Wireless project

github link :

https://github.com/fgnbrua/Wirelessfinal/tree/main

```
%load the data
load roomPathData.mat
```

Choose a RX which has LOS path. In this case, I choose RX 1000.

```
%0=Outage = No paths to this link, 1=LOS path, 2 = NLOS paths only.
pathData.linkState()
```

```
ans = 5071×1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
     ⋮
```

Create antenna arrays

```
fc=pathData.fc;
elem = design(patchMicrostrip, fc);
nantgNB = [4,4];
nantUE = [2,2];
lambda = physconst('Lightspeed') / fc;
dsep = 0.5*lambda;
arrgNB = phased.URA(nantgNB,dsep,'ArrayNormal','x');
arrUE  = phased.URA(nantUE,dsep,'ArrayNormal','x');
```

```
arrPlatformgNB = ArrayPlatform('elem', elem, 'arr', arrgNB, 'fc', fc);
arrPlatformgNB.computeNormMatrix();
arrPlatformUE =  ArrayPlatform('elem', elem, 'arr', arrUE, 'fc', fc);
arrPlatformUE.computeNormMatrix();
```

Orient the array

```
azUE = 105;
elUE = -10;
arrPlatformUE.alignAxes(azUE, elUE);
```

```

```

```
aoaAz=pathData.aoaAz(1000,: );
aoaEl=pathData.aoaEl(1000,: );
aodAz=pathData.aodAz(1000,: );
aodEl=pathData.aodEl(1000,: );
[svTx, elemGainTx] = arrPlatformgNB(aodAz', aodEl');
[svRx, elemGainRx] = arrPlatformUE(aoaAz', aoaEl');
pathgain=pathData.gain(1000,:);
gainElem = pathgain' + elemGainTx + elemGainRx;
gain=sum(db2mag(gainElem));
display(gain);%print the total element gain
```

```
gain = 2.6452e-04
```

Calculate the overall element gain for each array orientation.

```
azUE1 = -180:10:180;
elUE1 = -90:10:90;
gain=zeros(length(azUE1),length(elUE1));
for i =1:length(azUE1)
    for j=1:length(elUE1)
        arrPlatformUE.alignAxes(azUE1(i), elUE1(j));
        [svTx, elemGainTx] = arrPlatformgNB(aodAz', aodEl');
        [svRx, elemGainRx] = arrPlatformUE(aoaAz', aoaEl');
        gainElem = pathgain' + elemGainTx + elemGainRx;
        gain(i,j)=sum(db2mag(gainElem));
    end
end
```

In order the get the max throughput, I find the angle when the gain reaches maximum

```
[maxgain i]=max(gain(:));
[x,y]=find(gain==maxgain);
azUE2=azUE1(x);
elUE2=elUE1(y);

arrPlatformUE.alignAxes(azUE2, elUE2);
[svTx, elemGainTx] = arrPlatformgNB(aodAz', aodEl');
[svRx, elemGainRx] = arrPlatformUE(aoaAz', aoaEl');
```

```
gainElem = pathgain' + elemGainTx + elemGainRx;
gain2=sum(db2mag(gainElem));
fprintf('The azimuth angle is %d degree and the elevation angle is %d degree',azUE2,elU
```

```
The azimuth angle is 50 degree and the elevation angle is 90 degree
```

