Frequency Estimation and Carrier Frequency Offset

Table of Contents

Complex exponentials are the most fundamental signals for all frequency domain analysis of linear system. In this lab you will learn to:

- Send a complex exponential or continuous-wave (CW) signal through the SDR
- Estimate the complex gain and frequency of a sinusoid via (1) correlation method, (2) FFT method
- Estimate the carrier frequency offset
- Save data for files for offline processing

Submissions: Run the lab with Pluto devices with either one or two hosts. Fill in all sections labeled `TODO`. Print and submit the PDF. Do not submit the source code.

Create the TX and RX objects

Following the initial lab, set the mode to the desired configuration (loopback, two hosts with single host, or two hosts with two hosts)

```
% TODO:  Set parameters
% Set to true for loopback, else set to false
loopback = false;

% Select to run TX and RX
runTx = true;
runRx = true;
```

In case you do not have access to an SDR, I have pre-recorded some samples that you can use.

```
usePreRecorded = false;  % Set to true to use pre-recorded
```

Plug one or two Plutos into the USB ports of your computer. If it is correctly working, in each device the Ready light should be solid on and the LED1 light should be blinking. Now run the following command will detect the devices and create the TX and RX objects.

```
% clear previous instances
clear tx rx

% add path to the common directory where the function is found
addpath('..\common');
```

```
% Parameters
sampleRate = 30.72e6;
nsampsFrame = 2^12;


% Run the creation function.  Skip this if we are using pre-recorded
% samples
if ~usePreRecorded
    [tx, rx] = plutoCreateTxRx(createTx = runTx, createRx = runRx, loopback = loopback, ...
        nsampsFrame = nsampsFrame, sampleRate = sampleRate);
end
```

Sending a Continous Wave

Send a complex exponential as in the previous lab.

```
% Set the normalized digital frequency
nu0 = 4/nsampsFrame;

% TODO:  Create a digital signal the length of one frame with a digital frequency
% of nu0
%    x = ...

if runTx && ~usePreRecorded
    % TODO:  Use the tx.release and tx.transmitRelease commands to
    % continuously send x
end

% If not running the RX, stop the live script now
if ~runRx
    return;
end
```

Print the following parameters:

- The normalized digital frequency (`nu0`)
- `f0` = The baseband frequency in Hz (compute this from `nu0` and the sample rate)
- `` `f0pb = ` `` The passband frequency of the sinusoid. Compute this from `tx.CenterFrequency`

```
% TODO
```

Capture and Plot the Sinusoid

Follow the previous lab to capture one frame of samples and convert to floating point. Plot the resulting waveform vs. time in us. If we are using pre-recorded samples, we will load them from a file.

```
nbits = 12;               % number of ADC bits
fullScale = 2^(nbits-1);  % full scale

if usePreRecorded
    % Loads the pre-recorded samples
    load freqSamples;
else
```

```
    % TODO:  Capture data
    %   r = rx.capture(...)


    % TODO:  Scale to floating point
    %   r = ...
end

% TODO:  Plot the real and imaginary RX samples vs. time in micro-seconds
```

We will save the data for later in case we want to run the remainder of the lab without the Pluto.

```
if ~usePreRecorded
   centerFreq = rx.CenterFrequency;
   save freqSamples r centerFreq
end
```

Estimating the RX Frequency and CFO via the Correlation Method

The RX signal should also approximately be a complex exponential. We will begin by estimating the signal's frequency. One simple method is via 'correlation'. Suppose we have a singal:

```
r[n] = c*exp(2*pi*1i*nu*n) + noise
```

where `nu` is an unknown digital frequency and `c` is an unknown complex gain. The **correlation method** estimates the frequency `nu` via the steps:

```
z = \sum_{n=1}^nr r[n+1]*conj( r[n] )
nuest = angle(z)/2/pi
```

You can verify that if there is no noise, the correlation method will return `nuest=nu`.

