Simulating the 5G NR PDSCH over a Fading Channel

In this lab, we will build a simple transmitter and receiver for the 5G NR downlink data channel, called the Physical Downlink Shared Channel or PDSCH. We will perform the downlink simulation over a fading channel. Writing this simulation from scratch would be months of work given all the complexity of the NR standard. However, in this lab, we will be able to use MATLAB's 5G Toolbox which has incredible functions to perform all the basic operations.

In going through the lab, you will learn to:

- Describe and visualize the OFDM frame structure of the NR standard including slots, subframes, and frames in time and sub-carriers and resource blocks in frequency.
- Configure an OFDM channel with the 5G Toolbox.
- Create a mutli-path channel in frequency domain for the OFDM link.
- Measure the variation in SNR over one slot and over multiple slots to determine the effects of fast and slow fading.
- Configure a PDSCH transmission including its resource block allocation and MCS.
- Use 5G Toolbox commands to compute the transport block size, and perform the transmit steps including scrambling, LDPC encoding, symbol mapping and resource mapping to the OFDM grid.
- Use 5G Toolbox commands to perform the RX steps including resource extraction, MMSE equalization, LDPC decoding, and

Some of the code in this lab is based on MATLAB's PDSCH throuhput demo. You will need to get the R2020B version of MATLAB's 5G Toolbox for the demo. The Toolbox is rapidly evolving, and I have used some functions that are not available in R2020A. Also, along with this file, you will see three other files:

- `FDChan.m`: A class for frequency-domain channel simulation
- `NRgNBTxFD.m`: A class for NR gNB TX frequency-domain simulation
- `NRUERxFD.m`: A class for NR UE RX frequency-domain simulation

These files have some TODO sections to be completed.

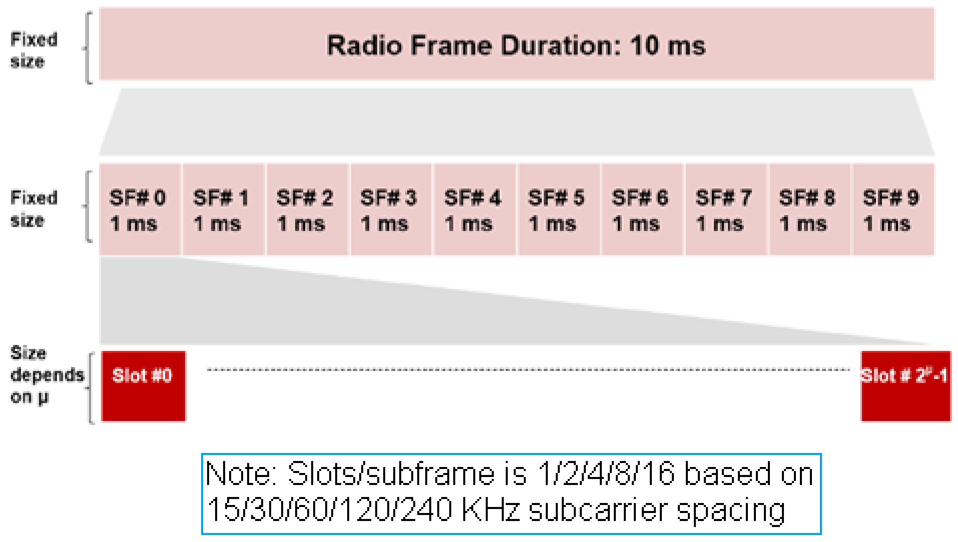
Submission: Complete this MATLAB live file and the other files. Run the code. Print all the files to PDF. Merge to a single PDF and submit the PDF for grading.

5G NR OFDM Channel Background

One of the main goals of the 5G NR standard was to build a highly configurable channel structure that can operate in a range of frequencies and band allocations. The figure below are taken from this excellent website: https://www.rfwireless-world.com/5G/5G-NR-Numerology-or-Terminology.html

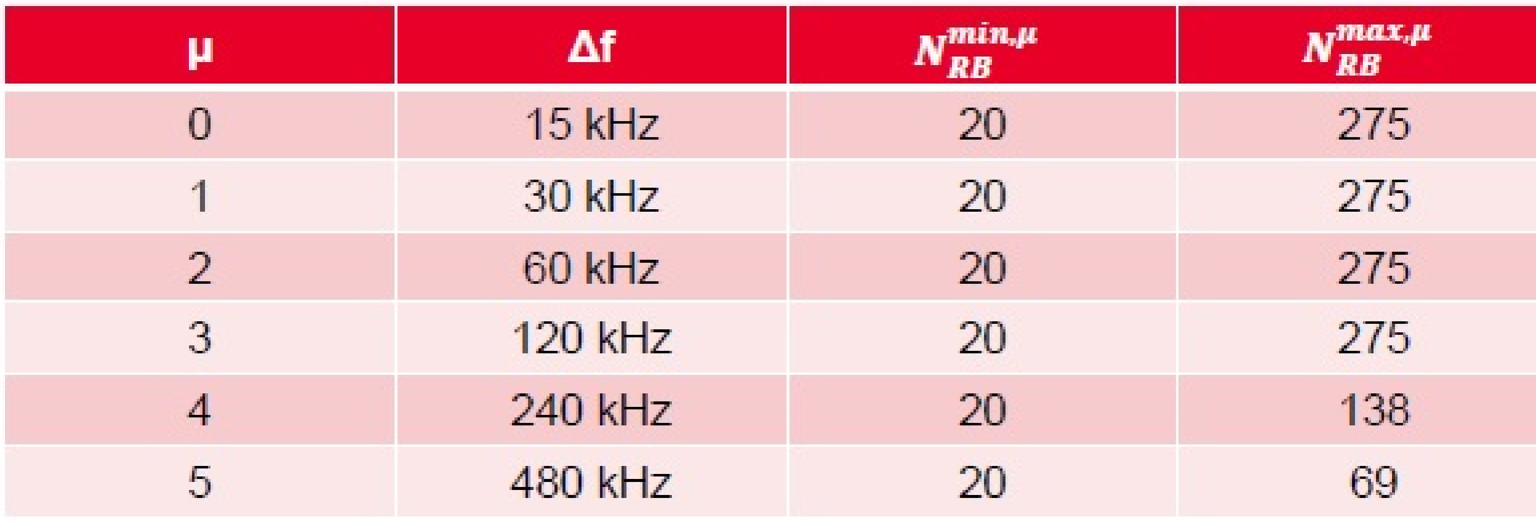
To enable configuration, the 5G NR standard uses OFDM with a configurable time and frequency structure. For the time structure, time is divided into **radio frames** of 10 ms, which are then divided into 10 **sub-frames** of 1ms each. The subframes are then divided into **slots** where the number of slots per subframe are configurable.

Specifically, the NR channel has a basic **configuration parameter** `mu=0,1,...,4,` and the number of slots per subframe is `2^mu.`



Note: Slots/subframe is 1/2/4/8/16 based on 15/30/60/120/240 KHz subcarrier spacing

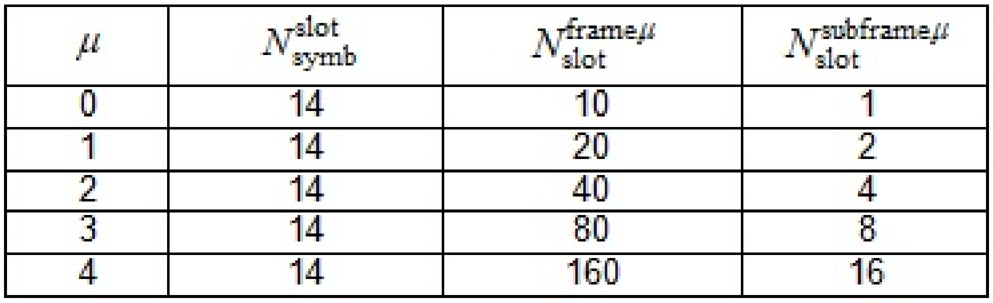
In frequency, the channel uses OFDM with a sub-carrier spacing `Delta f,` that is tied to the configuration `mu.` In each slot, the bandwidth is then divided into what are called **resource blocks (RBs)** which are 12 subcarriers for one slot. The total bandwidth is then controlled by configuring the total number of resource blocks. The sub-carrier spacing and the minimum and maximum number of RBs for each configuration `mu` is shown below.



μ	Δf	$N_{RB}^{min,\mu}$	$N_{RB}^{max,\mu}$
0	15 kHz	20	275
1	30 kHz	20	275
2	60 kHz	20	275
3	120 kHz	20	275
4	240 kHz	20	138
5	480 kHz	20	69

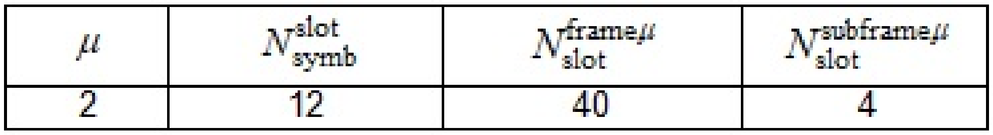
Finally, each slot is divided into OFDM symbols. The number of symbols per slot is 14 for what is called the normal cyclic prefix and 12 for extended cyclic prefix. Most systems use normal CP. The extended CP is used for environments with very large delay spreads.

Table 4.3.2-1: Number of OFDM symbols per slot, slots per frame, and slots per subframe for normal cyclic prefix.



μ	N_{symb}^{slot}	$N_{slot}^{frame\mu}$	$N_{slot}^{subframe\mu}$
0	14	10	1
1	14	20	2
2	14	40	4
3	14	80	8
4	14	160	16

Table 4.3.2-2: Number of OFDM symbols per slot, slots per frame, and slots per subframe for extended cyclic prefix.



μ	N_{symb}^{slot}	$N_{slot}^{frame\mu}$	$N_{slot}^{subframe\mu}$
2	12	40	4

Configuring the OFDM Channel

In this lab, we will configure the parameters for an OFDM channel at a carrier frequency of 28 GHz. We will use the following parameters, which are widely-used in these frequencies.

```
fc = 28e9;                  % Carrier frequency
SubcarrierSpacing = 120;  % SCS in kHZ
NRB = 66;  % number of resource blocks
```

Using the SCS and number of resource blocks, compute and print the bandwidth occupied by this waveform.

```
nscPerRB = 12;  % number of sub-carriers per RB

% TODO:
bwMHz = SubcarrierSpacing * NRB * nscPerRB / 1e3;
fprintf(1, 'The bandwidth in [MHz] = %.2f\n', bwMHz);
```

```
The bandwidth in [MHz] = 95.04
```

In the 5G Toolbox, you can configure the channel with the `nrCarrierConfig` command which creates an object with all the relevant parameters. Use the `nrCarrierConfig` command to create a `carrierConfig` structure with the `SubcarrierSpacing` from above and the property NSizeGrid set to `NRB`.

```
% TODO
carrierConfig = nrCarrierConfig("NSizeGrid",NRB,"SubcarrierSpacing",SubcarrierSpacing);
```

Given the `carrierConfig` we can the configuration of the waveform with following command:

```
waveformConfig = nrOFDMInfo(carrierConfig);
```

Use the `waveformConfig` structure to print:

- The sample rate in MHz
- Number of OFDM symbols per subframe

```
% TODO
srMHz = waveformConfig.SampleRate / 1e6;
symbsSubframe = waveformConfig.SlotsPerSubframe * waveformConfig.SymbolsPerSlot;
fprintf(1, 'The sample rate in [MHz] = %.2f\n', srMHz);
```

```
The sample rate in [MHz] = 122.88
```

```
fprintf(1, 'The number of symbols per subframe = %.f\n', symbsSubframe);
```

```
The number of symbols per subframe = 112
```

Configuring a Frequency-Domain OFDM Channel

In this lab, we will simulate the channel in frequency domain. For OFDM systems, frequency-domain simulation enables us to avoids the need to simulate the TX and RX filtering, synchronization and channel estimation. We can add these components of the simulation later. We will use the following simple channel with two paths.

```
% Parmaeters for each path
gain = [0,-3]';      % path gain in dB
dly = [0, 200e-9]';  % path delays in seconds
aoaAz = [0, 180]';  % angles of arrival
aoaEl = [0, 20]';

% Mobile velocity vector in m/s
rxVel = [30,0,0]';

% Parameters for computing the SNR
Etx = 1;        % Average transmitted symbol energy
EsN0Avg = 20;   % Average SNR
```

The lab material contains a file, `FDChan.m` which has a skeleton for modeling channels in frequency domain. You can create this channel object using this file with the following command:

```
fdchan = FDChan(waveformConfig, 'gain', gain, 'dly', dly, 'aoaAz', aoaAz, 'aoaEl', aoaEl, ...
                   'rxVel', rxVel, 'Etx', Etx, 'EsN0Avg', EsN0Avg, 'fc', fc);
```

The file `FDChan.m` has various sections labeled `TODO` that should be completed. First, complete the section in the constructor that sets complex gains for each path

The vector `gainComplex` should have one complex value for each path, with the gain set by gaindB and a random complex phase from `[0,2*pi]` .

Next, set the Doppler for each path based on the mobile velocity and angles of the path. by completing the following part:

Edit the code, recreate the `fdchan` object and print the Doppler shifts:

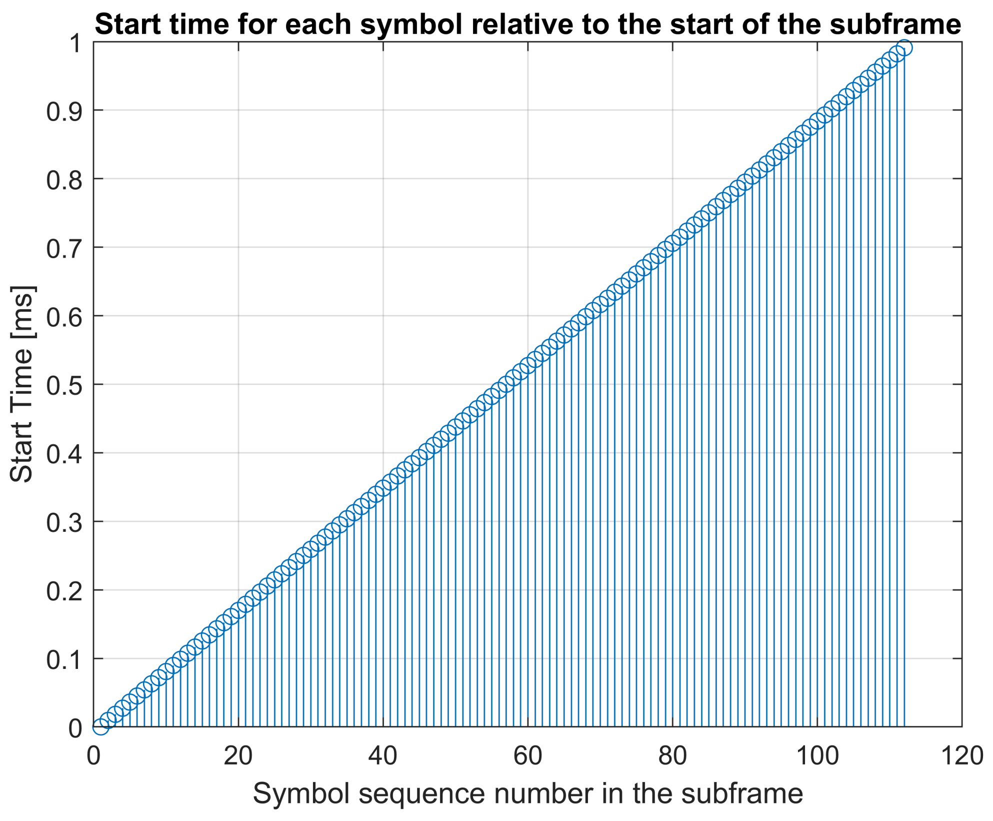
```
% TODO:  Print fdchan.fd
fprintf(1, 'The Doppler shifts [Hz] = %.2f\n', fdchan.fd);
```

```
The Doppler shifts [Hz] = 2801.94
The Doppler shifts [Hz] = -2632.96
```