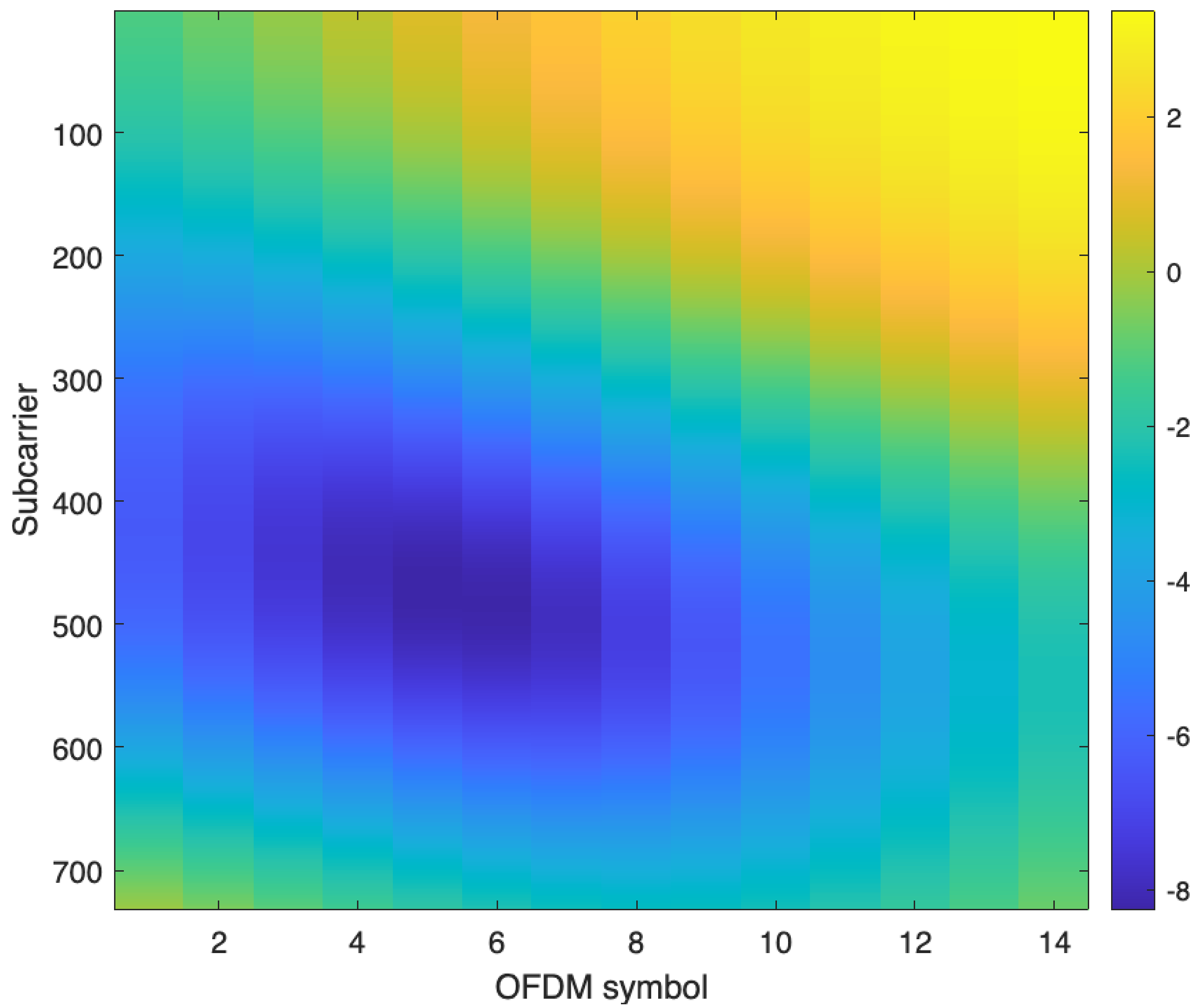
```
dly=pathData.dly(1000,:);
SubcarrierSpacing = 120;    % SCS in kHZ
NRB = 61;  % number of resource blocks
nscPerRB = 12;  % number of sub-carriers per RB
carrierConfig = nrCarrierConfig(...
    'NSizeGrid', NRB, 'SubcarrierSpacing', SubcarrierSpacing);
waveformConfig = nrOFDMInfo(carrierConfig);
```

```
Enoise = -50; %In order to get higher SNR, assume the noise is small
fdchan = FDMIMOChan(carrierConfig, 'txArrPlatform', arrPlatformgNB, 'rxArrPlatform', a
    'aoaAz', aoaAz', 'aodAz', aodAz', 'aoaEl', aoaEl', 'aodEl', aodEl',  ...
    'gain', pathgain, 'dly', dly, 'fc', fc, 'Enoise', Enoise);
```

```
frameNum = 0;
slotNum = 0;
[chanGrid, noiseVar] = fdchan.step(frameNum, slotNum);
```

OFDM frequency domain channel

```
figure();
set(gcf,'Position', [0,0,500,400]);
chanGainSing = squeeze( abs(chanGrid(3,4,:,:)).^2 );
ChanSing = 10*log10(chanGainSing/noiseVar );
imagesc(ChanSing);
colorbar();
xlabel('OFDM symbol');
ylabel('Subcarrier');
```



Analysis: this OFDM channel is different from the one in Lab8. The main reason is that the path loss for RX is too huge so the SNR is small.

