EE194/EE290C HW1 -- Simulating radio basics in Matlab -- Spring 2018

In this HW we will:

1. Generate a data signal: a random binary vector representing frequency vs time where 0 means -250kHz and 1 means +250kHz. The binary values should change at 1Mbit/s (for GFSK, this is the same as 1Msymbol/s), so each binary value in your original vector represents 1 μs of time. Be sure to oversample it so you get at least 256 points per bit.

2. Use that to modulate the frequency of an oscillator (performing up-conversion via direct modulation). In the chip this will be ~2.4GHz but in this HW it'll be 24MHz to make the matlab processing faster,

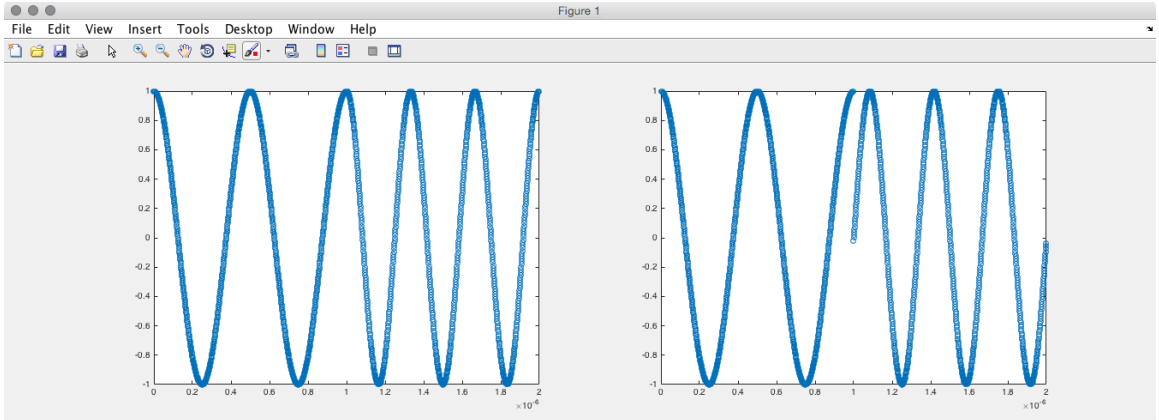
3. Pretend the signal travels over the air perfectly; pass that through a mixer by multiplying by another (24-2) MHz sinusoid (The receiver Local Oscillator frequency is f_{RF}-f_{IF}).

4. Filter with an appropriate low-pass filter. Multiplying two cosines yields the sum and difference of their frequencies; we need to get rid of the sum component.

Assignment 1: Write a matlab script that performs these functions, and plot at the time-domain waveforms (amplitude vs time) and frequency-domain spectrum (including negative and positive frequency) at each of these 4 steps to see how the signal evolves at each point. Then plot freq vs time at step 1 and step 4 to make sure the input and output match!

Spec details:
The oscillator should be modulated at 24MHz +/- 250kHz at 1Msymbol/s (in BLE this is the same as saying 1MHz -- the frequency could change every 1us.). In reality, this will be 2.4GHz + BLE channel offset +/- 250kHz. You can accomplish this however you want, but make sure it doesn't have any phase discontinuities. For instance, here's a 1V amplitude time-domain signal transitioning from 2MHz to 3MHz at t=1us:

```
t=0:1e-9:2e-6-1e-9;
subplot(1,2,1)
plot(t,[cos(2*pi*2e6*t(1:end/2-1)) cos(2*pi*3e6*t(end/2:end))],'o')
subplot(1,2,2)
plot(t,[cos(2*pi*2e6*t(1:end/2)) cos(2*pi*3e6*t(end/8:5*end/8-1))],'o')
```



On the left there is a smooth transition but on the right there's an abrupt phase jump. