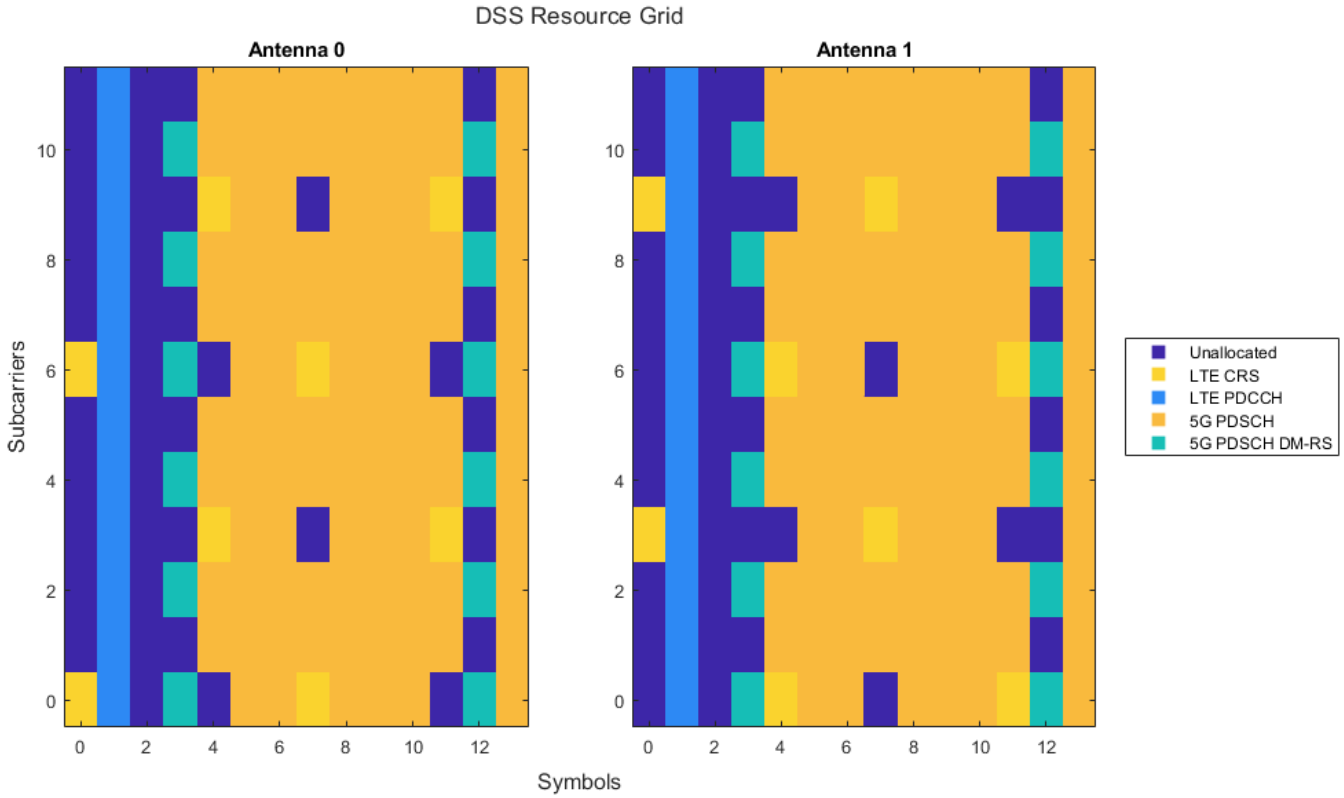
- 5G physical downlink shared channel (PDSCH)

- LTE CRS
- LTE PSS
- LTE SSS
- LTE PBCH
- LTE physical downlink control channel (PDCCH)
- LTE physical control format indicator channel (PCFICH)
- LTE physical hybrid ARQ indicator channel (PHICH)

The 5G PDSCH occupies all resources unused by the LTE waveform. The example implements these techniques to enable DSS.

- Perform rate-matching of the 5G PDSCH around the resource elements (REs) used by the LTE CRS. The higher layer parameter `RateMatchPatternLTE-CRS`, defined in TS 38.331 Section 6.3.2, contains the information needed by the gNodeB to rate-match the PDSCH around the CRS.
- Reserve 5G PDSCH physical resource blocks (PRBs) that are occupied by the LTE PBCH, PSS, and SSS.
- Use Rel-16 alternative locations for the 5G PDSCH demodulation reference signal (DM-RS), as described in TS 38.211 Section 7.4.1.1.2, for a 5G PDSCH configuration with: mapping type A, the first DM-RS allocated in symbol 3, and an additional DM-RS.
- Allocate the 5G PDSCH starting from symbol 3 of each slot to ensure that the 5G PDSCH never collides with LTE control information channels.

This figure shows an example resource grid for a single PRB and single subframe containing the LTE CRS, LTE PDCCH, and 5G PDSCH with its DM-RS. The LTE configuration consists of two CRS ports and the PDCCH scheduled in symbol 1. The 5G PDSCH configuration consists of two layers, mapping type A, allocated symbols from 3 to 13, a DM-RS in symbol 3, and an additional DM-RS. The figure shows the 5G PDSCH rate-matching around the REs allocated for the LTE CRS and the additional DM-RS allocated in symbol 12, instead of 11, to avoid collisions with the LTE CRS in antenna 1.



Waveform Configuration

Configure the LTE and 5G waveforms. You can change the length of the generated waveform, the bandwidth of 5G and LTE carriers in terms of number of PRBs, the number of ports for the LTE CRS, the shifting value for assigning the LTE cell identity, and the LTE carrier offset to NR point A. Set the LTE carrier offset to NR point A in units of 15 kHz subcarriers, as discussed in TS 38.214 Section 5.1.4.2.

```
numSubframes = 10;          % Number of subframes
gnbNumPRB = 52;             % Size of 5G carrier in number of PRBs (1...275)
enbNumPRB = 25;             % Size of LTE carrier in number of PRBs
enbNumCRSPorts = 2;         % Number of LTE CRS ports
enbVShift = 0;              % LTE shifting value v-Shift
enbCarrierFrequency = 0; % LTE carrier offset to NR point A, in units of 15
```

Initialize the random number generator to its default value for reproducibility.

```
rng("default");
```

LTE Configuration

Configure the cell-wide settings for LTE.

```
enb = getLTEConfig(enbNumPRB,enbNumCRSPorts,enbVShift,enbCarrierFrequency,numSubframes);
```

5G Configuration

Configure the cell-wide settings for 5G considering a subcarrier spacing of 15 kHz. This value of the subcarrier spacing ensures that the 5G waveform has the same time-frequency OFDM structure as the LTE waveform.

```
gnb = nrCarrierConfig;
gnb.NSizeGrid = gnbNumPRB;
disp(gnb)
```

```
  nrCarrierConfig with properties:

             NCellID: 1
    SubcarrierSpacing: 15
         CyclicPrefix: 'normal'
            NSizeGrid: 52
           NStartGrid: 0
                NSlot: 0
               NFrame: 0

   Read-only properties:
       SymbolsPerSlot: 14
     SlotsPerSubframe: 1
        SlotsPerFrame: 10
```

5G PDSCH Configuration

Configure the 5G PDSCH to occupy all the resources available and unused by the LTE transmission. In particular, symbol 3 is the first symbol allocated for the PDSCH. This choice is intentionally conservative because the PCFICH, PHICH, and PDCCH in LTE can occupy up to the first three symbols in a subframe.

```
pdsch = nrPDSCHConfig;
pdsch.PRBSet = 0:gnb.NSizeGrid-1;      % Full-band allocation
pdsch.SymbolAllocation = [3 11];       % Full-slot allocation, starting from symbol 3
pdsch.MappingType = "A";               % PDSCH mapping type A
pdsch.DMRS.DMRSTypeAPosition = 3;      % First DM-RS in symbol 3 for PDSCH mapping type A
pdsch.DMRS.DMRSAdditionalPosition = 1; % Additional DM-RS position
```

For a 5G PDSCH mapping type A configured with a DM-RS on symbol 3 and an additional DM-RS, the second DM-RS collides with the LTE CRS on some REs. To avoid collision between the 5G PDSCH DM-RS and the LTE CRS, TS 38.211 Section 7.4.1.1.2 defines an alternative DM-RS location, moving the allocation of the second DM-RS from symbol 11 to symbol 12. The option of using an alternative DM-RS location is subject to UE capability, as specified in TS 38.306 Section 4.2.7.5. The example sets the alternative DM-RS location by using the `CustomSymbolSet` property of the `nrPDSCHDMRSConfig` object.

```
if (pdsch.MappingType == "A") && ...
        (pdsch.DMRS.DMRSAdditionalPosition == 1) && ...
        (pdsch.DMRS.DMRSTypeAPosition == 3)
    pdsch.DMRS.CustomSymbolSet = [3 12];
end
```

Compute the indices occupied by the LTE CRS in the 5G numerology and assign them to the `ReservedRE` property of the `nrPDSCHConfig` object. This property specifies the RE indices that are unavailable for the PDSCH.

```
pdsch.ReservedRE = getReservedRE(enbNumPRB,enbNumCRSPorts,enbVShift,enbCarrierFrequency,gnb);
```

Compute the PRBs occupied by the LTE PBCH and synchronization signals (PSS and SSS) in the 5G numerology and assign them to the `ReservedPRB` property of the `nrPDSCHConfig` object. This property specifies the PRB and symbol patterns that are unavailable for the PDSCH.

```
pdsch.ReservedPRB = getReservedPRB(enb,gnb);
```

Define the power scaling for each channel and signal in dB. The example chooses these values solely to ease the visualization of each channel and signal in the spectrogram view.

```
pdschPower = 15;     % PDSCH power scaling in dB
pdschDMRSPower = 18; % PDSCH DM-RS power scaling in dB
```

LTE Waveform Generation

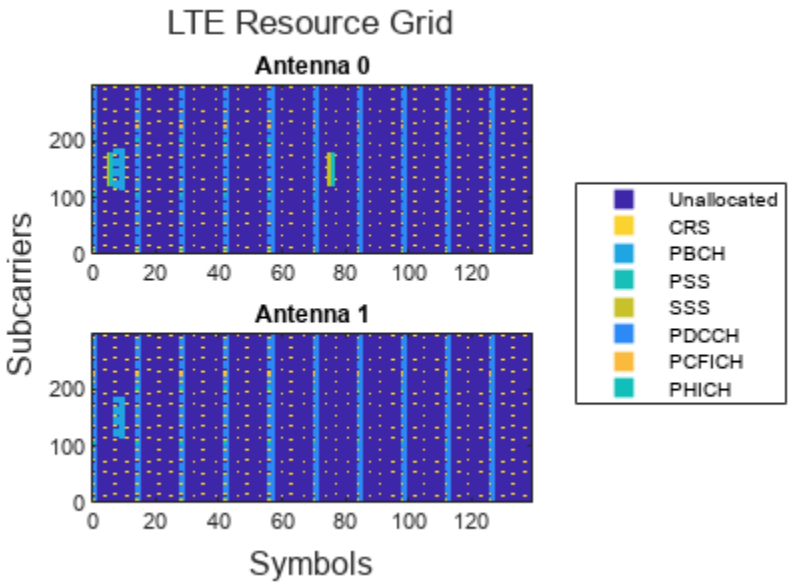
Generate the LTE waveform, resource grid, and information output. The LTE waveform contains:

- Always-on signals CRS, PSS, and SSS.
- Always-on channel PBCH.
- PCFICH, PHICH, and PDCCH.

```
[enbWave,enbGrid] = lteRMCDLTool(enb,[1;0;0;1]);
enbInfo = lteOFDMInfo(enb);
```

Plot the LTE resource grid.

```
enbChannelNames = ["CRS","PBCH","PSS","SSS","PDCCH","PCFICH","PHICH"];
plotResourceGrid(enb,enbGrid,numSubframes,enbChannelNames);
```



5G Waveform Generation

Generate the 5G waveform containing the PDSCH and its DM-RS.

Define the total number of slots.

```
numSlots = numSubframes*gnb.SlotsPerSubframe;
```

Initialize an empty resource grid.

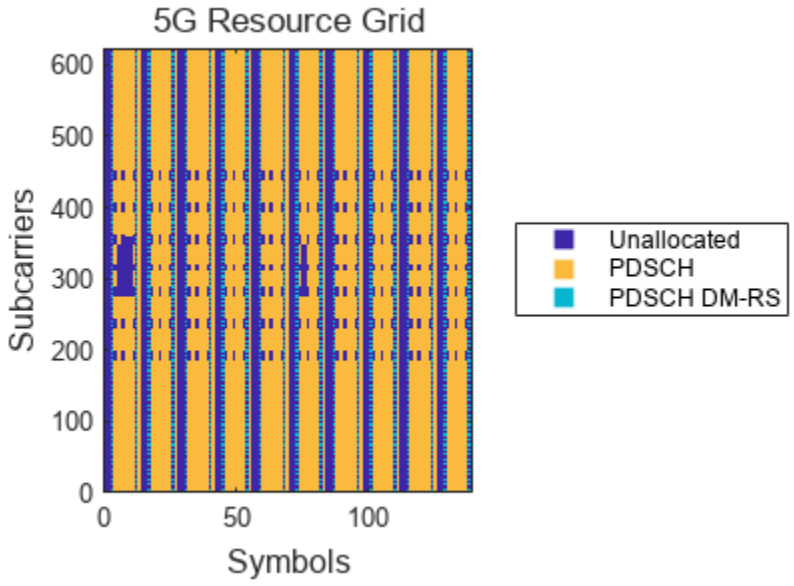
```
gnbGrid = [];
```

Loop over the total number of slots to generate the 5G resource grid.

```
for nslot = 0:numSlots-1

    % Update the carrier slot number and create an empty resource grid for
    % the current slot
    gnb.NSlot = nslot;
    gridSlot = nrResourceGrid(gnb,pdsch.NumLayers);

    % Get the PDSCH and PDSCH DM-RS indices and symbols
    [pdschIndices,pdschIndicesInfo] = nrPDSCHIndices(gnb,pdsch);
    cw = randi([0 1],pdschIndicesInfo.G,1);
    pdschSymbols = nrPDSCH(gnb,pdsch,cw) * db2mag(pdschPower);
    pdschDMRSIndices = nrPDSCHDMRSIndices(gnb,pdsch);
    pdschDMRSSymbols = nrPDSCHDMRS(gnb,pdsch) * db2mag(pdschDMRSPower);

    % Place the symbols in the grid
    gridSlot(pdschIndices) = pdschSymbols;
    gridSlot(pdschDMRSIndices) = pdschDMRSSymbols;

    % Append the resource grid for the current slot to the overall resource
    % grid
    gnbGrid = cat(2,gnbGrid,gridSlot);

end
```

Plot the 5G resource grid.

```
gnbChannelNames = ["PDSCH","PDSCH DM-RS"];
plotResourceGrid(struct("gnb",gnb,"pdsch",pdsch),gnbGrid,numSubframes,gnbChannelNames);
```



Generate the 5G waveform by performing OFDM modulation on the resource grid.

```
[gnbWave,gnbInfo] = nrOFDMModulate(gnb,gnbGrid,Windowing=0);
```

Generate Combined DSS Waveform

Shift the LTE waveform according to the value of `enbCarrierFrequency`.

```
enbfotx = comm.PhaseFrequencyOffset;
enbfotx.SampleRate = enbInfo.SamplingRate;
enbfotx.FrequencyOffset = enbCarrierFrequency*15e3; % Hz
enbWave = enbfotx(enbWave);
```

If LTE and 5G sample rates are different, resample the waveform with the lowest sample rate to the highest sample rate.

```
enbSR = enbInfo.SamplingRate;
gnbSR = gnbInfo.SampleRate;
if gnbSR >= enbSR
    enbWave = resample(enbWave,gnbSR,enbSR);
    ofdmInfo = gnbInfo;
else
    gnbWave = resample(gnbWave,enbSR,gnbSR);
    ofdmInfo = enbInfo;
end
```

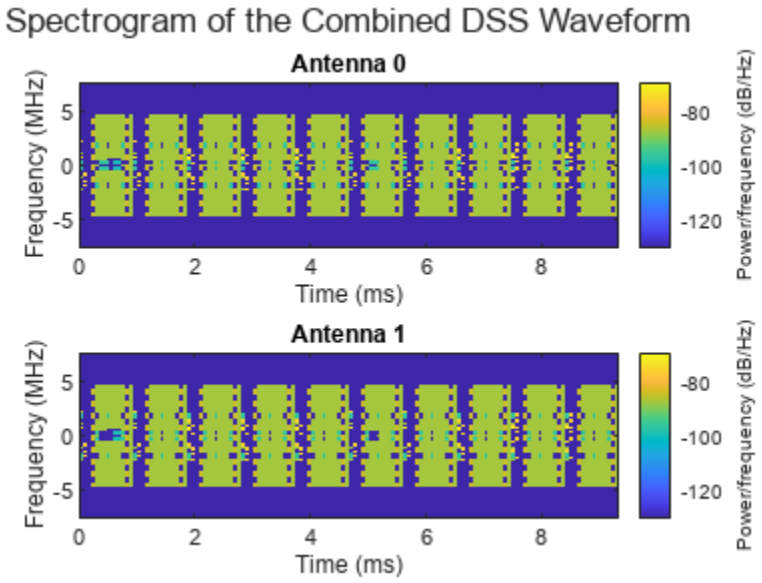
Combine the two waveforms.

```
wave = enbWave + gnbWave;
```

Spectrogram of Combined DSS Waveform

Plot the spectrogram of the combined waveform. You can see the LTE and 5G waveforms coexisting in the same spectrum.

```
plotDSSSpectrogram(wave,ofdmInfo,enbSR,gnbSR,numSubframes);
```



References

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

[2] 3GPP TS 38.214. “NR; Physical layer procedures for data.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[3] 3GPP TS 38.306. “NR; User Equipment (UE) radio access capabilities.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

[4] 3GPP TS 38.331. "NR; Radio Resource Control (RRC) protocol specification.” *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.

Local Functions

```
function enb = getLTEConfig(numPRB,numCRSPorts,vShift,carrierFrequency,numSubframes)
    % Generate the cell-wide configuration for LTE, given the number of
    % PRBs, the number of CRS ports, the shifting value v-Shift for the LTE
    % CRS, and the carrier frequency offset from point A.

    % Define the LTE configuration
    enb = struct();
    enb.NDLRB = numPRB;
    enb.CellRefP = numCRSPorts;
    enb.NCellID = vShift;
    enb.CyclicPrefix = "Normal";
    enb.CFI = 2;
    enb.Ng = "Sixth";
    enb.PHICHDuration = "Normal";
    enb.NFrame = 0;
    enb.NSubframe = 0;
    enb.TotSubframes = numSubframes;
    enb.DuplexMode = "FDD";
    if numCRSPorts>1
        enb.PDSCH.TxScheme = "SpatialMux";
    end
    enb.CarrierFrequency = carrierFrequency;

    % Define the power scaling of LTE signals and channels.
    % These values are chosen solely to ease the visualization of each
    % channel and signal in the spectrogram view.
    enb.PDSCH.PDCCHPower = 24;
    enb.PCFICHPower = 12;
    enb.PHICHPower = 30;
    % To make sure that the LTE waveform contains PDCCH but not PDSCH, set
    % the PDSCH power resource element allocation so that the content of
    % the PDSCH does not affect the LTE waveform
    enb.PDSCH.Rho = -inf;
end

function ind = getReservedRE(numPRB,numCRSPorts,vShift,carrierFrequency,gnb)
    % Compute the REs used by the LTE CRS that the 5G PDSCH needs to
    % rate-match around.

    % Get the LTE configuration for the rate-matching around the CRS
    enb = struct();
    enb.NDLRB = numPRB;
```

```
    enb.CellRefP = numCRSPorts;
    enb.NCellID = vShift;
    enb.CarrierFrequency = carrierFrequency;
    enb.CyclicPrefix = "Normal";
    enb.DuplexMode = "FDD";

    % Get the RE indices for the CRS
    ind = getLTEREIndices(enb,gnb,"CRS");
end

function prbs = getReservedPRB(enb,gnb)
    % Compute the physical resource blocks used by the LTE PBCH, PSS, and
    % SSS that the 5G PDSCH needs to rate-match around.

    % Initialize empty nrPDSCHReservedConfig objects to store the reserved
    % PRBs. The example defines two objects because the periodicity of the
    % PBCH is different from that of the PSS and SSS.
    prbs = {nrPDSCHReservedConfig,nrPDSCHReservedConfig};

    % Compute the 5G BWP grid size
    gnbSize = size(nrResourceGrid(gnb));

    % Get the RE indices for the PBCH and convert them to subscript
    pbchInd = getLTEREIndices(enb,gnb,"PBCH");
    [pbchREInd,pbchSymbInd] = ind2sub(gnbSize,pbchInd);

    % Store the reserved PRBs for the LTE PBCH. Set the period considering
    % that the PBCH is transmitted in subframe 0 of each frame.
    prbInd = unique(floor(pbchREInd/12)); % 0-based PRBs associated to the RE indices
    prbs{1}.PRBSet = prbInd;
    prbs{1}.SymbolSet = unique(pbchSymbInd-1); % 0-based
    prbs{1}.Period = 10*gnb.SlotsPerSubframe;

    % Get the RE indices for the PSS and SSS and convert them to subscript
    ssInd = getLTEREIndices(enb,gnb,["PSS","SSS"]);
    [ssREInd,ssSymbInd] = ind2sub(gnbSize,ssInd);

    % Store the reserved PRBs for the LTE synchronization signals PSS and
    % SSS. Set the period considering that the PSS and SSS are transmitted
    % in subframes 0 and 5 of each frame.
    prbInd = unique(floor(ssREInd/12)); % 0-based PRBs associated to the RE indices
    prbs{2}.PRBSet = prbInd;
    prbs{2}.SymbolSet = unique(ssSymbInd-1); % 0-based
    prbs{2}.Period = 5*gnb.SlotsPerSubframe;
end

function ind = getLTEREIndices(enb,gnb,signalName)
    % Compute the REs used by LTE always-on signals (CRS, PBCH, PSS, and
    % SSS) that the 5G PDSCH needs to rate-match around.

    % enb frequency indices 'enbf' in terms of 15 kHz subcarrier spacing
    enbSize = lteResourceGridSize(enb);
    enbK = enbSize(1);
    enbf = [-enbK/2:-1 1:enbK/2].' + enb.CarrierFrequency;

    % gnb frequency indices 'gnbf' in terms of 15 kHz subcarrier spacing
    gnbSize = size(nrResourceGrid(gnb));
    gnbK = gnbSize(1);
```

```
    gnbf = (-gnbK/2:(gnbK/2 - 1)).';
    gnbf = gnbf(gnb.NStartGrid*12 + (1:gnbK));

    % Compute the CRS indices
    crsIndices = [];
    if any(signalName=="CRS")
        crsIndices = lteCellRSIndices(enb,"sub");
    end
    % Compute the PBCH indices
    pbchIndices = [];
    if any(signalName=="PBCH")
        pbchIndices = ltePBCHIndices(enb,"sub");
    end
    % Compute the PSS indices
    pssIndices = [];
    if any(signalName=="PSS")
        pssIndices = ltePSSIndices(enb,0,"sub");
    end
    % Compute the SSS indices
    sssIndices = [];
    if any(signalName=="SSS")
        sssIndices = lteSSSIndices(enb,0,"sub");
    end
    % Create the list of all indices
    indTot = [crsIndices; pbchIndices; pssIndices; sssIndices];
    enbk = indTot(:,1);
    enbl = indTot(:,2);

    enbf = enbf(enbk);

    gnbk = arrayfun(@(x)find(gnbf==x),enbf,UniformOutput=false);
    z = cellfun(@isempty,gnbk);
    gnbk = cat(1,zeros(0,1),gnbk{:});

    gnbl = enbl;
    gnbl(z) = [];

    ind = sub2ind(gnbSize,gnbk,gnbl);
    ind = ind - 1; % 1-based to 0-based
end

function plotResourceGrid(in,grid,numSubframes,chNames)
    % Plot the resource grid for each antenna.

    if isfield(in,"CellRefP") % LTE
        isLTE = 1;
        nsf = numSubframes;
        enb = in;
        numPorts = enb.CellRefP;
    else % 5G
        isLTE = 0;
        gnb = in.gnb;
        pdsch = in.pdsch;
        nsf = numSubframes* gnb.SlotsPerSubframe;
        numPorts = pdsch.NumLayers;
    end
    chNames = ["Unallocated" chNames];
```

```
% Define channel power levels
chplevel.Unallocated = 0;   % Unallocated REs
chplevel.CRS = 0.9;         % LTE CRS
chplevel.PBCH = 0.39;       % LTE PBCH
chplevel.PSS = 0.51;        % LTE PSS
chplevel.SSS = 0.75;        % LTE SSS
chplevel.PDCCH = 0.3;       % LTE PDCCH
chplevel.PCFICH = 0.83;     % LTE PCFICH
chplevel.PHICH = 0.5;       % LTE PHICH
chplevel.PDSCH = 0.83;      % 5G PDSCH
chplevel.PDSCH_DMRS = 0.45; % 5G PDSCH DM-RS

% Generate the resource grid for plotting
rbGrid = [];
gridSize = size(grid);
gridSize(2) = gridSize(2)/nsf; % Length of a single subframe or slot
for ns = 0:nsf-1 % Loop over each LTE subframe or 5G slot
    % Initialize an empty grid for the current subframe or slot
    rbGridS = zeros(gridSize);

    % Populate the grid
    if isLTE
        enb.NSubframe = mod(ns,10);

        rbGridS(lteCellRSIndices(enb)) = chplevel.CRS;
        rbGridS(ltePSSIndices(enb)) = chplevel.PSS;
        rbGridS(lteSSSIndices(enb)) = chplevel.SSS;
        rbGridS(ltePBCHIndices(enb)) = chplevel.PBCH;
        rbGridS(ltePCFICHIndices(enb)) = chplevel.PCFICH;
        rbGridS(ltePHICHIndices(enb)) = chplevel.PHICH;
        rbGridS(ltePDCCHIndices(enb)) = chplevel.PDCCH;

    else % 5G
        gnb.NSlot = mod(ns,10);

        rbGridS(nrPDSCHIndices(gnb,pdsch)) = chplevel.PDSCH;
        rbGridS(nrPDSCHDMRSIndices(gnb,pdsch)) = chplevel.PDSCH_DMRS;
    end

    % Append the resource grid for the current subframe or slot to the
    % overall resource grid
    rbGrid = cat(2,rbGrid,rbGridS);
end

% Define the colormap and the scaling needed for the plot
cmap = parula(256);
chpval = struct2cell(chplevel);
cscaling = length(cmap);
fnames = strrep(fieldnames(chplevel),"_"," ");
fnames = strrep(fnames,"DMRS","DM-RS");
chpval = chpval(contains(fnames,chNames));
clevels = floor(cscaling*[chpval{:}]);

% Define the plot title
if isLTE
    figTitle = "LTE Resource Grid";
else
    figTitle = "5G Resource Grid";
```

```
    end
    plotTitle = "Antenna ";

    % Define the axes limits for the plot to be 0-based
    minX = 0;
    maxX = size(rbGrid,2)-1;
    minY = 0;
    maxY = size(rbGrid,1)-1;

    % Plot the resource grid for each port or layer
    fh = figure;
    t = tiledlayout(fh,"flow",TileSpacing="Compact");
    for idx = 1:numPorts
        nexttile;
        ax = gca;
        img = image(ax,cscaling*abs(rbGrid(:,:,idx)));
        img.XData = minX:maxX; % 0-based symbol number
        img.YData = minY:maxY; % 0-based subcarrier number
        axis(ax,[minX-0.5 maxX+0.5 minY-0.5 maxY+0.5]);
        axis(ax,"xy");
        if numPorts>1
            title(ax,plotTitle+num2str(idx-1)); % 0-based antenna number
        end
        colormap(ax,cmap);
    end
    title(t,figTitle);
    xlabel(t,"Symbols");
    ylabel(t,"Subcarriers");

    % Create the legend
    N = length(clevels);
    p = struct(Marker="s",LineStyle="none",MarkerSize=8,LineWidth=1.25);
    L = line(ax,NaN(N),NaN(N),p); % Generate markers
    % Index the color map and associate the selected colors with the markers
    c = mat2cell([cmap(1,:);cmap(clevels(2:end),:)],ones(1,N),3);
    set(L,{"color"},c); %#ok<STRSCALR>
    set(L,{"MarkerFaceColor"},c); %#ok<STRSCALR>
    lg = legend(ax,chNames{:});
    lg.Layout.Tile = "East";
end

function plotDSSSpectrogram(wave,ofdmInfo,enbSR,gnbSR,numSubframes)
    % Plot the spectrogram of the combined waveform.

    % To minimize the artifacts shown in the spectrogram and due to out of
    % band leakage of the resampled waveform, adjust the waveform to
    % consider the filter response.
    cpLengths = double(ofdmInfo.CyclicPrefixLengths);
    OSR = max(enbSR/gnbSR,gnbSR/enbSR);
    n = min(10,floor(max(cpLengths)/(2*OSR)));
    order = 2*n*OSR; % Order of the filter used during resampling
    wave = circshift(wave,order/2);

    fh = figure;
    t = tiledlayout(fh,"flow",TileSpacing="Compact");
    % Compute the cyclic prefixes to remove from the waveform for each
    % subframe.
    nfft = double(ofdmInfo.Nfft);
```

```
    if isfield(ofdmInfo,'SampleRate')
        sampleRate = ofdmInfo.SampleRate;
    else
        sampleRate = ofdmInfo.SamplingRate;
    end
    symbolLengths = nfft+cpLengths;
    cpidxs = arrayfun(@(x,y)(x+(1:y)),cumsum([0 symbolLengths(1:end-1)]),cpLengths,UniformOutput=
    cpidxs = [cpidxs{:}];
    sfWaveLength = sampleRate/1000;

    % Loop over each antenna
    for idx = 1:size(wave,2)
        nexttile;
        txWaveNoCP = [];
        for nsf = 1:numSubframes
            waveNoCP = wave(1+(nsf-1)*sfWaveLength:nsf*sfWaveLength,idx);
            waveNoCP(cpidxs) = []; % Remove the cyclic prefix in this subframe
            txWaveNoCP = cat(1,txWaveNoCP, waveNoCP);
        end
        spectrogram(txWaveNoCP,ones(nfft,1),0,nfft,"centered",sampleRate,"yaxis",MinThreshold=-1
        title(['Antenna ' num2str(idx-1)]); % 0-based antenna number
    end
    title(t,"Spectrogram of the Combined DSS Waveform");
end
```

See Also

Functions
`lteRMCDLTool`

Objects
`nrPDSCHConfig`

Related Examples

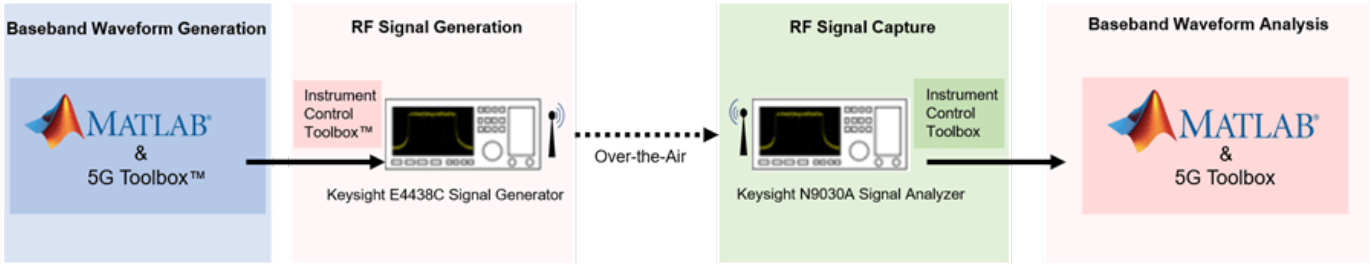
- “5G NR Downlink Vector Waveform Generation” on page 1-2
- “5G NR Uplink Vector Waveform Generation” on page 2-2
- “5G NR-TM and FRC Waveform Generation” on page 7-12

5G NR Waveform Acquisition and Analysis

This example shows how to generate a 5G NR test model (NR-TM) waveform using the 5G Waveform Generator app and download the generated waveform to a Keysight™ vector signal generator for over-the-air transmission using the Instrument Control Toolbox™ software. The example then captures the transmitted over-the-air signal using a Keysight signal analyzer and analyzes the signal in MATLAB®.

Introduction

This example generates a 5G NR-TM waveform using the **5G Waveform Generator** app, downloads and transmits the waveform onto a Keysight vector signal generator, and then receives the waveform using a Keysight signal analyzer for waveform analysis in MATLAB. This diagram shows the general workflow.



Requirements

To run this example, you need these instruments:

- Keysight E4438C ESG vector signal generator
- Keysight N9030A PXA signal analyzer

Generate Baseband Waveform Using 5G Waveform Generator App

In MATLAB, on the **Apps** tab, click the **5G Waveform Generator** app.

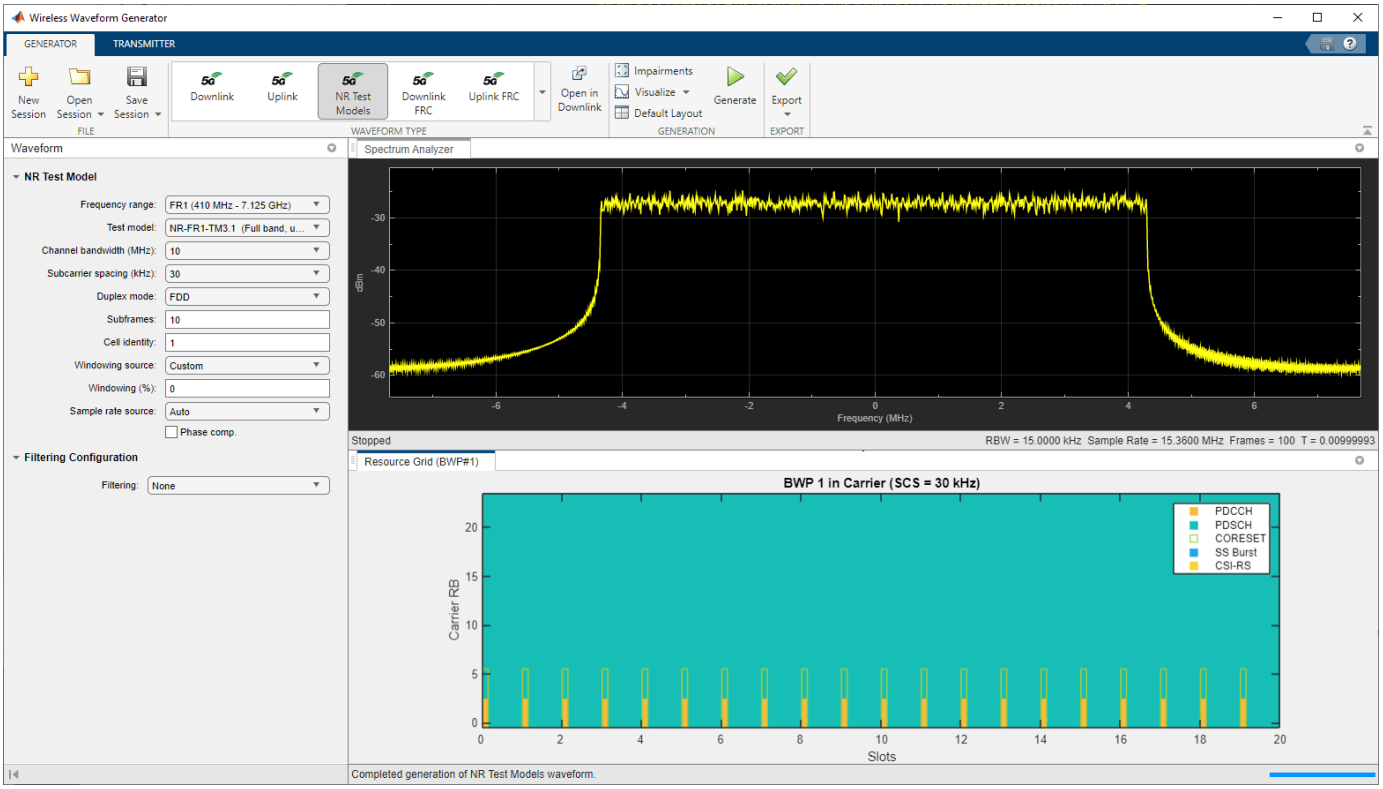
In the **Waveform Type** section, click **NR Test Models.** In the left-most pane of the app, you can set the parameters for the selected waveform. For this example:

- Set **Frequency range** as `FR1 (410 MHz - 7.125 GHz)`
- Set **Test model** as `NR-FR1-TM3.1 (Full band, uniform 64 QAM)`
- Set **Channel bandwidth (MHz)** as `10`
- Set **Subcarrier spacing (kHz)** as `30`
- Set **Duplex mode** as `FDD`
- Set **Subframes** as `10`

On the app toolstrip, click **Generate**.

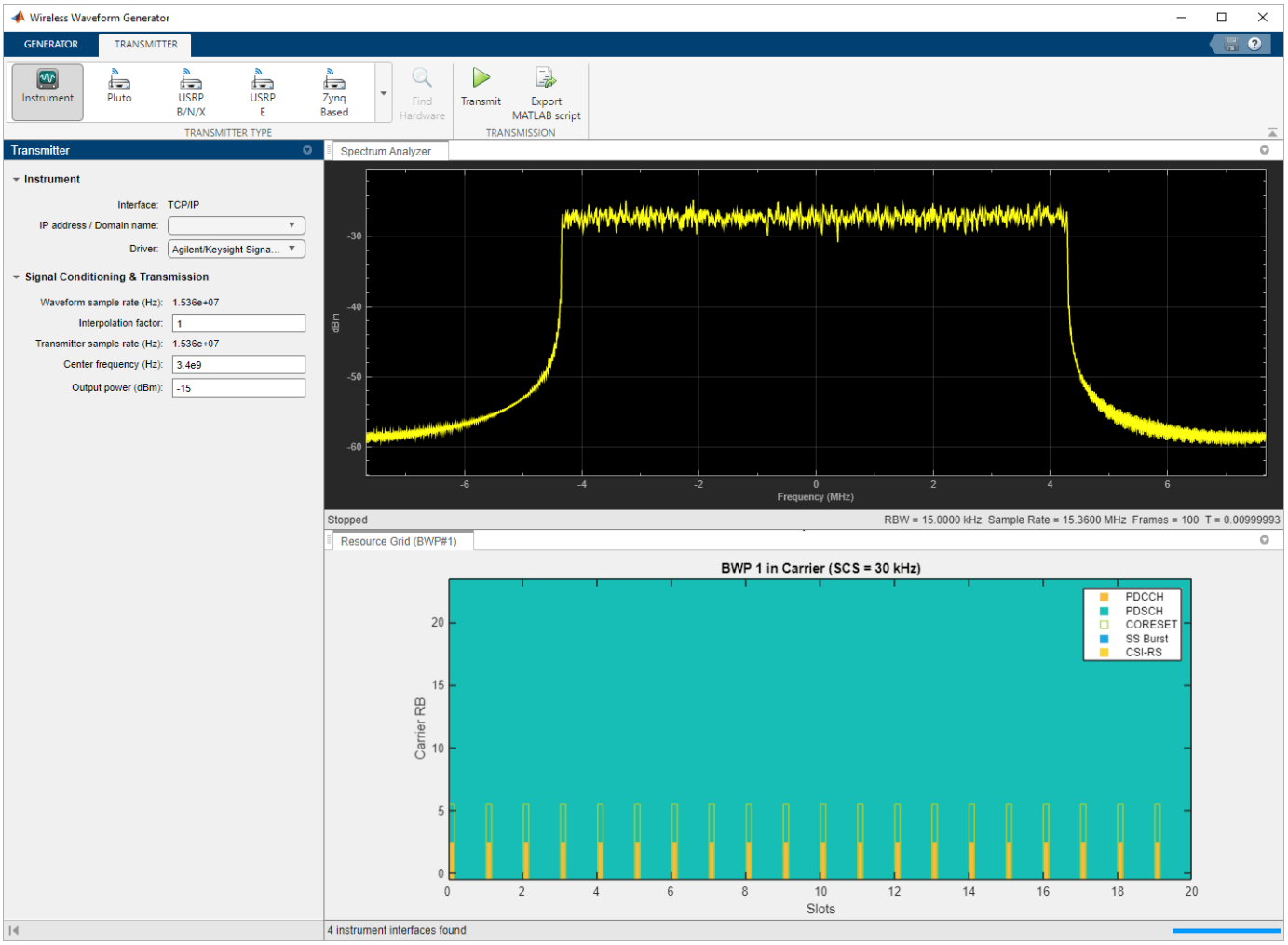
```
% Set the NR-TM parameters for the receiver
nrtm = "NR-FR1-TM3.1"; % Reference channel
bw   = "10MHz";        % Channel bandwidth
scs  = "30kHz";        % Subcarrier spacing
dm   = "FDD";          % Duplexing mode
```

This figure shows a 10 MHz 5G NR waveform visible at baseband.



Transmit Over-the-Air Signal

Download the generated signal to the RF signal generator over one of the supported communication interfaces (requires Instrument Control Toolbox). The app automatically finds the signal generator that is connected over the TCP/IP interface. On the **Transmitter** tab of the app, select `Agilent/Keysight Signal Generator SCPI` from the **Driver** list. Set the **Center frequency (Hz)** parameter to `3.4e9` and the **Output power (dBm)** parameter to `-15`. The app automatically obtains the baseband sample rate from the generated waveform. To start the transmission, click **Transmit** in the toolstrip.



Read IQ Data from a Signal Analyzer over TCP/IP

To read the in-phase and quadrature (IQ) data into MATLAB for analysis, configure the Keysight N9030A signal analyzer using the Instrument Control Toolbox software.

Define the instrument configuration parameters based on the signal you are measuring.

```
% Set parameters for the spectrum analyzer
centerFrequency = 3.4e9;
sampleRate = 15.36e6;
measurementTime = 20e-3;
mechanicalAttenuation = 0; %dB
startFrequency = 3.39e9;
stopFrequency = 3.41e9;
resolutionBandwidth = 220e3;
videoBandwidth = 220000;
```

Perform these steps before connecting to the spectrum analyzer.

- Find the resource ID of the Keysight N9030A signal analyzer.
- Connect to the instrument using the virtual instrument software architecture (VISA) interface.

- Adjust the input buffer size to hold the data that the instrument returns.
- Set the timeout to allow sufficient time for the measurement and data transfer.

```
foundVISA = visadevlist;
resourceID = foundVISA(foundVISA.Model == "N9030A",:).ResourceName;
resourceID = resourceID(contains(resourceID,"N9030A"));             % Extract resourceID which co
sigAnalyzerObj = visadev(resourceID);
sigAnalyzerObj.ByteOrder = "big-endian";
sigAnalyzerObj.Timeout = 20;
```

Reset the instrument to a known state using the appropriate standard command for programmable instruments (SCPI). Query the instrument identity to ensure the correct instrument is connected.

```
writeline(sigAnalyzerObj,"*RST");
instrumentInfo = writeread(sigAnalyzerObj,"*IDN?");
fprintf("Instrument identification information: %s",instrumentInfo);
```

```
Instrument identification information: Agilent Technologies,N9030A,US00071181,A.14.16
```

The X-Series signal and spectrum analyzers perform IQ measurements as well as spectrum measurements. In this example, you acquire time domain IQ data, visualize the data using MATLAB, and perform signal analysis on the acquired data. The SCPI commands configure the instrument and define the format of the data transfer after the measurement is complete.

```
% Set up signal analyzer mode to basic IQ mode
writeline(sigAnalyzerObj,":INSTrument:SELect BASIC");

% Set the center frequency
writeline(sigAnalyzerObj,strcat(":SENSe:FREQuency:CENTer ",num2str(centerFrequency)));

% Set the capture sample rate
writeline(sigAnalyzerObj,strcat(":SENSe:WAVeform:SRATe ",num2str(sampleRate)));

% Turn off averaging
writeline(sigAnalyzerObj,":SENSe:WAVeform:AVER OFF");

% Set the spectrum analyzer to take one single measurement after the trigger line goes high
writeline(sigAnalyzerObj,":INIT:CONT OFF");

% Set the trigger to external source 1 with positive slope triggering
writeline(sigAnalyzerObj,":TRIGger:WAVeform:SOURce IMMediate");
writeline(sigAnalyzerObj,":TRIGger:LINE:SLOPe POSitive");

% Set the time for which measurement needs to be made
writeline(sigAnalyzerObj,strcat(":WAVeform:SWE:TIME ",num2str(measurementTime)));

% Turn off electrical attenuation
writeline(sigAnalyzerObj,":SENSe:POWer:RF:EATTenuation:STATe OFF");

% Set the mechanical attenuation level
writeline(sigAnalyzerObj,strcat(":SENSe:POWer:RF:ATTenuation ",num2str(mechanicalAttenuation)));

% Turn IQ signal ranging to auto
writeline(sigAnalyzerObj,":SENSe:VOLTage:IQ:RANGe:AUTO ON");

% Set the endianness of returned data
writeline(sigAnalyzerObj,":FORMat:BORDer NORMal");
```

```
% Set the format of the returned data
writeline(sigAnalyzerObj,":FORMat:DATA REAL,64");
```

Trigger the instrument to make the measurement. Wait for the measurement operation to complete, and then read-in the waveform. Before processing the data, separate the I and Q components from the interleaved data that is received from the instrument and create a complex vector in MATLAB.

```
% Trigger the instrument and initiate measurement
writeline(sigAnalyzerObj,"*TRG");
writeline(sigAnalyzerObj,":INITiate:WAVeform");

% Wait until measure operation is complete
measureComplete = writeread(sigAnalyzerObj,"*OPC?");

% Read the IQ data
writeline(sigAnalyzerObj,":READ:WAV0?");
data = readbinblock(sigAnalyzerObj,"double");

% Separate the data and build the complex IQ vector
inphase = data(1:2:end);
quadrature = data(2:2:end);
rxWaveform = inphase+1i*quadrature;
```

Capture and display the information about the most recently acquired data.

```
writeline(sigAnalyzerObj,":FETCH:WAV1?");
signalSpec = readbinblock(sigAnalyzerObj,"double");

% Display the measurement information
captureSampleRate = 1/signalSpec(1);
fprintf("Sample Rate (Hz) = %s",num2str(captureSampleRate));
```

```
Sample Rate (Hz) = 15360000
```

```
fprintf("Number of points read = %s",num2str(signalSpec(4)));
```

```
Number of points read = 307201
```

```
fprintf("Max value of signal (dBm) = %s",num2str(signalSpec(6)));
```

```
Max value of signal (dBm) = -43.1954
```

