Lab: Channel sounder

In this lab, we will demonstrate how to build and simulate a simple channel sounder. Channel sounders are used to measure the channel response between a TX and RX. These are vital to study propagation. They are also an excellent tool for debugging the front-end of a transceiver system.

In doing this lab, you will learn to:

- Generate and receive signals for frequency-domain channel sounding
- Estimate the channel in both frequency and time-domain
- Simulate the channel sounder over a 3GPP multi-path TDL model. Note that in this simulation, we will ignore "fading".
- Incorporate ADC and DAC into the simulation.
- Display the frequency domain and time-domain estimated channel

System parameters

We will use the following parameters. These are the parameters for an 8 channel 5G New Radio front-end.

```
fsamp = 8*120*1.024*1e6;   % Sampling frequency in Hz
Tsamp = 1/fsamp;       % Sampling period
```

Create a TX signal in frequency domain

In a frequency-domain channel sounder, we first create a vector of `nfft` complex symbols in frequency, where `nfft` is a parameter. We will call this group of `nfft` samples, one "frame".

Create one frame of `nfft` random QPSK symbols. You can use the `randi` function to create the bits and then map them to QPSK symbols using the `qammod` function. Store the vector of symbols in `x0_fd`, where we use the subscript `fd` to denote that the symbols are in frequency-domain.

```
nfft = 1024;
nreptx = 16;

% TODO
% x0Fd = ...
```

Now take the IFFT of the `x0Fd` and store that in a vector `x0`. The vector `x0` will represent one frame of samples in time-domain.

```
% TODO
% x0 = ...
```

In frequency-domain sounding, this vector is repeatedly transmitted. For the simulation in this example, we will repeat this `nrep=16` times. Repeat the vector `x0`, `nrep` times and store the result in `xtx`. The resulting length should be `nfft*nreptx`.

```
% TODO
% xtx = ...
```

Note that, in reality, we would transmit a signal at less than the full sampling rate and use filtering to avoid out of band emissions. To make this lab simpler, we have skipped this step.

3GPP TDL model

3GPP is an organization that designs common standards for wireless technologies such as 5G. Among many other documents, 3GPP publishes channel models that can be used by designer for evaluating their products. In this lab, we will use a simple tap delay line (TDL) model from 3GPP TR 38.900. In this model, the channel is described as a number of "taps" or physical paths,

```
        hchan(t) = \sum_k g(k) \delta(t-tau(k)),
```

where g(k) and tau(k) are the gain and delay of path k.   MATLAB can get the paths for these models with the following command:

```
tdl = nrTDLChannel('DelayProfile', 'TDL-A', 'DelaySpread', 1e-8);
chaninfo = info(tdl);
gainPathdB = chaninfo.AveragePathGains';
delayPath = chaninfo.PathDelays';
```

To simulate the channel:

- Convert the `gainPathdB` to a magnitude and add a random phase. Store the complex vector of numbers in a vector `gainPath`.
- Plot the magnitude of `gainPath` (in linear scale) vs. the delay of the paths (in microseconds) using a stem plot

```
% TODO
%   gainPath = ...
%   stem(...);
```

Compute the channel response in frequency domain

To simulate the channel, we first compute the channel response in frequency domain. We will use the following frequency discretization points.

```
% Freq discretization points
npts = 256;
fplot = fsamp*(-npts/2:npts/2-1)'/npts;
```

Compute the frequency response of the channel at the frequencies: Store the values in a vector `Hchan`

```
% TODO
% Hchan = ...
```

Compute the frequency response of the TX and RX filter. We assume that

```
prx(t) = ptx(t) = 1/sqrt(Tsamp)*Rect(t/Tsamp)
```

which corresponds to a zero order hold ADC and DAC.

```
% TODO
% Prx = ...
% Ptx = ...
```

Compute the effective frequency response with filtering by multipling `Prx`, `Ptx` and `Hchan`. Store the result in a vector `G`.

```
% TODO
% G = ...
```

Compute the effective discrete-time channel by taking the IFFT of `G` and scaling by `1/Tsamp`. Then, ue the `fftshift` function to shift the frequency response so that the zero term appears at `npts/2`.

```
% TODO:
% h = ...
```

Create a plot:

- Plot `abs(h)`, the effective discrete time channel response against time. Note that each sample of `h` is at a delay of `Tsamp` and `h(i)` for `i=npts/2` corresponds to `t=0`. Use the `stem` plot.
- On the same plot, plot the TDL gains.
- Label the axes
- Create a legend

```
% TODO
```

You will see that the discrete-time channel taps align roughly with the channel taps in the RF channel. However, they are not exactly equal since the taps in the TDL channel get spread out from the filtering. Also, several TDL taps are unresolvable since they are within one sampling period.

Filter the signal

Filter the signal `xtx` with the discrete-time filter `h`. Put the output in `yrx`.

```
% TODO
% yrx = filter(...)
```

Remove the first npts/2+1 samples as these are the delay in the discrete-time filter.

```
% TODO
% yrx = yrx(...)
```

Performing the FFT at the Receiver

To implement the FFT channel sounder, divide the received samples into "frames" of `nfft` samples. Remove the first frame since it has not passed through the channel. You will also have to remove samples at the end so that you have an integer number of frames. Rearrange the samples in to an array `yrxFrames` which is `nfft x nreprx`, where `nreprx` is the number of RX frames.

```
% TODO
% yrxFrame = ...
```

Take the FFT of each fraems in `yrxFrames`. Then average the result over the frames. Store the result in `yrxFd`.

```
% TODO
% yrxFd = ...
```

Compute the estimated channel frequency response by dividing `yrxFd` and `x0Fd`. Store the result in `hestFd`.

```
% TODO
% hestFd = ...
```

Compute the estimated time-domain channel response by taking the IFFT.

```
% TODO
% hest = ifft(...)
```

Plot the first 100 samples of the estimated channel `hest`. Also, plot the channel taps in the original TDL model. As before, you will see that they somewhat align but there is blurring due to the pulse shaping.

```
% TODO
```

Notes: In the above exercised, we have carefully started the RX frames at the right point so that the first path starts at zero. In general, it will start at arbitrary point. So, channel sounding in this manner does not provide absolute delay, but only relative delays with other paths.