For the fd simulator, we will also need to compute the time of the OFDM symbols relative to the beginning of the sub-frame. Complete the following section in the `FDChan` class to create a vector obj.symStart where `obj.symStart(i) is the time, in seconds, of the start of the `i`-th OFDM symbol relative to the beginning of the subframe. You can use the waveformConfig.SymbolLengths and waveformConfig.SampleRate.

Plot `fdchan.symStart.`   If you did this correctly, the values should go from 0 to 1ms.

```
% TODO
symbStarts = fdchan.symStart;
figure;
stem(symbStarts*1e3);
xlabel("Symbol sequence number in the subframe"); ylabel("Start Time [ms]");
title("Start time for each symbol relative to the start of the subframe");
grid on;
```



Simulating the Frequency-Domain Channel

The method, `FDChan.stepImpl()` takes as an input an array, `txGrid(n,j)`, representing the TX OFDM data on sub-carriers n and OFDM symbol numbers j. The output grid is given by,

```
        rxGrid(n,j) = chan(n,j)*txGrid(n,j) + noise(n,j)
```

where the channel is:

```
        chan(n,j) = \sum_k gainComplex(k)*exp(2*pi*1i*(time(j)*fd(k) + n*scs*dly(k))
```

and `time(j)` is the start time of the `j`-th symbol and `scs` is the sub-carrier spacing. The noise variance is set to obtain a cerain average SNR given by,

```
        EsN0Avg = Etx*E| chan(n,t) |^2 / noiseVar
```

The average channel gain is

```
        E| chan(n,t) |^2 = \sum_k |gainComplex(k)|^2
```

Use these formulae to complete the method `stepImpl`.

```
% TODO:  Complete the method FDChan.stepImpl()
```

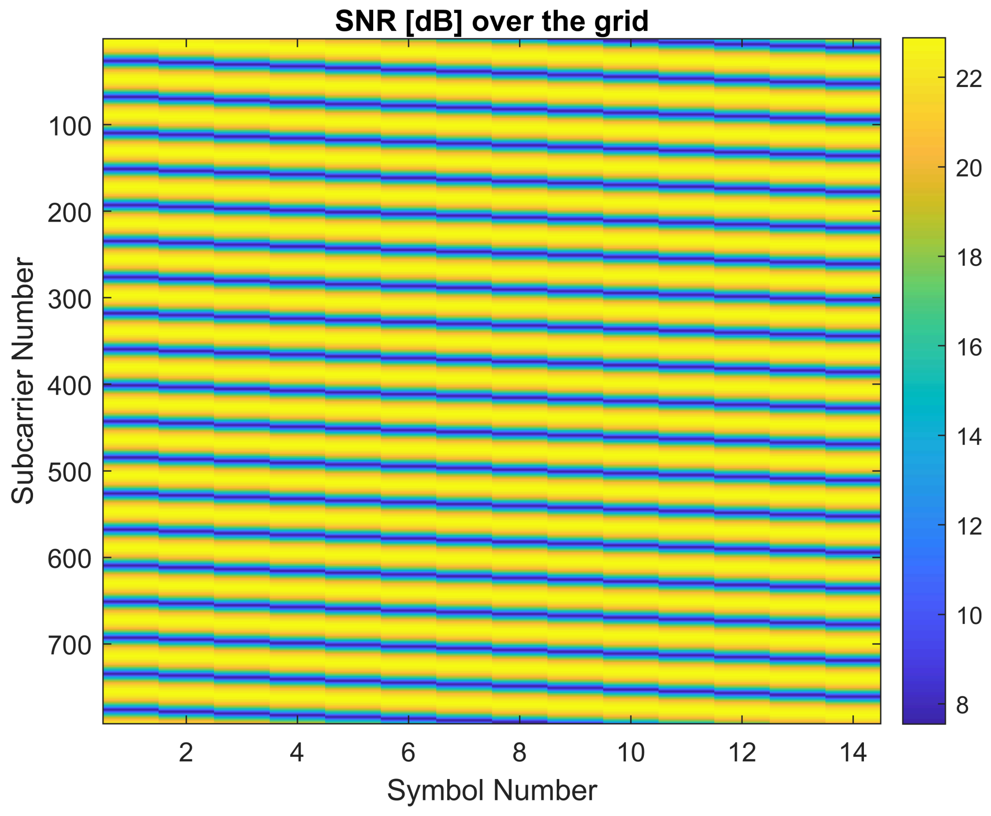
To test the channel, we will create a TX grid corresponding to one slot. You can do this with the following command.

```
NumLayers = 1;
```

```
txGrid = nrResourceGrid(carrierConfig, NumLayers);
txGrid(:) = 1;
```

Use the `fdchan.step()` method to find the channel and noise variance on this grid at subframe number, `sfNum=0` and slot number, `slotNum=0`. Compute the SNR in dB over the grid and plot the SNR using the `imagesc` command. You should see there is a high level of variation of the SNR across the slot indicating significant **fast fading**.