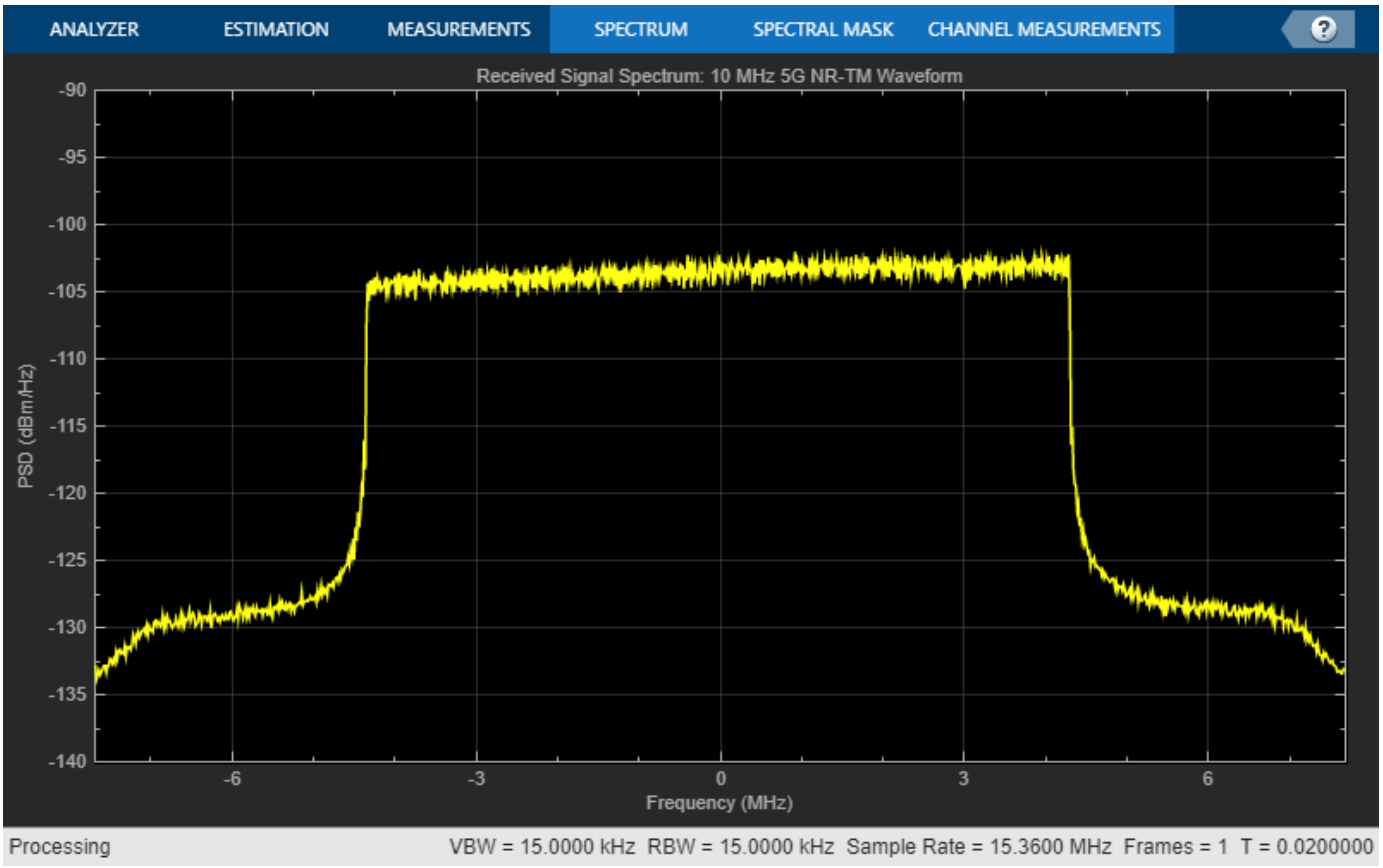
```
fprintf("Min value of signal (dBm) = %s",num2str(signalSpec(7)));
```

```
Min value of signal (dBm) = -104.8862
```

Plot the spectrum of the acquired waveform to confirm the bandwidth of the received signal.

```
% Ensure rxWaveform is a column vector
if ~iscolumn(rxWaveform)
    rxWaveform = rxWaveform.';
end

% Plot the power spectral density (PSD) of the acquired signal
spectrumPlotRx = spectrumAnalyzer;
spectrumPlotRx.SampleRate =  captureSampleRate;
spectrumPlotRx.SpectrumType = "Power density";
spectrumPlotRx.YLimits = [-140 -90];
spectrumPlotRx.YLabel = "PSD";
```

```
spectrumPlotRx.Title = "Received Signal Spectrum: 10 MHz 5G NR-TM Waveform";
spectrumPlotRx(rxWaveform);
```



Switch the instrument to spectrum analyzer mode and compare the spectrum view generated in MATLAB with the view on the signal analyzer. Use additional SCPI commands to configure the instrument measurement and display settings.

```
% Switch back to the spectrum analyzer view
writeline(sigAnalyzerObj,":INSTrument:SELect SA");

% Set the mechanical attenuation level
writeline(sigAnalyzerObj,strcat(":SENSe:POWer:RF:ATTenuation ",num2str(mechanicalAttenuation)));

% Set the center frequency, RBW, and VBW
writeline(sigAnalyzerObj,strcat(":SENSe:FREQuency:CENTer ",num2str(centerFrequency)));
writeline(sigAnalyzerObj,strcat(":SENSe:FREQuency:STARt ",num2str(startFrequency)));
writeline(sigAnalyzerObj,strcat(":SENSe:FREQuency:STOP ",num2str(stopFrequency)));
writeline(sigAnalyzerObj,strcat(":SENSe:BANDwidth:RESolution ",num2str(resolutionBandwidth)));
writeline(sigAnalyzerObj,strcat(":SENSe:BANDwidth:VIDeo ",num2str(videoBandwidth)));

% Enable continuous measurement on the spectrum analyzer
writeline(sigAnalyzerObj,":INIT:CONT ON");

% Begin receiving the over-the-air signal
writeline(sigAnalyzerObj,"*TRG");
```

For instrument cleanup, clear the instrument connection:

```
clear sigAnalyzerObj;
```

To stop the 5G NR-TM waveform transmission, in the **Instrument** section on the app toolstrip, click **Stop Transmission**.

Perform Measurements of Received 5G Waveform

Use the `generateWaveform` function of the `hNRReferenceWaveformGenerator` helper file to extract the waveform information for a specific TM.

```
tmwavegen = hNRReferenceWaveformGenerator(nrtm,bw,scs,dm);
[~,tmwaveinfo,resourcesInfo] = generateWaveform(tmwavegen);
```

Coarse Frequency Offset Compensation Using Demodulation Reference Symbols (DM-RS)

Look for offsets in increments of 1 kHz up to 100 kHz.

```
frequencyCorrectionRange = -100e3:1e3:100e3;
[rxWaveform, coarseOffset] = DMRSFrequencyCorrection(rxWaveform,captureSampleRate,frequencyCorre
fprintf("Coarse frequency offset = %.0f Hz", coarseOffset)
```

```
Coarse frequency offset = 0 Hz
```

Fine Frequency Offset Compensation Using DM-RS

Look for offsets in increments of 5 Hz up to 100 Hz

```
frequencyCorrectionRange = -100:5:100;
[rxWaveform, fineOffset] = DMRSFrequencyCorrection(rxWaveform,captureSampleRate,frequencyCorrecti
fprintf("Fine frequency offset = %.1f Hz", fineOffset)
```

```
Fine frequency offset = -25.0 Hz
```

EVM Measurements

Use the `hNRPDSCHEVM` function to analyze the waveform. The function performs these steps.

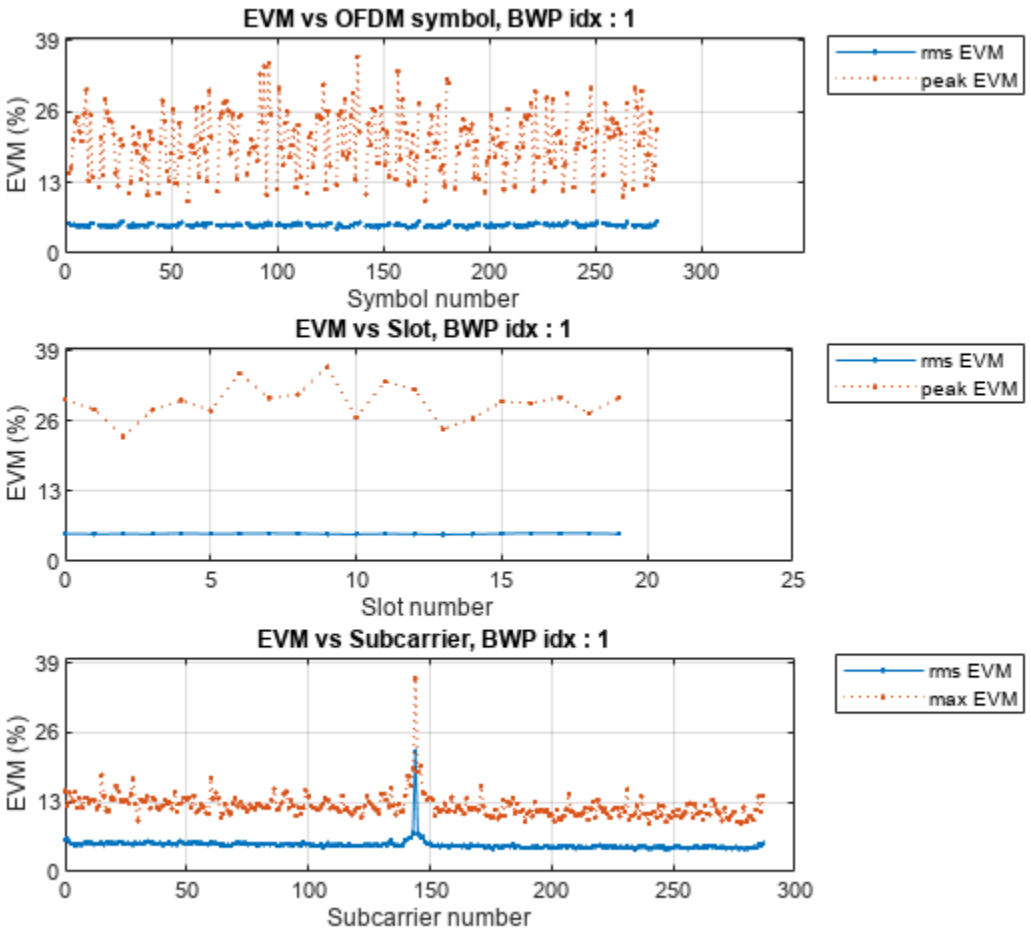
- Synchronizes the DM-RS over one frame for frequency division duplexing (FDD) (two frames for time division duplexing (TDD))
- Demodulates the received waveform
- Estimates the channel
- Equalizes the symbols
- Estimates and compensates for common phase error (CPE)

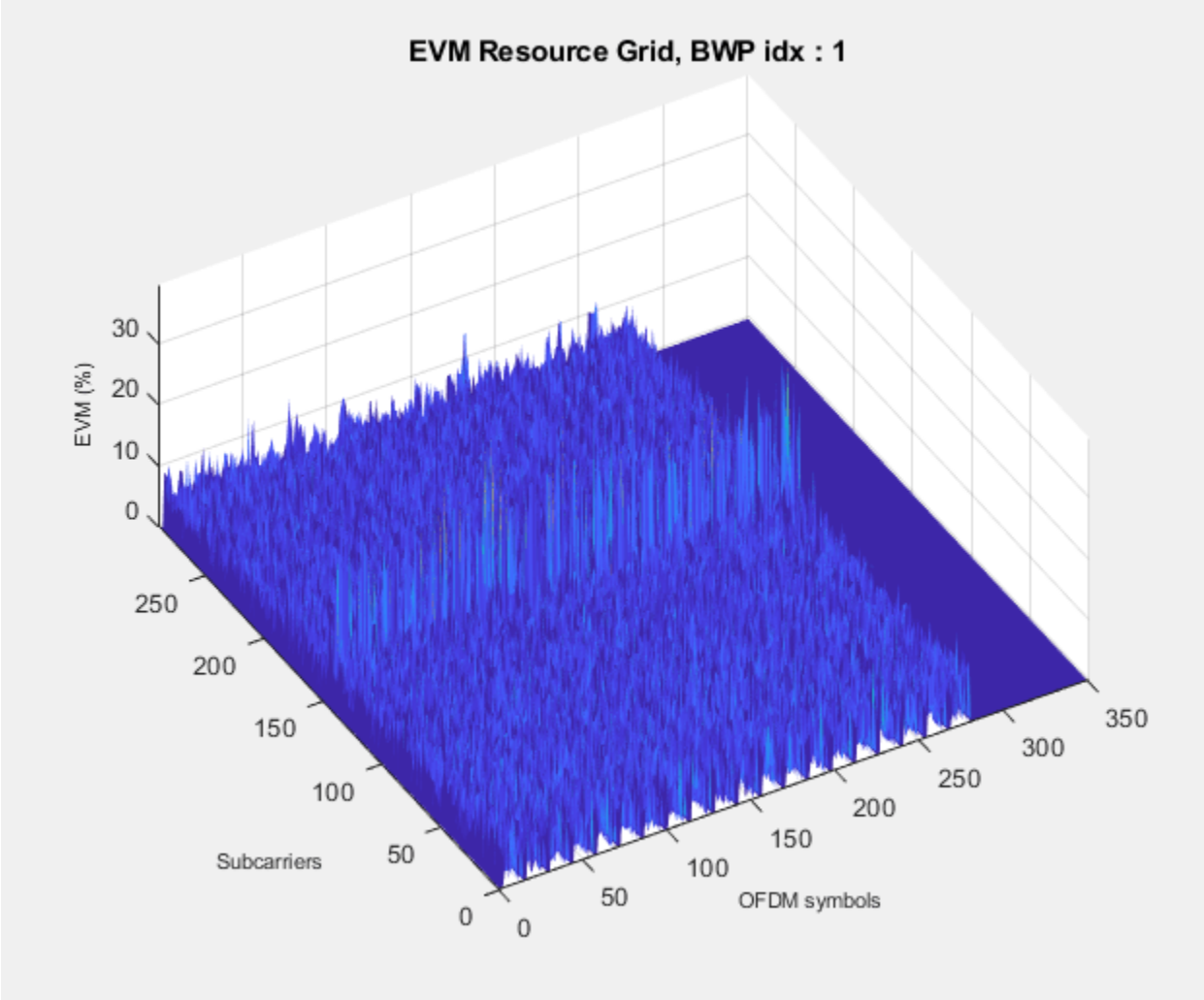
Define the configuration settings for the `hNRPDSCHEVM` function.