I printed the max snr and min snr in dB, which are pretty low compared to snr from Lab8.

```
maxsnr=max(10*log10(abs(chanGrid(:)).^2/noiseVar))
```

```
maxsnr = 7.0869
```

```
minsnr=min(10*log10(abs(chanGrid(:)).^2/noiseVar))
```

```
minsnr = -59.2261
```

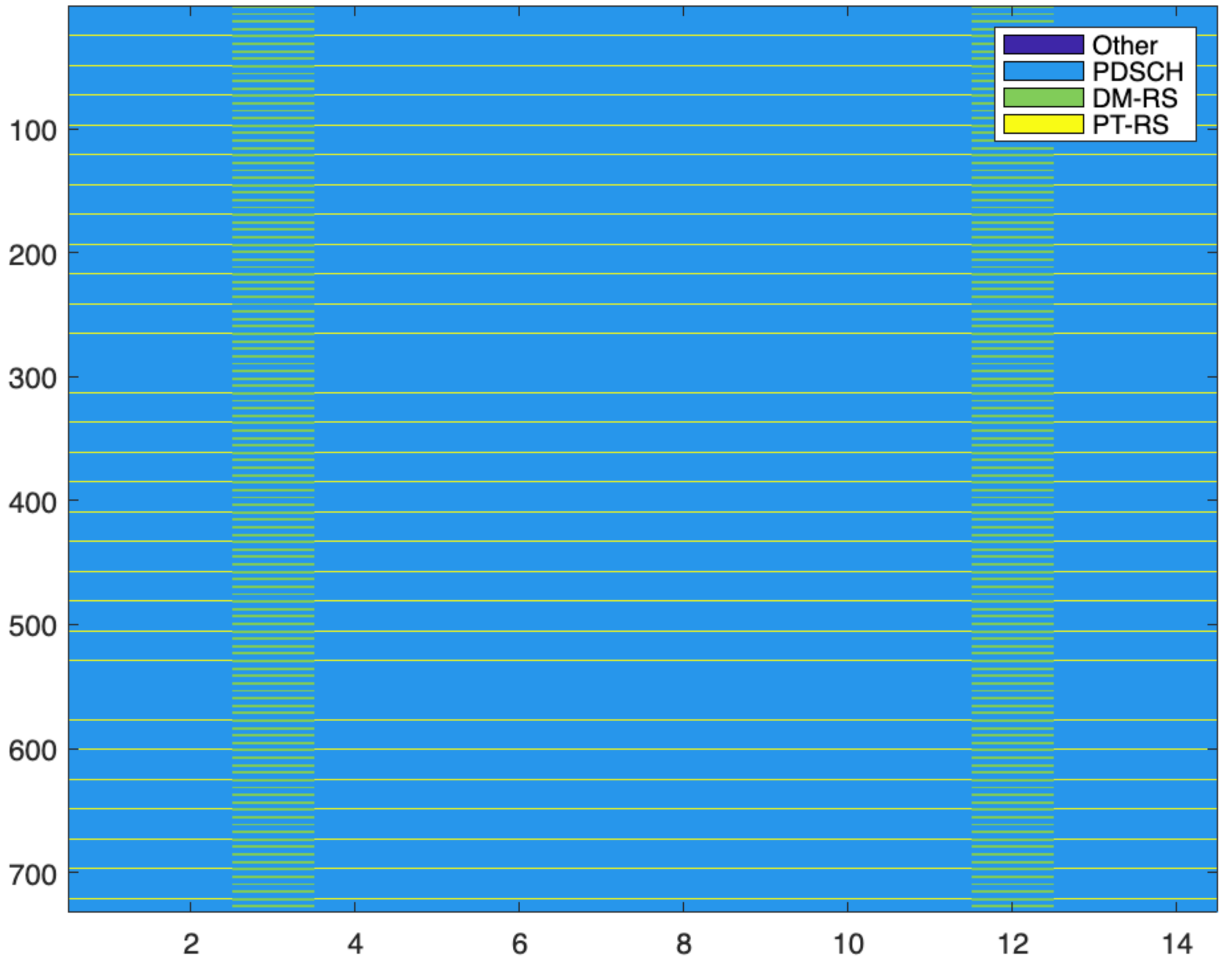
DL PDSCH from Lab7

```
dmrsConfig = nrPDSCHDMRSConfig(...
    'NumCDMGroupsWithoutData', 1, ...  % No unused DM-RS
    'DMRSAdditionalPosition', 1, ...   % Number additional DM-RS in time
    'DMRSConfigurationType', 2);       % 1=6 DM-RS per sym,  2=4 per sym
```

```
pdschConfig = nrPDSCHConfig();
pdschConfig.NSizeBWP = [];   % Empty implies that the value is equal to NSizeGrid
pdschConfig.NStartBWP = [];  % Empty implies that the value is equal to NStartGrid
pdschConfig.PRBSet = (0:NRB-1);   % Allocate the complete carrier
pdschConfig.SymbolAllocation = [0 14]; % Symbol allocation [S L]
pdschConfig.MappingType = 'A'; % PDSCH mapping type ('A' or 'B')
pdschConfig.EnablePTRS = true;
pdschConfig.PTRS =  nrPDSCHPTRSConfig();
pdschConfig.DMRS = dmrsConfig;
```

```
tx = NRgNBTxFD(carrierConfig, pdschConfig);
txgrid=tx.step();

rxgrid=zeros(size(chanGrid));

for i =1:length(rxgrid(:,1,1,1))
    for j=1:length(rxgrid(1,:,1,1))
        rxgrid(i,j,:,:)=squeeze(chanGrid(i,j,:,:)).*txgrid;
    end
end
```

```
figure();
plotChan(tx.txGridChan, tx.chanNames);
```