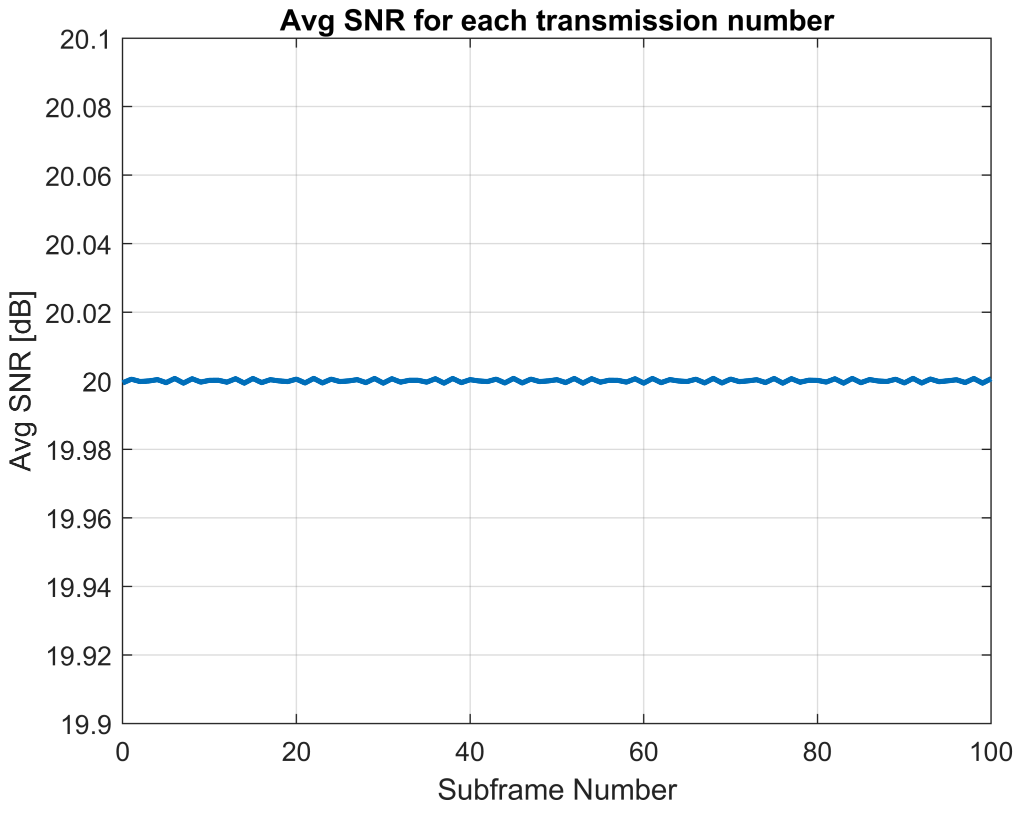
```
% TODO
[rxGrid_test, chanGrid_test, noiseVar_test] = fdchan.step(txGrid, 0, 0);
nsc = NRB*12;
figure;
imagesc(1:14, 1:nsc, pow2db(abs(chanGrid_test).^2/noiseVar_test));
ylabel("Subcarrier Number"); xlabel("Symbol Number");
title("SNR [dB] over the grid");
colorbar();
```



We will now look at the average SNR over different slots. Suppose we transmit on `slotNum=0` for `sfNum=i` for `i=0,...,ntx` for `ntx=100`. Hence, we are transmitting once per subframe. For each transmission, compute the channel and the average SNR. Plot the average SNR as a function of the transmission number. You should see that it is very constant. Thus, although individual resource elements are fading, the average SNR experienced over different blocks indicating that there is **neglible slow fading**.

```
% TODO
ntx = 100;
```

```
slotNum = 0;
avgSNRs = zeros(ntx+1,1);
for j=1:ntx+1
    [~, chanGrid_sf, noiseVar_sf] = fdchan.step(txGrid, j-1, slotNum);
    SNR = mean((abs(chanGrid_sf).^2 / noiseVar_sf),"all");
    avgSNRs(j) = pow2db(SNR);
end

figure;
plot(0:ntx, avgSNRs, "LineWidth",2);
xlabel("Subframe Number"); ylabel("Avg SNR [dB]"); grid on;
title("Avg SNR for each transmission number"); ylim([19.9, 20.1]);
```



Creating a Simple NR PDSCH Transmitter

Downlink data (the link from the base station to the UE) in the NR system is transmitted on what is called the PDSCH channel or Physical Downlink Shared Channel. It is called *shared* since multiple UEs share the channel. Each transmissions is called a transport block. The transport block occupies one slot with a variable number of resource blocks. Multiple UEs can be scheduled in the slot on different resource blocks. When we discuss MIMO technologies, users will also be able to be transmitted on different *streams* or layers, but for now we will just use a single antenna system with one layer.

Each transport block has one of 28 **modulation and coding scheme (MCS)** given by this table.

< 38.214 - Table 5.1.3.1-1: MCS index table 1 for PDSCH >

MCS Index I_{MCS}	Modulation Order Q_m	Target code Rate x [1024] R	Spectral efficiency
0	2	120	0.2344
1	2	157	0.3066
2	2	193	0.3770
3	2	251	0.4902
4	2	308	0.6016
5	2	379	0.7402
6	2	449	0.8770
7	2	526	1.0273
8	2	602	1.1758
9	2	679	1.3262
10	4	340	1.3281
11	4	378	1.4766
12	4	434	1.6953
13	4	490	1.9141
14	4	553	2.1602
15	4	616	2.4063
16	4	658	2.5703
17	6	438	2.5664
18	6	466	2.7305
19	6	517	3.0293
20	6	567	3.3223
21	6	616	3.6094
22	6	666	3.9023
23	6	719	4.2129
24	6	772	4.5234
25	6	822	4.8164
26	6	873	5.1152
27	6	910	5.3320
28	6	948	5.5547
29	2	reserved	
30	4	reserved	
31	6	reserved	

For this simulation we will use MCS=13.

```
mcsInd = 13;

% TODO
Modulation = '16QAM'; % MCS 13 is 4 bits per symbol
targetCodeRate = 490/1024;
```

Once you have selected these parameters, we can confgure the PDSCH transmission with the following command. In this case, I have configured the PDSCH to use all the RBs and all but the first OFDM symbol in a slot. I have also added what is called the phase tracking reference signal (PTRS) for mitigating phase noise. We will discuss this channel more later when we disucss equalaization.

```
pdschConfig = nrPDSCHConfig(...
    'Modulation', Modulation, ...
    'PRBSet', (0:NRB-1), ...
    'SymbolAllocation', [1, waveformConfig.SymbolsPerSlot-1], ...
    'EnablePTRS', 1,...
    'PTRS', nrPDSCHPTRSConfig());
```

The NR has an extremely complex way of transmitting data involving LDPC encoding, scrambling, mapping to QAM symbols and then placing the symbols on the RX grid. These would takes months of time to write from scratch. The 5G toolbox has amazing functions to do all of this for you.

I have created a simple TX class using these functions. You will not need to modify it. But, we will just look at the outputs. You can create an instance of this class as follows.

```
tx = NRgNBTxFD(carrierConfig, pdschConfig, 'targetCodeRate', targetCodeRate);
```

You can then call the `step` method to generate random bits for one slot of data based on the PDSCH configuration. If you look at the code in the step implementation you can see the MATLAB 5G Toolbox commands to encode and modulate the data.