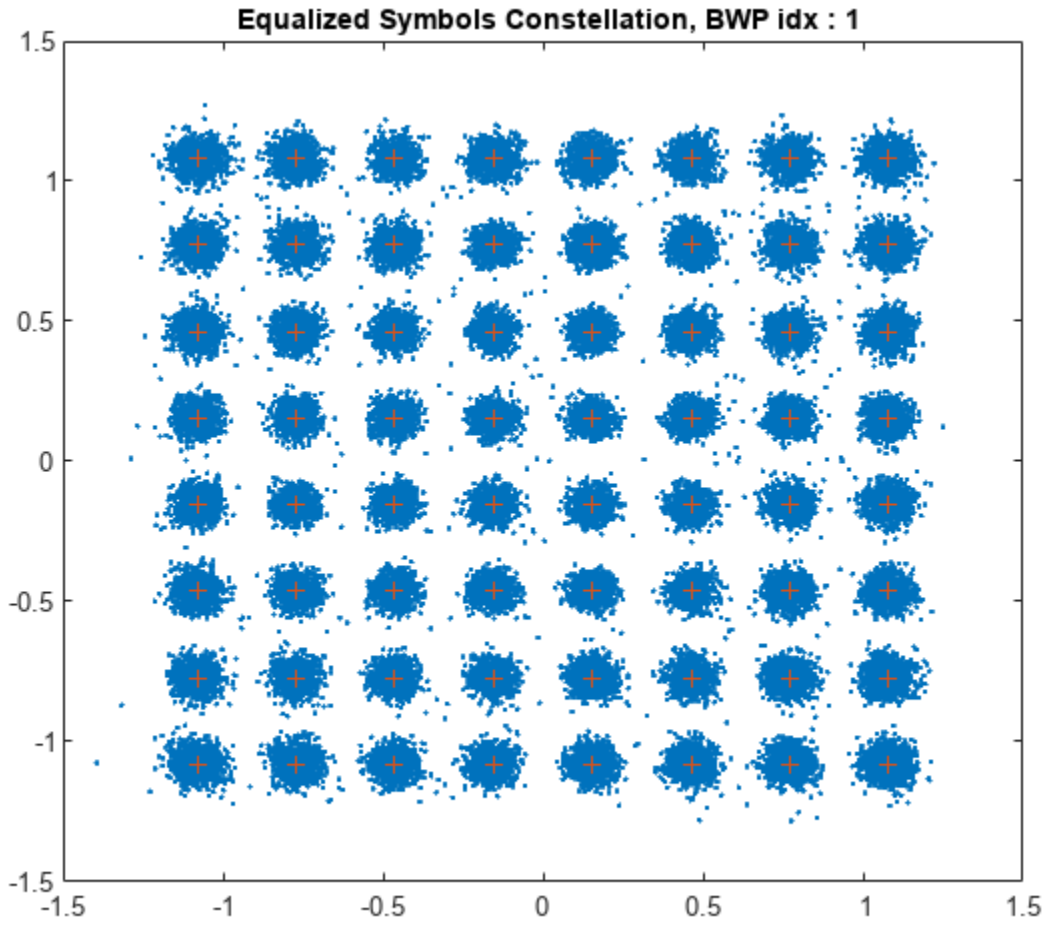
```
cfg = struct();
cfg.PlotEVM = true;                % Plot EVM statistics
cfg.DisplayEVM = true;             % Print EVM statistics
cfg.Label = nrtm;                  % Set to TM name of captured waveform
cfg.SampleRate = captureSampleRate; % Use sample rate during capture
```

```
[evmInfo,eqSym,refSym] = hNRPDSCHEVM(tmwavegen.Config,rxWaveform,cfg);
```

```
EVM stats for BWP idx : 1
 RMS EVM, Peak EVM, slot 0: 5.141 29.966%
 RMS EVM, Peak EVM, slot 1: 5.117 28.236%
 RMS EVM, Peak EVM, slot 2: 5.129 23.172%
 RMS EVM, Peak EVM, slot 3: 5.118 28.065%
 RMS EVM, Peak EVM, slot 4: 5.218 29.877%
 RMS EVM, Peak EVM, slot 5: 5.136 27.957%
 RMS EVM, Peak EVM, slot 6: 5.179 34.915%
 RMS EVM, Peak EVM, slot 7: 5.220 30.278%
 RMS EVM, Peak EVM, slot 8: 5.204 30.946%
 RMS EVM, Peak EVM, slot 9: 5.089 36.078%
 RMS EVM, Peak EVM, slot 10: 5.063 26.745%
 RMS EVM, Peak EVM, slot 11: 5.140 33.408%
 RMS EVM, Peak EVM, slot 12: 5.101 31.880%
 RMS EVM, Peak EVM, slot 13: 5.017 24.540%
 RMS EVM, Peak EVM, slot 14: 5.068 26.508%
 RMS EVM, Peak EVM, slot 15: 5.180 29.606%
 RMS EVM, Peak EVM, slot 16: 5.259 29.391%
 RMS EVM, Peak EVM, slot 17: 5.234 30.314%
 RMS EVM, Peak EVM, slot 18: 5.229 27.510%
 RMS EVM, Peak EVM, slot 19: 5.136 30.313%
Averaged RMS EVM frame 0: 5.149%
```


EVM vs OFDM symbol, BWP idx : 1
EVM (%)
Symbol number
rms EVM
peak EVM
EVM vs Slot, BWP idx : 1
EVM (%)
Slot number
rms EVM
peak EVM
EVM vs Subcarrier, BWP idx : 1
EVM (%)
Subcarrier number
rms EVM
max EVM


EVM Resource Grid, BWP idx : 1
EVM (%)
30
20
10
0
250
200
150
100
50
0
Subcarriers
0
50
100
150
200
250
300
350
OFDM symbols




```
Averaged overall RMS EVM: 5.149%
Overall Peak EVM = 36.0778%
```

The measurements show that the demodulation of the received waveform is successful. The interference from the DC component of the spectrum analyzer to the DC subcarrier causes high EVM values in the measurements.

Local functions

These functions assist in processing the received 5G waveform.

```
function [correctedWaveform,appliedFrequencyCorrection] = DMRSFrequencyCorrection(waveform,sample
    % waveform - Waveform to be corrected. Needs to be a Nx1 column vector.
    % sampleRate - Sample rate of waveform
    % frequencyCorrectioRange - Range and granularity at which frequency
    % correction is inspected
    % tmwavegen and resourcesInfo - Outputs of generateWaveform method
    [pdschArray,~,carrier] = hListTargetPDSCHs(tmwavegen.Config,resourcesInfo.WaveformResources)
    bwpCfg = tmwavegen.Config.BandwidthParts{1,1};
    nSlots = carrier.SlotsPerFrame;
```

```
    % Generate a reference grid spanning 10 ms (one frame). This grid
    % contains only the DM-RS and is used for synchronization.
    refGrid = referenceGrid(carrier,bwpCfg,pdschArray,nSlots);

    % Apply frequency offsets to the waveform as specified by
    % freuqnecyCorrectionRange.
    nSamples = (0:length(waveform)-1)';
    frequencyShift = (2*pi*frequencyCorrectionRange.*nSamples)./sampleRate;

    % Each column represents an offset waveform.
    offsetWaveforms = waveform.*exp(1j*frequencyShift);

    [~,mag] = nrTimingEstimate(offsetWaveforms,carrier.NSizeGrid,...
        carrier.SubcarrierSpacing,nSlots,refGrid, ...
        "SampleRate",sampleRate);

    % Find the frequency at which the DM-RS correlation is at a maximum.
    [~,index] = max(max(mag));
    appliedFrequencyCorrection = frequencyCorrectionRange(index);
    correctedWaveform = offsetWaveforms(:,index);
end

function refGrid = referenceGrid(carrier,bwpCfg,pdschArray,nSlots)
    % Create a reference grid for the required number of slots. The grid
    % contains the DM-RS symbols specified in pdschArray. The function
    % returns REFGRID of dimensions K-by-S-by-L, where K is the number of
    % subcarriers of size carrier.NSizeGrid, S is the number of symbols
    % spanning nSlots, and L is the number of layers.

    nSubcarriers = carrier.NSizeGrid * 12;
    L = carrier.SymbolsPerSlot*nSlots;                          % Number of OFDM symbols in the
    nLayers = size(pdschArray(1).Resources(1).ChannelIndices,2);
    bwpStart = bwpCfg.NStartBWP;
    bwpLen = bwpCfg.NSizeBWP;
    refGrid = zeros(nSubcarriers,L,nLayers);                    % Empty grid
    bwpGrid = zeros(bwpLen*12,L,nLayers);
    rbsPerSlot = bwpLen*12*carrier.SymbolsPerSlot;

    % Populate the DM-RS symbols in the reference grid for all slots. Place
    % bwpGrid in a carrier grid (at an appropriate location) in case the
    % BWP size is not the same as the carrier grid
    for slotIdx = carrier.NSlot + (0:nSlots-1)
        [~,~,dmrsIndices,dmrsSymbols] = hSlotResources(pdschArray,slotIdx);
        if ~isempty(dmrsIndices)
            for layerIdx = 1:nLayers
                if layerIdx <= size(dmrsIndices,2)
                    dmrsIndices(:,layerIdx) = dmrsIndices(:,layerIdx) - rbsPerSlot*(layerIdx -1)
                    bwpGrid(dmrsIndices(:,layerIdx)+(slotIdx-carrier.NSlot)*rbsPerSlot) = dmrsSy
                end
            end
            refGrid(12*bwpStart+1:12*(bwpStart+bwpLen),:,:) = bwpGrid;
        end
```

```
    end
end
```