In order to calcluate the DL PDSCH throughput I used the code from https://www.mathworks.com/help/5g/ug/nr-pdsch-throughput.html

By giving different SNR from frequency domain channel, I can measure the throughput

```
simParameters = struct();       % Clear simParameters variable to contain all key simu
simParameters.NFrames =1;      % Number of 10 ms frames
snrin=linspace(minsnr,maxsnr,8);
simParameters.SNRIn = linspace(minsnr,maxsnr,8); % SNR range (dB)
```

```
simParameters.PerfectChannelEstimator = true;
simParameters.DisplaySimulationInformation = true;
simParameters.DisplayDiagnostics = false;
```

```
% 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 blo
simParameters.Carrier.SubcarrierSpacing = 15;    % 15, 30, 60, 120 (kHz)
simParameters.Carrier.CyclicPrefix = 'Normal';   % 'Normal' or 'Extended' (Extended CP
simParameters.Carrier.NCellID = 1;               % Cell identity

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

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

% 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 tra
    simParameters.PDSCH.Modulation = {'16QAM','16QAM'};             % 'QPSK', '16QAM',
    simParameters.PDSCHExtension.TargetCodeRate = [490 490]/1024;   % Code rate used to
else
    simParameters.PDSCH.Modulation = '16QAM';                       % 'QPSK', '16QAM',
    simParameters.PDSCHExtension.TargetCodeRate = 490/1024;         % Code rate used to
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
simParameters.PDSCH.DMRS.DMRSTypeAPosition = 2;      % Mapping type A only. First DM-RS
simParameters.PDSCH.DMRS.DMRSLength = 1;             % Number of front-loaded DM-RS sym
simParameters.PDSCH.DMRS.DMRSAdditionalPosition = 2; % Additional DM-RS symbol position
simParameters.PDSCH.DMRS.DMRSConfigurationType = 2;  % DM-RS configuration type (1,2)
simParameters.PDSCH.DMRS.NumCDMGroupsWithoutData = 1;% Number of CDM groups without dat
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
```

```
simParameters.PDSCH.PTRS.FrequencyDensity = 2;         % PT-RS frequency density (K_PT-RS
simParameters.PDSCH.PTRS.REOffset = '00';              % PT-RS resource element offset (
simParameters.PDSCH.PTRS.PTRSPortSet = [];             % PT-RS antenna port, subset of DM

% 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 resourc

% 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 "wideban
%
% HARQ process and rate matching/TBS parameters
simParameters.PDSCHExtension.XOverhead = 6*simParameters.PDSCH.EnablePTRS; % Set PDSCH
simParameters.PDSCHExtension.NHARQProcesses = 16;    % Number of parallel HARQ process
simParameters.PDSCHExtension.EnableHARQ = true;      % Enable retransmissions for each

% LDPC decoder parameters
% Available algorithms: 'Belief propagation', 'Layered belief propagation', 'Normalize
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 antenn
if simParameters.PDSCH.NumCodewords > 1           % Multi-codeword transmission
    simParameters.NRxAnts = 8;                    % Number of UE receive antennas (even
else
    simParameters.NRxAnts = 2;                    % Number of UE receive antennas (1 o
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);
```

```
waveformInfo = nrOFDMInfo(simParameters.Carrier);
```

```
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;
```