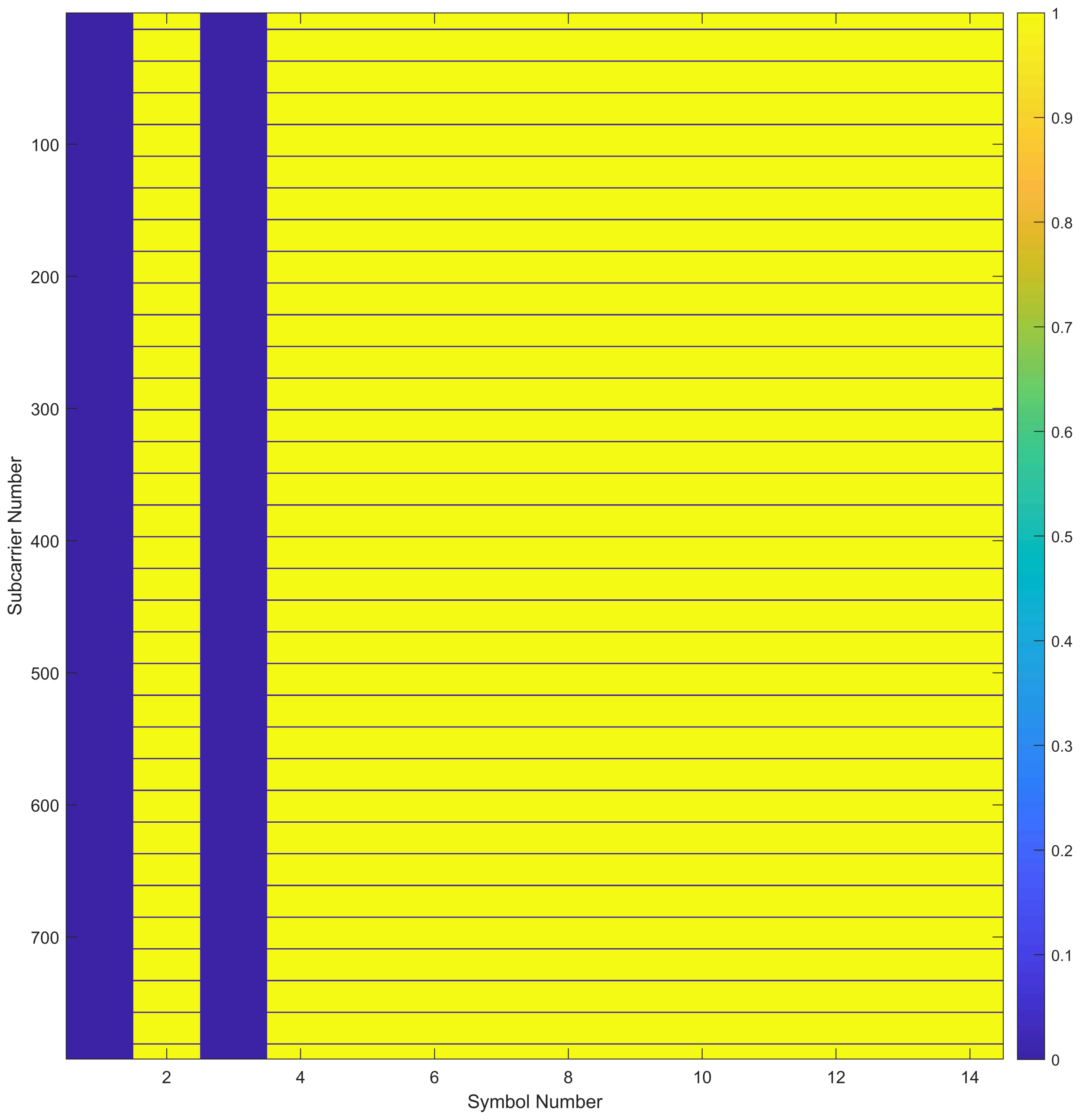
```
txGrid = tx.step();
```

Use the `imagesc` command to plot the set of resource elements in the grid where `txGrid` is non-zero. These are the resource elements (REs) where the PDSCH channel is allocated. You should see that the PDSCH is allocated except at the following locations:

- The first OFDM symbol. This is where the assignment channel, called the PDCCH, will go. Basically this communicates to the UEs the resource allocation, UE identity in that slot. We will discuss
- The third OFDM symbol. which is used as demodulation reference signal (DM-RS) used in channel estimation. We will discuss this when we discuss equalization
- The horizontal strips. These are PT-RS for phase noise mitigation that we will also discuss later.

```
% TODO
figPos = get(0,'defaultfigureposition');
```

```
width = 1e3;
height = 1e3;
figure('Position', [figPos(1), figPos(2), width, height]);
reGrid = zeros(size(txGrid));
reGrid(abs(txGrid) ~= 0) = 1;
imagesc(1:14, 1:nsc, uint8(reGrid));
xlabel("Symbol Number"); ylabel("Subcarrier Number");
colorbar();
```



```
% We can see that the first and third OFDM symbols across all subcarriers
% are occupied (by the PDCCH and DM-RS respectively)
% We can also see the horizontal strips
```

Next, compute and print:

- The number of REs allocated to the PDSCH in the slot
- The fraction of PDSCH REs to the total number of REs.

```
% TODO
reSlot = reGrid(:,4);
nREPdsch = sum(reSlot)*12;
bwFrac = sum(reGrid,"all")/(NRB*12*14);

fprintf(1, 'The number of REs allocated in one slot = %.f\n', nREPdsch);
```

```
The number of REs allocated in one slot = 9108
```

```
fprintf(1, 'The fraction of PDSCH REs to the total number of REs = %.6f\n', bwFrac);
```

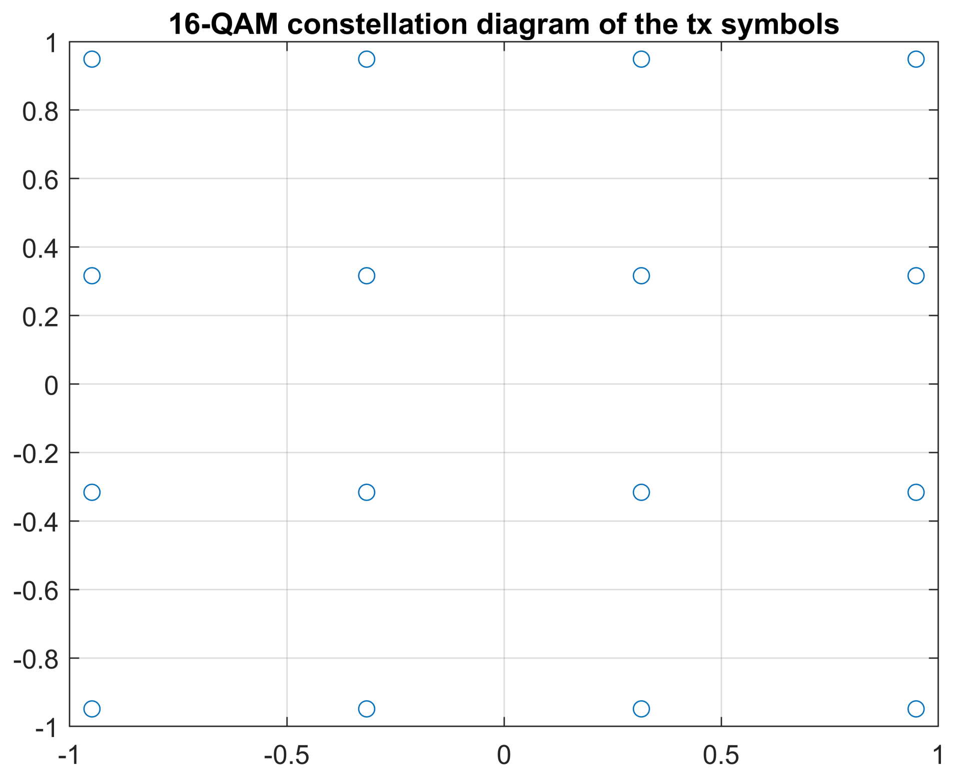
```
The fraction of PDSCH REs to the total number of REs = 0.821429
```

Next, let's look at the transmitted QAM symbols. The NR standard has a complex mapping of the PDSCH to the REs.. We can use the following command to get the indices, `pdschInd`, of the PDSCH locations for a given configuraiton. It also returns a structure, `pdschInfo` with other modualtion data.

```
% Get indices on where the PDSCH is allocated
[pdschInd,pdschInfo] = nrPDSCHIndices(carrierConfig, pdschConfig);
```

Use the PDSCH symbols from `txGrid` from the `pdschInd` and plot the resulting constellation diagram. You should see a 16-QAM constellation with unit average power.

```
% TODO
flat_txGrid = txGrid(:);
pdschSym = flat_txGrid(pdschInd);
plot(real(pdschSym), imag(pdschSym), 'o');
grid on; title("16-QAM constellation diagram of the tx symbols");
```



Finally, let's compute the spectral efficiency. In NR, the transmitted information bits for a transport block can be stored in either one or two codewords. For this configuration, there will only be one codeword (two codewords are used when there are more than four spatial streams). The randomly generated information bits in the transmission are stored in a field, `tx.txBits{1}`. Compute and print:

- The number of information bits that were transmitted in this transport block.
- The spectral efficiency, meaning the number of information bits per PDSCH RE.