See Also

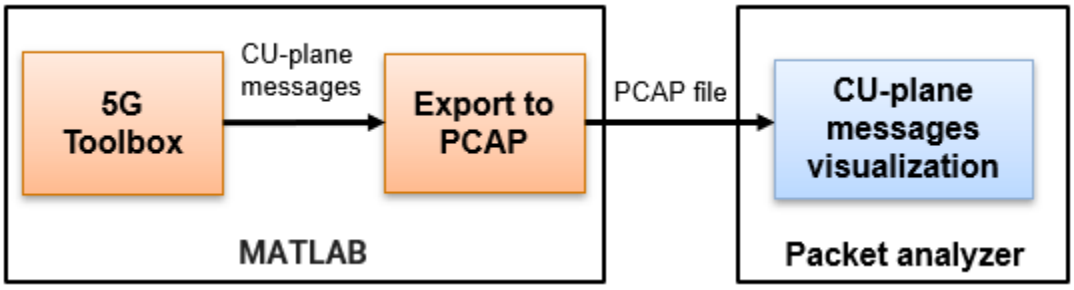
Apps
5G Waveform Generator

Generate CU-Plane Messages for O-RAN Fronthaul Test

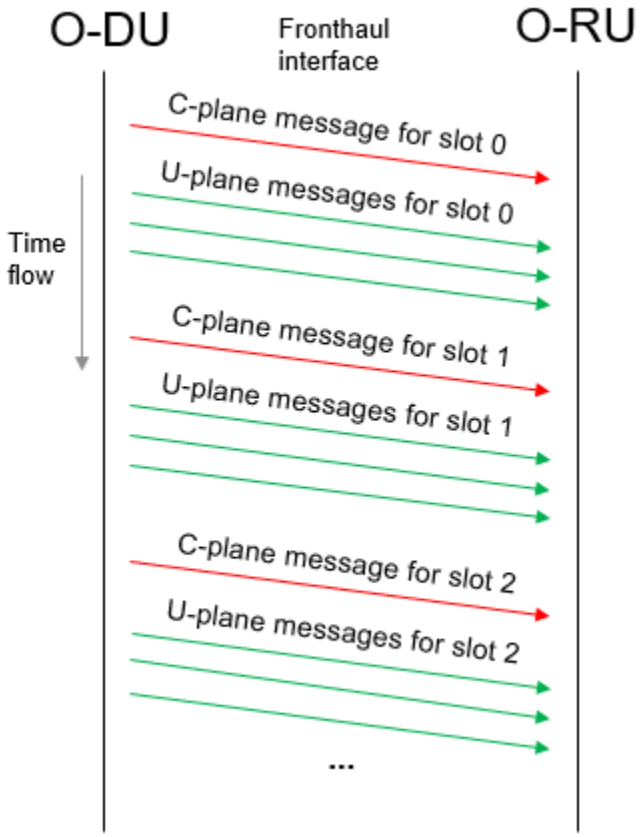
This example shows how to generate fronthaul control and user (CU) plane messages for open radio access network (O-RAN) conformance tests using 5G Toolbox™. These O-RAN compliant messages are considered as split option 7.2x test vectors generated from an O-RAN distributed unit (O-DU). The example generates a packet capture (PCAP) file that contains these messages.

Introduction

This example shows how to build the O-RAN fronthaul CU-plane messages, as defined in TS O-RAN.WG4.CUS, which transmit an NR test model waveform, as defined in TS 38.141-1. These O-RAN compliant messages are considered as split option 7.2x test vectors generated from an O-DU. The example generates a PCAP file, which contains the messages. You can use the generated packets to test an O-RAN radio unit (O-RU) following the conformance test specifications in TS O-RAN.WG4.CONF. You can also analyze the generated PCAP file with third-party packet analysis tools. In this example, Wireshark is used to verify that the content of the CU-plane messages is as expected.



This example builds the CU-plane messages required to transmit the specified NR test model waveform. The example generates a single C-plane message per slot, using Section Type 1, and a single U-plane message per symbol. The U-plane messages encapsulate the IQ data using a single section per symbol. The following diagram shows the flow of the generated CU-plane messages.



Set Configuration Parameters

The data frame to transmit consists of a full band 5G NR test model waveform, as defined in TS 38.141-1. You can set the test model, the channel bandwidth, and the subcarrier spacing of the test waveform. This example supports FDD duplex mode only and does not apply any precoding or beamforming.

The generated PCAP file includes the Ethernet, eCPRI, and O-RAN protocols. In this section, you can configure parameters available in the three protocols:

- **O-RAN** — Set the compression method and the IQ samples bit-width before and after compression.
- **eCPRI** — Set the extended antenna-carrier identifier (eAxC ID) fields. All packets have the same eAxC ID because this example supports only a single stream.
- **Ethernet** — Set the Ethernet VLAN tag, and the MAC source and destination addresses.

This example does not generate management plane (M-plane) messages. However, the example enables you to set these M-plane parameters: compression mode, byte order, and number of bits per field in eAxC ID.

```
% Select the NR test waveform parameters
waveConfig.tm = NR-FR1-TM1.1 (...;  % Test model (must be full band)
waveConfig.bw = 100MHz (FR1 & FR2);  % Channel bandwidth in MHz
```

```
waveConfig.scs = 30kHz (FR1); % Subcarrier spacing in kHz

% O-RAN configuration
% Select the compression parameters
oranConfig.method = BFP; % Compression method
oranConfig.IQWidth = 16;                % IQ samples bit-width before compression
oranConfig.cIQWidth = 9;                % Compressed IQ samples bit-width (1 to 16)

% eCPRI configuration
% Select the eAxC ID fields
eCPRIConfig.DUPortID = 0;      % Distributed unit identifier
eCPRIConfig.BandSectorID = 0;  % Band and sector identifier
eCPRIConfig.CCID = 0;          % Component carrier identifier
eCPRIConfig.RUPortID = 0;      % Spatial stream identifier

% Ethernet configuration
% Select the Ethernet VLAN tag
ethernetConfig.TPID = 0x8100; % Tag protocol identifier
ethernetConfig.priority = 7;              % Priority level (0 to 7)
ethernetConfig.DEI = Not eligible to be dr...;  % Drop eligible indicator
ethernetConfig.VID = 1;                   % Unique VLAN identifier (0 to 4095)
% Select the MAC source and destination addresses
ethernetConfig.sourceAddress = ["56","3b","be","a9","92","4c"]; % MAC source address
ethernetConfig.destAddress = ["da","14","de","b0","55","63"];   % MAC destination address

% Select the M-plane parameters
mPlaneConfig.compMode = Dynamic;  % Compression mode
mPlaneConfig.byteOrder = Big endian; % Byte order
mPlaneConfig.eAxCIDBits = [2 6 4 4];          % Number of bits per field in eAxC ID

% Set the name of the PCAP file
pcapFileName = 'CUMessages'; % PCAP file name
```

Generate U-Plane Data

Use the `hNRReferenceWaveformGenerator` class to generate the complete 5G NR configuration of the frame that you selected in the previous section. The resource grid of the NR frame (split option 7.2x) is the data carried in the U-plane messages.

```
% Create waveform generator object
waveConfig.dm = "FDD"; % FDD duplexing mode
tmWaveGen = hNRReferenceWaveformGenerator(waveConfig.tm,waveConfig.bw, ...
    waveConfig.scs,waveConfig.dm);

% Generate waveform and get resource grid of one frame of data
[~,gridSet] = generateWaveform(tmWaveGen);
grid = gridSet.ResourceGridBWP;
```

Compress U-Plane Data

Next, compress the generated resource by using the `nrORANBlockCompress` function. Compression method options include block floating point (BFP), block scaling, and mu-law, as defined in TS O-

RAN.WG4.CUS Annex A.1.1, A.2.1, and A.3.1, respectively. Before applying the compression, scale the IQ samples in the resource grid to the bit-width specified by `oranConfig.IQWidth`.

```
% Scale the IQ samples in the resource grid using IQWidth
peak = max(abs([real(grid(:)); imag(grid(:))]));
scaleFactor = peak / (0.95*(2^(oranConfig.IQWidth-1)-1));
oranConfig.scaledGrid = round(grid/scaleFactor);

% Apply compression to the scaled resource grid if selected
if ~strcmp(oranConfig.method,'No compression')
    [oranConfig.cGrid,oranConfig.cParam] = nrORANBlockCompress(oranConfig.scaledGrid, ...
        oranConfig.method,oranConfig.cIQWidth,oranConfig.IQWidth);
end
```

Extract CU-Plane Message Configuration

Use the `getCUMessagesConfig` local function to get the configuration per CU-plane message:

- Ethernet header
- eCPRI header
- eCPRI payload (O-RAN configuration)

```
% Get the number of symbols per slot and the number of slots per subframe
waveConfig.symbolsPerSlot = gridSet.Info.SymbolsPerSlot;
waveConfig.slotsPerSubframe = gridSet.Info.SlotsPerSubframe;

% Obtain set of CU-plane messages
[cMessagesConfig,uMessagesConfig] = getCUMessagesConfig(waveConfig,oranConfig, ...
    eCPRIConfig,ethernetConfig);
```

Generate CU-Plane Messages

The `generateCUMessages` local function generates the CU-plane messages according to their configuration in `cMessagesConfig` and `uMessagesConfig`, respectively, and writes the messages to a PCAP file. The generated C-plane messages are Section Type 1, see TS O-RAN.WG4.CUS section 5.4.

```
% Generate CU-plane messages and write the packets to PCAP file
generateCUMessages(cMessagesConfig,uMessagesConfig,mPlaneConfig,pcapFileName);
```

Visualize Generated CU-Plane Messages

You can open the PCAP file containing the generated CU-plane messages in a packet analyzer. The packets decoded by Wireshark match the configuration selected. These figures show the analysis of two captured CU-plane messages in Wireshark.

First C-Plane Message

```
> Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
> Ethernet II, Src: 56:3b:be:a9:92:4c (56:3b:be:a9:92:4c), Dst: da:14:de:b0:55:63 (da:14:de:b0:55:63)
> 802.1Q Virtual LAN, PRI: 7, DEI: 0, ID: 1
> evolved Common Public Radio Interface
v O-RAN Fronthaul CUS-C, Type: 1 (Most channels), Id: 0 (all PRBs)
  > ecpriRtcid (DU_Port_ID: 0, A_Cell_ID: 0, CC_ID: 0, RU_Port_ID: 0)
  > ecpriSeqid, SeqId: 0, SubSeqId: 0, E: 1
  v C-Plane Section Type 1, Downlink, Frame: 0, Subframe: 0, Slot: 0, StartSymbol: 0
      1... .... = Data Direction: Downlink (1)
      .001 .... = Payload Version: 1
      .... 0000 = Filter Index: standard channel filter (0)
      Frame ID: 0
      0000 .... = Subframe ID: 0
      .... 0000 00.. .... = Slot ID: 0
      ..00 0000 = Start Symbol ID: 0
      [RefA: 0-0-0]
      Number of Sections: 1
      Section Type: Most DL/UL radio channels (1)
      0000 .... = User Data IQ width: I and Q are each 16 bits wide (0) (16 bits)
      .... 0000 = User Data Compression Method: No compression (0)
      Reserved: 0
  v Section, Id: 0 (all PRBs), Symbols: 14, BeamId: 0)
      0000 0000 0000 .... = Section ID: 0
      .... 0... = RB Indicator: Every RB used (0)
      .... .0.. = Symbol Number Increment Command: Use the current symbol number (0)
      .... ..00 0000 0000 = Starting PRB of Control Plane Section: 0
      Number of Contiguous PRBs per Control Section: 0
      1111 1111 1111 .... = RE Mask: 0xfff
      .... 1110 = Number of Symbols: 14
      0... .... = Extension Flag: False
      .000 0000 0000 0000 = Beam ID: 0
```

First U-Plane Message

```
> Frame 2: 7680 bytes on wire (61440 bits), 7680 bytes captured (61440 bits)
> Ethernet II, Src: 56:3b:be:a9:92:4c (56:3b:be:a9:92:4c), Dst: da:14:de:b0:55:63 (da:14:de:b0:55:63)
> 802.1Q Virtual LAN, PRI: 7, DEI: 0, ID: 1
> evolved Common Public Radio Interface
v O-RAN Fronthaul CUS-U, Id: 0 (all PRBs)
  > ecpriPcid (DU_Port_ID: 0, A_Cell_ID: 0, CC_ID: 0, RU_Port_ID: 0)
  > ecpriSeqid, SeqId: 0, SubSeqId: 0, E: 1
  v Timing header Downlink, Frame: 0, Subframe: 0, Slot: 0, StartSymbol: 0
      1... .... = Data Direction: Downlink (1)
      .001 .... = Payload Version: 1
      .... 0000 = Filter Index: standard channel filter (0)
      Frame ID: 0
      0000 .... = Subframe ID: 0
      .... 0000 00.. .... = Slot ID: 0
      ..00 0000 = Start Symbol ID: 0
      [RefA: 0-0-0]
  v Section, Id: 0 (all PRBs)
      0000 0000 0000 .... = Section ID: 0
      .... 0... = RB Indicator: Every RB used (0)
      .... .0.. = Symbol Number Increment Command: Use the current symbol number (0)
      .... ..00 0000 0000 = Starting PRB of User Plane Section: 0
      Number of PRBs per User Plane Section: 0
      .... 0001 = User Data Compression Method: Block floating point compression (1)
      1001 .... = User Data IQ width: Bit width of I and Q (9)
      Reserved: 0
    v PRB 0
        0000 .... = Reserved: 0
        .... 0111 = Exponent: 7
        IQ User Data: 80c06030180c0603017fc0602ff7fc06030180c05ff017fbfe0301
    v PRB 1
        0000 .... = Reserved: 0
        .... 0111 = Exponent: 7
        IQ User Data: 80c05feff80c05ff017fc0602ff80c05ff017fbfe02ff7fbfe02ff
    > PRB 2
    > PRB 3
    > PRB 4
    > PRB 5
    > PRB 6
    > PRB 7
```

References

1. 3GPP TS 38.141-1. "NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing." *3rd Generation Partnership Project; Technical Specification Group Radio Access Network*.
2. TS O-RAN.WG4.CUS. "O-RAN Fronthaul Working Group - Control, User and Synchronization Plane Specification".
3. TS O-RAN.WG4.CONF. "O-RAN Fronthaul Working Group - Conformance Test Specification".
4. Wireshark: https://www.wireshark.org/. Accessed 13 July 2022.