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

```
% 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.MaximumLDPCIterat

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 n
    % PCT broadcast variables when using parfor
    simLocal = simParameters;
    waveinfoLocal = waveformInfo;

    % Take copies of channel-level parameters to simplify subsequent parameter referen
    carrier = simLocal.Carrier;
    pdsch = simLocal.PDSCH;
    pdschextra = simLocal.PDSCHExtension;
    decodeDLSCHLocal = decodeDLSCH;  % Copy of the decoder handle to help PCT classifi
    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 a
        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),pdschIr

        % HARQ processing
        for cwIdx = 1:pdsch.NumCodewords
            % If new data for current process and codeword then create a new DL-SCH tra
            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 decod
                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.NStart(

        % 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.NStartGr
        pdschGrid(dmrsAntIndices) = dmrsAntSymbols;

        % PDSCH PT-RS precoding and mapping
        ptrsSymbols = nrPDSCHPTRS(carrier,pdsch);
        ptrsIndices = nrPDSCHPTRSIndices(carrier,pdsch);
        [ptrsAntSymbols,ptrsAntIndices] = hPRGPrecode(size(pdschGrid),carrier.NStartGr
        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 char
                    ' This will result in a decoding failure. This may be caused by low
                    ' or not enough DM-RS symbols to synchronize successfully.'],offset
            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,off

            % 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.PRGBund

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

            % Apply precoding to channel estimate
            pdschHest = hPRGPrecode(size(estChannelGrid),carrier.NStartGrid,pdschHest,p
        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,dmrsInd

            % 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,estChannelGri

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

            % Get precoding matrix for next slot
            newWtx = getPrecodingMatrix(carrier,pdsch,estChannelGridPorts,pdschextra.P
        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] = nrExtractResource

            if (simLocal.PerfectChannelEstimator)
                % Apply precoding to channel estimate
                ptrsHest = hPRGPrecode(size(estChannelGrid),carrier.NStartGrid,ptrsHes
            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.PTRSSymbolSe
            tempGrid(:,symLoc,:) = tempGrid(:,symLoc,:).*exp(-1i*cpe(symLoc));
```