The spectral efficiency should be close to the target spectral efficiency in the MCS table.

```
% TODO
txbits = tx.txBits{1};
n_info_bits = length(txbits);
tot_bits = nREPdsch*4;
se = n_info_bits/tot_bits*4;
fprintf(1, 'The number of information bits in the TB = %.f\n', n_info_bits);
```

```
The number of information bits in the TB = 17424
```

```
fprintf(1, 'The spectral efficiency = %.6f\n', se);
```

```
The spectral efficiency = 1.913043
```

Building an NR Receiver

We will next build a simple NR receiver at the UE. I have written most of the code containe in a class `NRUERxFD`. However, there are a few sections labeled `TODO` in the `stepImpl` method. These just call built-in functions from the 5G Toolbox for equalization, LLR computation and decoding. You just need to supply each function with the correct inputs.

Once you have completed the `TODO` parts, an RX object can be created as follows.

```
% rx = NRUERxFD(carrierConfig, pdschConfig, 'targetCodeRate', targetCodeRate);
```

Now test the receiver:

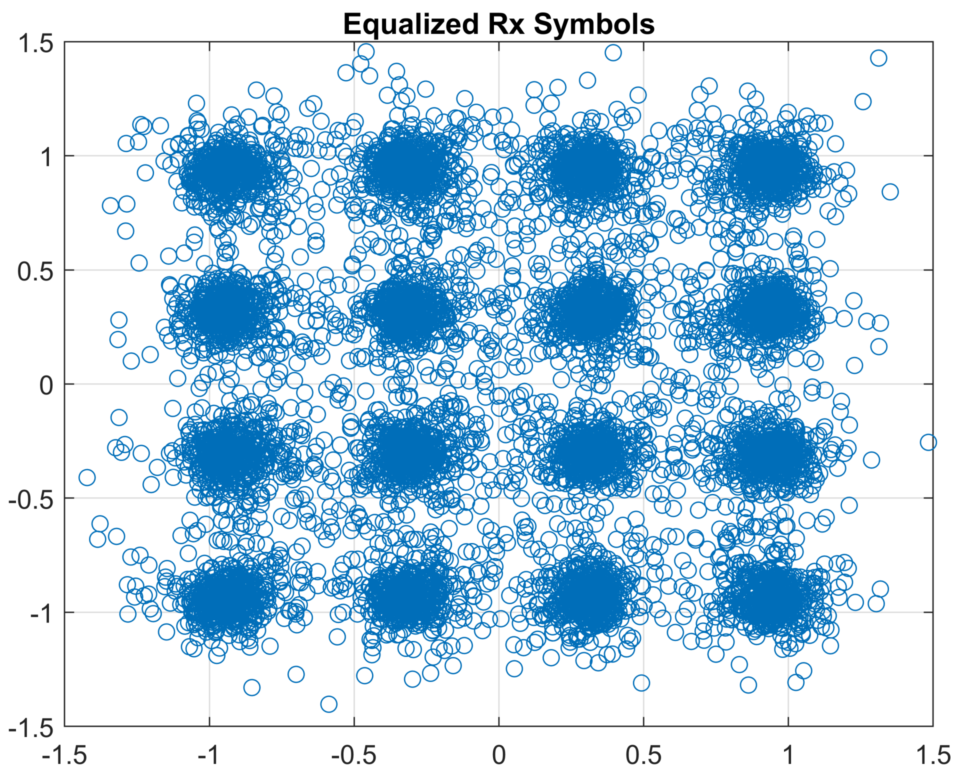
- Create a `NRgNBTxFD` transitter object, `tx` as above.
- Run the TX to create one slot of symbols.
- Create an `FDChan` object as above with the two paths and an average SNR of 20 dB.
- Run the `txGrid` through the chanel at slot = 0 and subframe = 0
- Create a RX object, `rx` of type `NRUERxFD.`
- Run the RX grid, channel grid and noise variance into the RX
- Make sure that `rx.rxBits == tx.txBits{1}`.

```
% TODO:  Create the TX
tx = NRgNBTxFD(carrierConfig, pdschConfig, 'targetCodeRate', targetCodeRate);
txGrid = tx.step();
fdchan = FDChan(waveformConfig, 'gain', gain, 'dly', dly, 'aoaAz', aoaAz, 'aoaEl', aoaEl, ...
                'rxVel', rxVel, 'Etx', Etx, 'EsN0Avg', 20, 'fc', fc);
[rxGrid, chanGrid, noiseVar] = fdchan.step(txGrid, 0, 0);
rx = NRUERxFD(carrierConfig, pdschConfig, 'targetCodeRate', targetCodeRate);
rx.step(rxGrid, chanGrid, noiseVar);
rxBits = rx.rxBits;
txBits = tx.txBits{1};
checkBits = sum(rxBits == txBits);

if checkBits == length(txBits)
    disp("All bits received correctly");
end
```

```
All bits received correctly
```

Also, plot the equalization constelllation stored in the RX.

```
% TODO
figure;
plot(real(rx.pdschEq), imag(rx.pdschEq), 'o');
title("Equalized Rx Symbols"); grid on;
```



Testing the Receiver Performance

The above validated the performance of the RX at high SNR. Now vary the SNR, `EsN0Avg` from 5 to 10 dB in 1 dB steps. In each SNR, simulate the link above with repeated transmissions one subframe apart. At each SNR, repeat until either:

- You get 50 block errors, or
- You have tested at least 500 TX

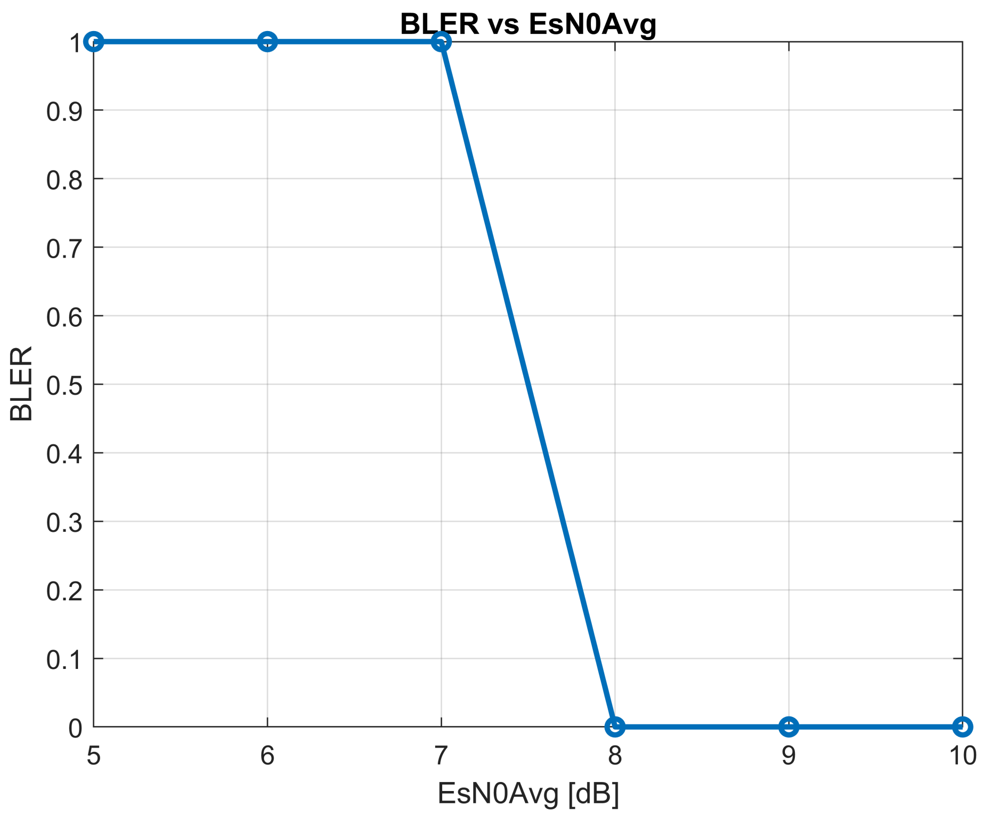
Once you have a SNR with zero block errors, you can stop and not continue to the higher SNRs. Measure the block error rate at each SNR. This may take a few minutes.