Local Functions

```
function [cMessagesConfig,uMessagesConfig] = getCUMessagesConfig(waveConfig,oranConfig,eCPRIConfi
% Obtain CU-plane messages configuration. This function considers Section Type 1 C-plane messages
```

```
% Get the following information per PRB:
% - PRBNumber: PRB number
% - cParam: computational parameter specific to the compression method
% - cIQWidth: compressed IQ samples bit-width (same as IQWidth if uncompressed)
% - IQData: IQ samples

K = size(oranConfig.scaledGrid,1);
L = size(oranConfig.scaledGrid,2);
if ~strcmp(oranConfig.method,'No compression')
    grid = oranConfig.cGrid;
    cParam = oranConfig.cParam;
    cIQWidth = oranConfig.cIQWidth;
else
    grid = oranConfig.scaledGrid;
    cParam = NaN(K/12,L);
    cIQWidth = oranConfig.IQWidth;
end
PRBsInfo = struct;
if length(cIQWidth) == 1
    cIQWidth = repmat(cIQWidth,[K/12 L]);
end
for l = 1:L
    PRBNumber = 0;
    for k = 1:K/12
        PRBsInfo(k,l).PRBNumber = PRBNumber;
        PRBsInfo(k,l).CompParam = [cParam(k,l) cIQWidth(k,l)];
        PRBsInfo(k,l).IQData = grid((k-1)*12+1:k*12,l);
        PRBNumber = PRBNumber+1;
    end
end

% Get time configuration of first U-plane message
uTimeInfo.DataDirection = 1;
uTimeInfo.PayloadVersion = 1;
uTimeInfo.FilterIndex = 0;
uTimeInfo.FrameID = 0;
uTimeInfo.SubframeID = 0;
uTimeInfo.SlotID = 0;
uTimeInfo.SymbolID = 0;

% Indicate compression method according to TS O-RAN.WG4.CUS
switch oranConfig.method
    case 'BFP'
        udCompMeth = 1;
    case 'blockScaling'
        udCompMeth = 2;
    case 'muLaw'
        udCompMeth = 3;
    otherwise % No compression
        udCompMeth = 0;
end

% Get section configuration of first U-plane message
uSectionsInfo.SectionID = 0;
uSectionsInfo.RB = 0;
uSectionsInfo.SymInc = 0;
uSectionsInfo.StartPRBu = 0;
```

```
uSectionsInfo.NumPRBu = 0; % All PRBs in the specified SCS and carrier bandwidth
uSectionsInfo.UdCompHdr = [cIQWidth(1),udCompMeth];
uSectionsInfo.Reserved = 0;
uSectionsInfo.PRBs = PRBsInfo(:,1);

% Set O-RAN configuration of first U-plane message
uOran.TimeInfo = uTimeInfo;
uOran.Sections = uSectionsInfo;

% Generate eCPRI header fields
ecpri.ProtocolRevision = 1;
ecpri.Reserved = 0;
ecpri.Concatenation = 0;
ecpri.MessageType = 'IQData'; % Message type for U-plane message
ecpri.eAxCID = [eCPRIConfig.DUPortID,eCPRIConfig.BandSectorID,eCPRIConfig.CCID, ...
    eCPRIConfig.RUPortID];
sequenceID = 0;
EBit = 1;
subSequenceID = 0;
ecpri.SEQ_ID = [sequenceID,EBit,subSequenceID];

% Generate Ethernet header fields
eth.DestinationAddress = ethernetConfig.destAddress;
eth.SourceAddress = ethernetConfig.sourceAddress;
etherType = 0xAEFE; % eCPRI uses Ethernet with the Ethertype 0xAEFE
vlanTag = [ethernetConfig.TPID,ethernetConfig.priority,ethernetConfig.DEI, ...
    ethernetConfig.VID];
eth.Type = [vlanTag etherType];

% Combine all configurations (Ethernet header, eCPRI header, and O-RAN
% information) of first U-plane message
uMessagesConfig = struct;
uMessagesConfig(1).eth = eth;
uMessagesConfig(1).ecpri = ecpri;
uMessagesConfig(1).oran = uOran;

% Get configuration of remaining U-plane messages
uPackets = size(PRBsInfo,2); % Number of U-plane messages per OFDM symbol
symbolsPerSlot = waveConfig.symbolsPerSlot;
slotsPerSubframe = waveConfig.slotsPerSubframe;
for i = 1:uPackets-1
    uOran.TimeInfo.SubframeID = floor(i/(symbolsPerSlot*slotsPerSubframe));
    uOran.TimeInfo.SlotID = mod(floor(i/symbolsPerSlot),slotsPerSubframe);
    uOran.TimeInfo.SymbolID = mod(i,symbolsPerSlot);
    uOran.Sections.PRBs = PRBsInfo(:,i+1);
    ecpri.SEQ_ID(1,1) = ecpri.SEQ_ID(1,1)+1;
    ecpri.SEQ_ID(1,1) = mod(ecpri.SEQ_ID(1,1),256);
    uMessagesConfig(i+1).eth = eth;
    uMessagesConfig(i+1).ecpri = ecpri;
    uMessagesConfig(i+1).oran = uOran;
end

% Get time configuration of first C-plane message
cTimeInfo.DataDirection = 1;
cTimeInfo.PayloadVersion = 1;
cTimeInfo.FilterIndex = 0;
cTimeInfo.FrameID = 0;
cTimeInfo.SubframeID = 0;
```

```
        cTimeInfo.SlotID = 0;
        cTimeInfo.SymbolID = 0; % Start symbol ID

        % Get section type and compression configurations of first C-plane message
        typeCompInfo.NumberOfSections = 1;
        typeCompInfo.SectionType = 1;
        typeCompInfo.UdCompHdr = 0; % This parameter will be ignored in DL
        typeCompInfo.Reserved = 0;

        % Generate section configuration of first C-plane message
        cSectionsInfo.SectionID = 0;
        cSectionsInfo.RB = 0;
        cSectionsInfo.SymInc = 0;
        cSectionsInfo.StartPRBc = 0;
        cSectionsInfo.NumPRBc = 0; % All PRBs in the specified SCS and carrier bandwidth
        cSectionsInfo.REMAsk = ones(1,12);
        cSectionsInfo.NumSymbol = 14;
        cSectionsInfo.Ef = 0; % No section extensions
        cSectionsInfo.BeamID = 0; % No beamforming

        % Set O-RAN information of first C-plane message
        cOran.TimeInfo = cTimeInfo;
        cOran.SectionInfo = typeCompInfo;
        cOran.Sections = cSectionsInfo;

        % Combine all configurations (Ethernet header, eCPRI header, and O-RAN
        % information) of first C-plane message
        cMessagesConfig = struct;
        cMessagesConfig(1).eth = eth;
        ecpri.MessageType = 'RealTimeControlData'; % Message type for C-plane message
        ecpri.SEQ_ID(1,1) = 0;
        cMessagesConfig(1).ecpri = ecpri;
        cMessagesConfig(1).oran = cOran;

        % Get configuration of remaining C-plane messages
        NSubframes = 10;
        cPackets = slotsPerSubframe*NSubframes;
        for i = 1:cPackets-1
            cOran.TimeInfo.SubframeID = floor(i/slotsPerSubframe);
            cOran.TimeInfo.SlotID = mod(i,slotsPerSubframe);
            ecpri.SEQ_ID(1,1) = ecpri.SEQ_ID(1,1)+1;
            ecpri.SEQ_ID(1,1) = mod(ecpri.SEQ_ID(1,1),256);
            cMessagesConfig(i+1).eth = eth;
            cMessagesConfig(i+1).ecpri = ecpri;
            cMessagesConfig(i+1).oran = cOran;
        end
    end

    function generateCUMessages(cMessagesConfig,uMessagesConfig,mPlaneConfig,pcapFileName)
    % Generate CU-plane messages and write the packets to PCAP file

        % Create a PCAP file writer object and write a global header to the PCAP file
        pcapObj = pcapWriter(FileName = pcapFileName);
        linkType = 1;  % Link type for Ethernet protocol
        writeGlobalHeader(pcapObj, linkType);
        timeStamp = 0; % Packet arrival time in microseconds
        timeDelay = 40; % Time delay between packets in microseconds
```

```
    % Encode Ethernet header of CU-plane messages
    ethHeader = hORANProtocolEncoder.ethernetHeader(cMessagesConfig(1).eth);

    % Calculate number of U-plane packets per slot
    NuPacketsPerSlot = size(uMessagesConfig,2)/size(cMessagesConfig,2);

    % Generate CU-plane packets and write the packets to the PCAP file
    for i = 1:size(cMessagesConfig,2)
        % Encode eCPRI payload of C-plane message
        cECPRIPayload = hORANProtocolEncoder.eCPRIPayloadControl(cMessagesConfig(i).oran);

        % Encode eCPRI header of C-plane message
        cECPRIHeader = hORANProtocolEncoder.eCPRIHeader(cMessagesConfig(i).ecpri, ...
            length(cECPRIPayload),mPlaneConfig.eAxCIDBits);

        % Generate C-plane packet
        cPacket = [ethHeader; cECPRIHeader; cECPRIPayload];

        % Write C-plane packet to PCAP file
        write(pcapObj, cPacket, timeStamp);
        timeStamp = timeStamp+timeDelay*7;

        for j = 1:NuPacketsPerSlot
            % Encode eCPRI payload of U-plane message
            uECPRIPayload = hORANProtocolEncoder.eCPRIPayloadUser(uMessagesConfig(j+NuPacketsPer
                mPlaneConfig.compMode,mPlaneConfig.byteOrder);

            % Encode eCPRI header of U-plane message
            uECPRIHeader = hORANProtocolEncoder.eCPRIHeader(uMessagesConfig(j+NuPacketsPerSlot*(
                length(uECPRIPayload),mPlaneConfig.eAxCIDBits);

            % Generate U-plane packet
            uPacket = [ethHeader; uECPRIHeader; uECPRIPayload];

            % Write U-plane packet to PCAP file
            write(pcapObj, uPacket, timeStamp);
            timeStamp = timeStamp+timeDelay;
        end
    end
end
```

See Also

Functions

nrORANBlockCompress | nrORANBlockDecompress

8

Code Generation and Deployment

What is C Code Generation from MATLAB?

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

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

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

Using MATLAB Coder

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

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

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

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

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See “Generating Standalone C/C++ Executables from MATLAB Code” (MATLAB Coder) for more information.

C/C++ Compiler Setup

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

After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

Functions and System Objects That Support Code Generation

For an alphabetized list of features supporting C/C++ code generation, see 5G Toolbox – Functions Filtered by C/C++ Code Generation.

See Also

Functions
`codegen` | `mex`

More About

- “Code Generation Workflow” (MATLAB Coder)
- Generate C Code from MATLAB Code Video