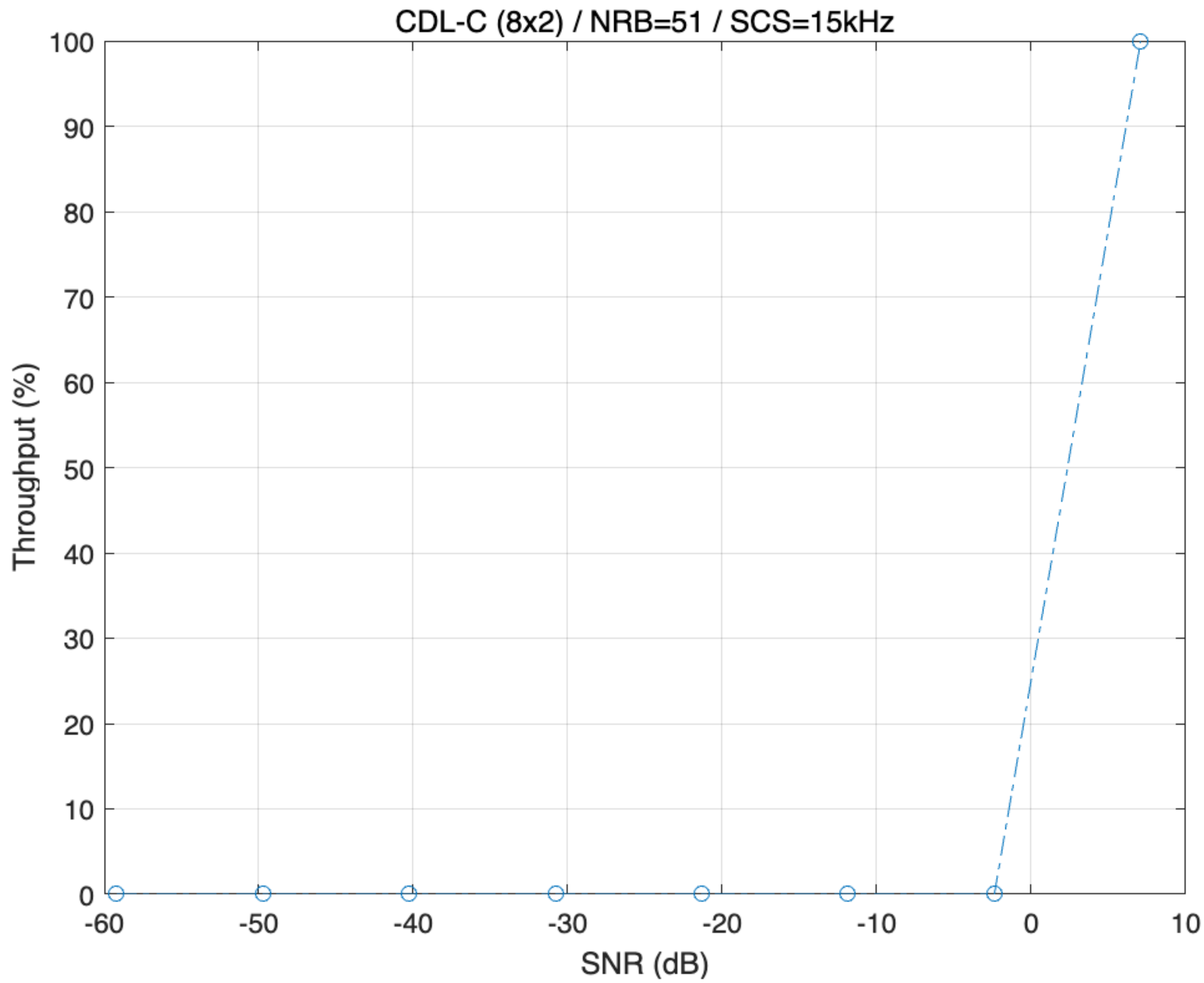
```
            % Extract PDSCH symbols
            pdschEq = tempGrid(pdschIndices);
        end

        % 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,p
        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 b
            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

        % 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*simThr
    fprintf('Throughput(%%) for %d frame(s) = %.4f\n',simLocal.NFrames,simThroughput(s

end
```

```
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -59.2261dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
```

```
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -49.7528dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -40.2795dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -30.8062dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -21.3329dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -11.8597dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
```

```
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at -2.38637dB SNR for 1 10ms frame
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission failed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission failed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission failed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission failed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission failed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission failed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission failed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission failed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission failed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission failed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 0.0000
Throughput(%) for 1 frame(s) = 0.0000
Simulating transmission scheme 1 (8x2) and SCS=15kHz with CDL-C channel at 7.08692dB SNR for 1 10ms frame(
(10.00%) NSlot=0, HARQ Proc 0: CW0: Initial transmission passed (RV=0,CR=0.474736).
(20.00%) NSlot=1, HARQ Proc 1: CW0: Initial transmission passed (RV=0,CR=0.474736).
(30.00%) NSlot=2, HARQ Proc 2: CW0: Initial transmission passed (RV=0,CR=0.474736).
(40.00%) NSlot=3, HARQ Proc 3: CW0: Initial transmission passed (RV=0,CR=0.474736).
(50.00%) NSlot=4, HARQ Proc 4: CW0: Initial transmission passed (RV=0,CR=0.474736).
(60.00%) NSlot=5, HARQ Proc 5: CW0: Initial transmission passed (RV=0,CR=0.474736).
(70.00%) NSlot=6, HARQ Proc 6: CW0: Initial transmission passed (RV=0,CR=0.474736).
(80.00%) NSlot=7, HARQ Proc 7: CW0: Initial transmission passed (RV=0,CR=0.474736).
(90.00%) NSlot=8, HARQ Proc 8: CW0: Initial transmission passed (RV=0,CR=0.474736).
(100.00%) NSlot=9, HARQ Proc 9: CW0: Initial transmission passed (RV=0,CR=0.474736).
Throughput(Mbps) for 1 frame(s) = 30.2160
Throughput(%) for 1 frame(s) = 100.0000
```

```
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;
```

```