Plot the BLER vs. SNR. If you do everything correctly, you should get that the BLER goes to zero Es/N0=8 dB.

```
% TODO:  Compute BLER vs. EsN0Avg
EsN0Avg_values = 5:10;
tx = NRgNBTxFD(carrierConfig, pdschConfig, 'targetCodeRate', targetCodeRate);
rx = NRUERxFD(carrierConfig, pdschConfig, 'targetCodeRate', targetCodeRate);
max_tx = 500;
max_be = 50;
BLER = zeros(length(EsN0Avg_values), 1);

for i=1:length(EsN0Avg_values)
    EsN0Average = EsN0Avg_values(i);
    fdchan = FDChan(waveformConfig, 'gain', gain, 'dly', dly, 'aoaAz', aoaAz, 'aoaEl', aoaEl, .
                'rxVel', rxVel, 'Etx', Etx, 'EsN0Avg', EsN0Average, 'fc', fc);
    subF_num = 0;
    num_be = 0;
    while (subF_num < max_tx) && (num_be < max_be)
        txGrid = tx.step();
        [rxGrid, chanGrid, noiseVar] = fdchan.step(txGrid, subF_num, 0);
        rx.step(rxGrid, chanGrid, noiseVar);
        rxBits = rx.rxBits;
        txBits = tx.txBits{1};
        checkBits = sum(rxBits == txBits);

        if checkBits ~= length(txBits)
            num_be = num_be+1;
        end
        subF_num = subF_num+1;
    end

    BLER(i) = num_be / (subF_num);
    fprintf(1, 'EsN0 = %.6f\n', EsN0Average);
    fprintf(1, 'BLER = %.6f\n', BLER(i));
    if num_be == 0
        disp("Zero block errors in 500 txs: stopping");
        break;
    end
end
```

```
EsN0 = 5.000000
BLER = 1.000000
EsN0 = 6.000000
BLER = 1.000000
EsN0 = 7.000000
BLER = 1.000000
EsN0 = 8.000000
BLER = 0.000000
Zero block errors in 500 txs: stopping
```

```
% TODO:  Plot BLER vs EsN0Avg
figure;
plot(EsN0Avg_values, BLER, 'o-', "LineWidth",2); grid on;
xlabel("EsN0Avg [dB]"); ylabel("BLER");
title("BLER vs EsN0Avg");
```



Estimating the Shannon Ergodic Capacity

We conclude by estimating the theoretical Shannon capacity over a typical slot. The Shannon ergodic spectral efficiency is:

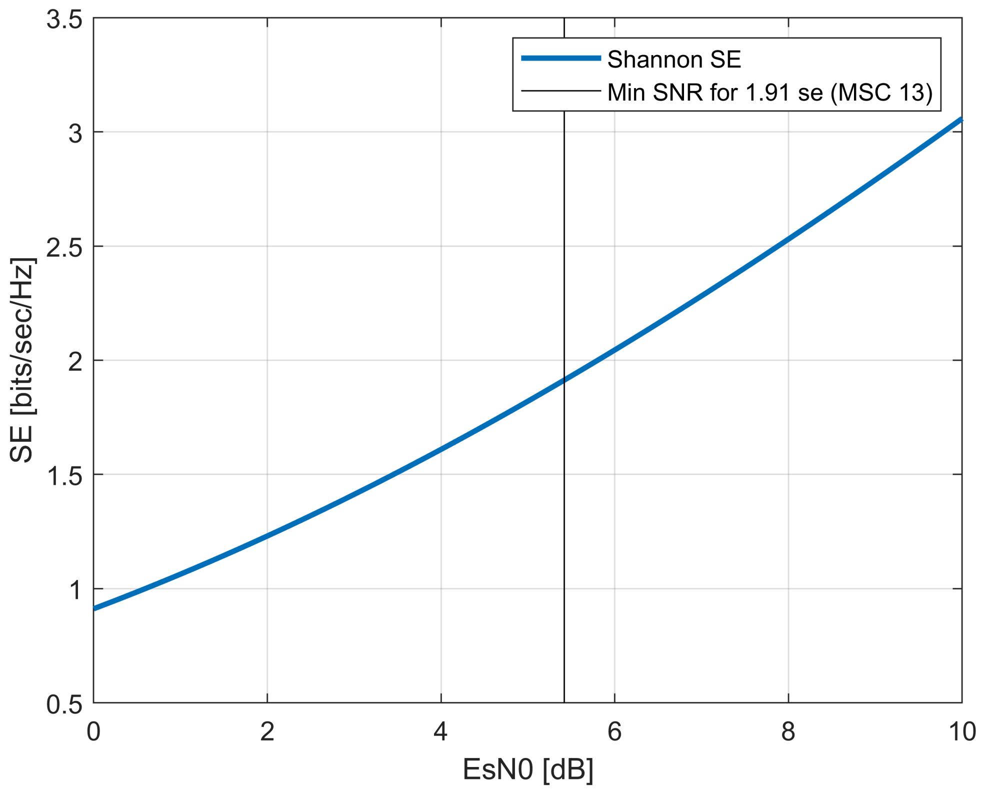
```
seShannon = mean( log2( 1 + c*gamma) )
c = abs(chanGrid).^2 / mean(abs(chanGrid).^2)
```

where `chanGrid` are the complex channel gains on the OFDM grid and gamma is the average Es/N0.