TODO: Modify the code in `estFreq.m` under the section "`correlaton" to implement the correlation method.` You can ignore the other parts for now. Then, run the function `estFreq()` to get the correlation frequency estimate on the TX and RX signals. Print the following:

- `nu0:` True TX digital frequency
- `nuestTx:` Estimated TX digital frequency (this should = `nu0`)
- `nuestRx`: Estimated RX digital frequeny

```
% Get the correlation estimate frequency of the TX and RX
nuestTx = estFreq(x, method='correlation');
nuestRx = estFreq(r, method='correlation');


% TODO:  Print the estimated digital frequencies above
```

You mayl see that the estimated TX and RX frequencies are slightly off. Print:

- `cfo`: The carrer frequency offset in Hz (the difference between the TX and RX frequencies)
- `ppm`: The error in PPM defined as: `ppm = cfo / centerFreq * 1e6` . The devices are rated for about ~10 ppm error. So, the total error should be `|ppm| < 20`.

```
% TODO:
%   cfo = ...
%   ppm = ...
```

Estimating the Amplitude of the Complex Exponential via Least Squares

We next estiamte the complex amplitude of the sinusoid. With our estimate `nu=nuestRx` the RX signal should be of the form:

```
r[n] = c*exp(2*pi*1i*nuestRx*n) + noise
```

The remaining unknown is the complex amplitude, `c`. This can be estiamted via *least squares:* If we define `u[n] = exp(2*pi*1i*n*nuestRx)`, the least squares estiamte for `c` is the minimum of the function:

```
J(c) = \sum_n | r[n] - c*u[n] |^2
```

The LS estimate is discussed in detail in any machine learning class. In MATLAB, we can ccompute the LS estimate via the backslash operator `cest = u \ r`.

TODO: Modify the end of the estFreq function to compute the LS estimate and return the estimated `cest`. Then, compute the estimated signal `rhat[n] = cest*exp(2*pi*1i*nuestRx*n).` Plot the real components of `r` and `rhat`. They should match well.

```
% Run the frequency estimation with the frequency and amplitude estimation
[nuestRx, cest] = estFreq(r, method='correlation');

% TODO:  Compute and plot the estimated RX sinusoid vs. time in
% micro-seconds
%   rhat = ...
%   plot(...)
```

Estimating the RX Frequency using an FFT

The correlation method is simple but does not tend to work well when there is a large amount of noise. If you like, you can verify this with a simple simulation. An alternate method is to use an FFT. Specifically, suppose we take an FFT magnitude.

```
R = abs(fft(r));
```

If the signal `r` has `nr` samples and has a normalized digital frequency `nu`, then the FFT should have a peak near `nu*nr`. So, we can use the estimate for the normalized frequency:

```
nuest = (ind-1)/nr
if nuest > 0.5
    nuest = nuest - 1;
end
```

where `ind` is the index where `R(ind)` is maximized. We subtract one since MATLAB starts the indices at 0. The mapping `nuest = nuest-1` makes sure estimates are in the range `nuest \in [-0.5,0.5]`.

TODO: Implement the FFT method in the section of `estFreq` and compute the estimated frequency. For now, ignore the components on oversampling. Print the estimated frequency along with correlation estimate frequency. You will see that there is an error since the FFT method must round the estiate to the nearest `1/nr`.

```
% Run the frequency estimation with the frequency and amplitude estimation
[nuestRxFFT, cest] = estFreq(r, method='fft');

% TODO:  Print the correlation and FFT estimates
```

To reduce the errors from rounding, we can oversample the received signal. Specifically, we take an oversampling ratio, `oversample`, usually 2 or 4. Then, we construct a zero-padded signal:

```
rov[n] = r[n], n=1,...,nr

       = 0     n=nr+1,..., nr*oversample
```

Then, we estimate the frequency on the oversampled signal. The estimated frequency will be:

```
nuest = (ind-1)/nr/oversample
```

TODO: Implement the FFT method with oversampling. Then, test the code as before. You should see a more accurate estimate

```
% Run the frequency estimation with the frequency and amplitude estimation
[nuestRxFFT, cest] = estFreq(r, method='fft', oversample=4);

% TODO:  Print the correlation and FFT estimates
```

Advanced Topics

We will discuss in the subsequent labs how to correct for the frequency offset. But, if you are interested, there are two more advanced items you can pursue:

- Add a gradient descent method to solve the non-linear least squares estimate of the magnitude and frequency. Non-linear least squares and gradient descent are covered in the ML class and you can use a simple algorithm.
- In general, the phase and frequency vary over time due to a process known as phase noise. Phase noise is particularly important in high frequency RF systems. You can run the code above to track the phase and frequency over time to see how the phases changes.