```

```
rx = NRUERxFD(carrierConfig, pdschConfig);
```

Frequency domain equalization and LLR calculation

```
% Get indices on where the PDSCH is allocated
pdschInd = nrPDSCHIndices(carrierConfig, pdschConfig);
pdschSymEq=zeros(length(rxgrid(:,1,1,1)),length(rxgrid(1,:,1,1)) ,length(pdschInd));
llr=zeros(length(rxgrid(:,1,1,1)),length(rxgrid(1,:,1,1)) ,length(pdschInd)*2);
for i =1:length(rxgrid(:,1,1,1))
    for j=1:length(rxgrid(1,:,1,1))
```

```
        rxGrid=squeeze(rxgrid(i,j,:,:));
        pdschSym = rxGrid(pdschInd);
        chanGrid1=squeeze(chanGrid(i,j,:,:));
        pdschChan = chanGrid1(pdschInd);
        pdschSymEq(i,j,:) = conj(pdschChan).*pdschSym./(abs(pdschChan).^2 + noiseVar);
        llr(i,j,:) = qamdemod(squeeze(pdschSymEq(i,j,:)),4,'OutputType','approxllr', .
        'UnitAveragePower',true,'NoiseVariance', noiseVar);
    end
end


%pdschSymEq(i,j,:,:) is the equalization between TX(i) and RX(j)

%llr(i,j,:,:) is the llr between TX(i) and RX(j)
```

Choose a different RX and find its throughput

```
aoaAz=pathData.aoaAz(50,: );
aoaEl=pathData.aoaEl(50,: );
aodAz=pathData.aodAz(50,: );
aodEl=pathData.aodEl(50,: );

azUE = 105;
elUE = -10;
arrPlatformUE.alignAxes(azUE, elUE);

[svTx, elemGainTx] = arrPlatformgNB(aodAz', aodEl');
[svRx, elemGainRx] = arrPlatformUE(aoaAz', aoaEl');
pathgain=pathData.gain(1000,:);
gainElem = pathgain' + elemGainTx + elemGainRx;
gain=mag2db(sum(db2mag(gainElem)));
snr=gain-Enoise;
[a index]=min(abs(snr-snrin));
throughput1=(1e-6*simThroughput/(simLocal.NFrames*10e-3));
throughput=throughput1(index);
fprintf('\nThroughput(Mbps) for RX(500) = %.4f\n',throughput);
```