Compute this Shannon spectral efficiency vs. EsN0 for 0 to 10 dB. You should see that theoretically, you can achieve the spectral efficiency in the MCS of 1.9 bps/Hz at about Es/N0 = 5.3 dB. So, the NR code only runs about 2 to 2.5 dB below Shannon capacity.

```
% TODO
npoints = 100;
EsN0 = linspace(0,10,npoints);
c = abs(chanGrid(:)).^2;
c = c / mean(c);
seShannon = zeros(npoints,1);
for i=1:npoints
    seShannon(i) = mean(log2(1 + c*db2pow(EsN0(i))));
end

EsN0_shannon = interp1(seShannon,EsN0,se);
figure;
```

```
plot(EsN0, seShannon, "LineWidth",2); grid on;
hold on;
plot(repelem(EsN0_shannon, npoints), linspace(0.5,3.5,npoints), "k-")
hold off; xlabel("EsN0 [dB]"); ylabel("SE [bits/sec/Hz]");
legend("Shannon SE","Min SNR for 1.91 se (MSC 13)");
```



```matlab
classdef FDChan < matlab.System
    % Frequency-domain multipath channel
    properties
        % Configuration
        carrierConfig;   % Carrier configuration

        % Path parameters
        gain;  % Path gain in dB
        dly;   % Delay of each path in seconds
        aoaAz, aoaEl; % Angle of arrival of each path in degrees
        fd;    % Doppler shift for each path

        rxVel = [30,0,0]';  % Mobile velocity vector in m/s
        fc = 28e9;    % Carrier freq in Hz

        gainComplex;  % Complex gain of each path

        % SNR parameters
        Etx = 1;       % average energy per PDSCH symbol
        EsN0Avg = 20;  % Avg SNR per RX symbol in dB

        % Symbol times
        symStart;  % symStart(i) = start of symbol i relative to subframe

    end
    methods
        function obj = FDChan(carrierConfig, varargin)
            % Constructor

            % Save the carrier configuration
            obj.carrierConfig = carrierConfig;


            % Set parameters from constructor arguments
            if nargin >= 1
                obj.set(varargin{:});
            end

            % TODO:  Create complex path gain for each path
            obj.gainComplex = 10.^(0.05*obj.gain).*exp(1i*(rand(size(obj.gain))*2*pi));

            % TODO:  Compute the Doppler shift for each path
            [x,y,z] = sph2cart(deg2rad(obj.aoaAz'),deg2rad(obj.aoaEl'),ones(size(obj.aoaAz')));
            u = [x; y; z]';
            obj.fd = u*obj.rxVel*(obj.fc/physconst('lightspeed'));

            % Compute unit vector in direction of each path

            % TODO:  Compute the vector of
            % symbol times relative to the start of the subframe
            durations = obj.carrierConfig.SymbolLengths ./ obj.carrierConfig.SampleRate;
            cumulative_sum = cumsum(durations);
            obj.symStart = [0, cumulative_sum(1:end-1)];

        end


    end
    methods (Access = protected)
```

```
        function [rxGrid, chanGrid, noiseVar] = stepImpl(obj, txGrid, sfNum, slotNum)
            % Applies a frequency domain channel and noise
            %
            % Given the TX grid of OFDM REs, txGrid, the function
            % *  Computes the channel grid, chanGrid, given the
            %    subframe number, sfNum, and slotNum.
            % *  Computes the noise variance per symbol, noiseVar,
            %    for a target SNR
            % *  Applies the channel and noise to create the RX grid
            %    of symbols, rxGrid.
            start_idx = 14*slotNum+1;
            % disp(obj.symStart(start_idx:start_idx+14-1));
            t = obj.symStart(start_idx:start_idx+14-1)' + sfNum*1e-3;

            [nsc, nsym] = size(txGrid);
            chanGrid = zeros(nsc, nsym);
            scs = obj.carrierConfig.SlotsPerSubframe*15*1e3;
            f = scs*(0:nsc-1)';

            for k = 1:length(obj.gainComplex)
                phase = obj.fd(k)*t' + obj.dly(k)*f;
                chanGrid = chanGrid + obj.gainComplex(k)*exp(2*pi*1i*phase);
            end

            E_ch_gain = sum(abs(obj.gainComplex).^2);
            noiseVar = obj.Etx*E_ch_gain/db2pow(obj.EsN0Avg);

            %chan_awgn = comm.AWGNChannel('NoiseMethod', 'Variance',...
            %'Variance', noiseVar);

            rxGrid = txGrid.*chanGrid + (randn(size(txGrid))+1i*randn(size(txGrid)))*sqrt(noiseVar/2);
        end

    end
end
```

```
Not enough input arguments.

Error in FDChan (line 31)
            obj.carrierConfig = carrierConfig;
```

```
classdef NRUERxFD < matlab.System
    % 5G NR UR receiver class implemented in frequency domain
    properties
        % Configuration
        carrierConfig;   % Carrier configuration
        pdschConfig;     % Default PDSCH config
        waveformConfig;  % Waveform config

        % OFDM grid
        rxGrid;

        % Transport block data for last transmission
        targetCodeRate = 490/1024;  % Target code rate
        trBlkSizes;                 % Transport block size

        % Received data in last slots
        pdschEq;       % Equalized PDSCH symbols
        rxBits;        % RX bits

        % DLSCH decoder
        decDLSCH;


    end
    methods
        function obj = NRUERxFD(carrierConfig, pdschConfig, ...
                varargin)
            % Constructor

            % Save the carrier and PDSCH configuration
            obj.carrierConfig = carrierConfig;
            obj.pdschConfig = pdschConfig;

            % Create the waveform configuration from the carrier
            % configuration
            obj.waveformConfig = nrOFDMInfo(obj.carrierConfig);

            % Set parameters from constructor arguments
            if nargin >= 1
                obj.set(varargin{:});
            end

            % Create DLSCH decoder
            obj.decDLSCH = nrDLSCHDecoder('MultipleHARQProcesses', false, ...
                'TargetCodeRate', obj.targetCodeRate, ...
                'LDPCDecodingAlgorithm', 'Layered belief propagation');

        end
    end
    methods (Access = protected)


        function stepImpl(obj, rxGrid, chanGrid, noiseVar)
            % Demodulates and decodes one slot of data

            % Get PDSCH received symbols and channel estimates
            % from received grid
            [pdschInd,pdschInfo] = nrPDSCHIndices(obj.carrierConfig, obj.pdschConfig);
            [pdschRx, pdschHest] = nrExtractResources(pdschInd, rxGrid, chanGrid);
            % TODO:  Perform the MMSE equalization using the
            % nrEqualizeMMSE() function.
            % Use the PDSCH Rx symbols, PDSCH channel estimate and noise
            % variance as the input.  Store the equalized symbols in
            % obj.pdschEq and  channel state information in a structure,
            % csi.
            [obj.pdschEq,csi] = nrEqualizeMMSE(pdschRx,pdschHest,noiseVar);

            % TODO:  Get the LLRs with the nrPDSCHDecode() function.
            % Use carrier and PDSCH configuration, the equalized symbols,
            % and the noise variance, noiseVar.
            [dlschLLRs,rxSym] = nrPDSCHDecode(obj.carrierConfig,obj.pdschConfig,obj.pdschEq, noiseVar);

            % Scale LLRs by EbN0.
            % The csi value computed in the nrEqualizeMMSE()
            % function is csi = |pdschHest|^2 + noiseVar.
            % Also, the Eb/N0 = snrEq/Qm where Qm is the number of bits
            % per symbol and snrEq is the SNR after equalization,
            %
            %   snrEq = (|pdschHest|^2 + noiseVar)/noiseVar = csi/noiseVar
            %
            % Hence, Eb/N0 = csi/(noiseVar*Qm).
            % Since the LLRs from the nrPDSCHDecode function are
            % already scaled by 1/noiseVar, we multiply them by  csi/Qm.

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

            % Compute the extra overhead from the PT-RS
            Xoh_PDSCH = 6*obj.pdschConfig.EnablePTRS;

            % Calculate the transport block size based on the PDSCH
            % allocation and target code rate
            obj.trBlkSizes = nrTBS(obj.pdschConfig.Modulation,obj.pdschConfig.NumLayers,...
                numel(obj.pdschConfig.PRBSet),pdschInfo.NREPerPRB,...
                obj.targetCodeRate,Xoh_PDSCH);
            obj.decDLSCH.TransportBlockLength = obj.trBlkSizes;

            % Reset the soft buffer
            harqId = 0;
            obj.decDLSCH.resetSoftBuffer(harqId);

            % TODO:  Decode the bits with the obj.decDLSCH() method.
            % Use the scaled LLRs from above. Use a redundancy version,
            % rv = 0,  since we are not using HARQ in this lab.
            rv = 0;
            obj.rxBits = obj.decDLSCH(dlschLLRs,obj.pdschConfig.Modulation,obj.pdschConfig.NumLayers,rv);

        end

    end
end
```

```
Not enough input arguments.

Error in NRUERxFD (line 31)
            obj.carrierConfig = carrierConfig;
```