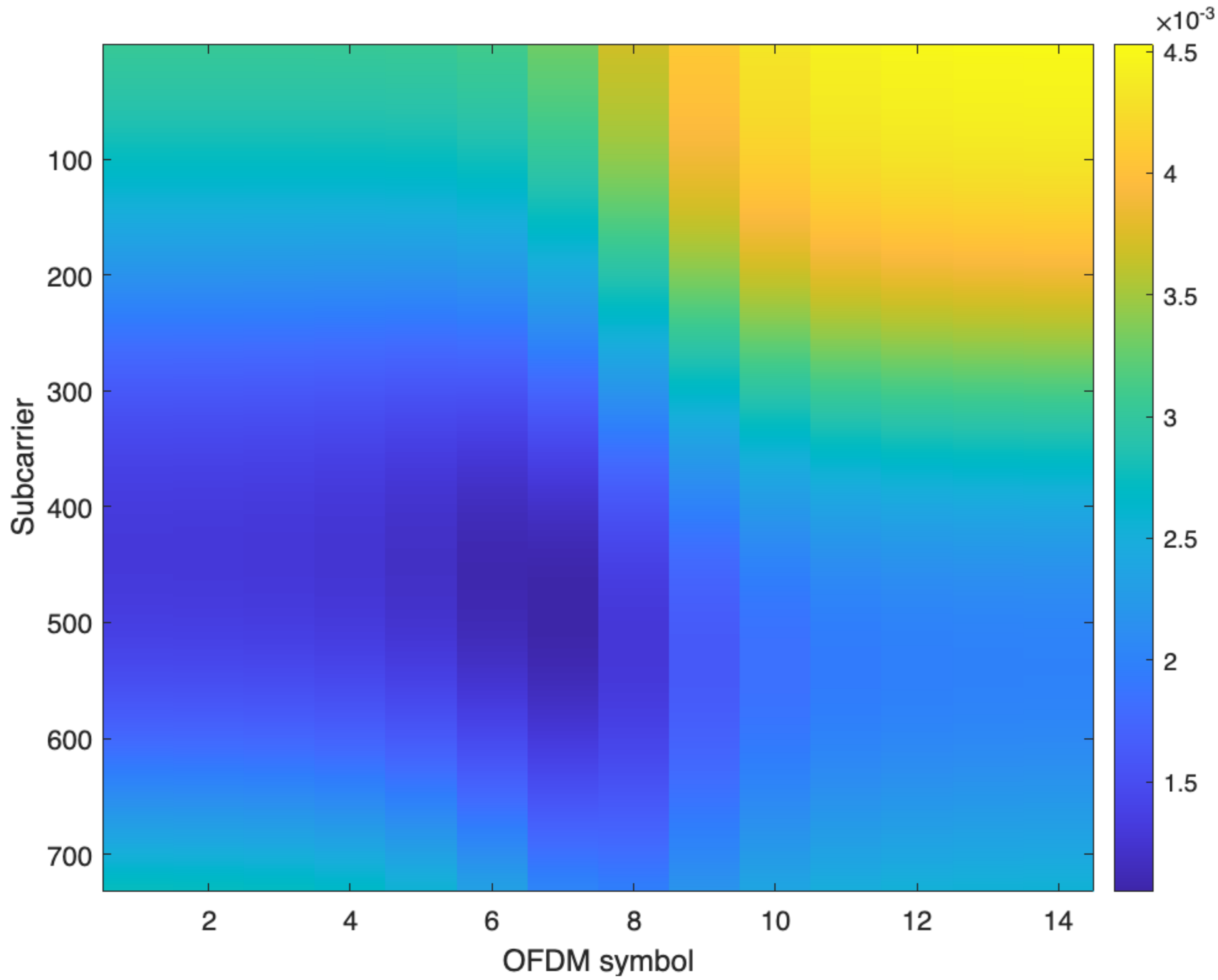
```
Throughput(Mbps) for RX(500) = 0.0000
```

Channel estimator

```
chanEstGrid=zeros(size(chanGrid));

for i =1:length(rxgrid(:,1,1,1))
    for j=1:length(rxgrid(1,:,1,1))
        rx.chanEst(squeeze(rxgrid(i,j,:,:)));
        chanEstGrid(i,j,:,:)=rx.chanEstGrid;
    end
```

```
end
```

```
%plot channel estimation at (3,4)
figure();
imagesc(squeeze(abs(chanEstGrid(3,4,:,:))));
colorbar();
xlabel('OFDM symbol');
ylabel('Subcarrier');
```



```
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,nrxan
    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 Nu
            ' 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)
    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

    % 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,samp

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
    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
```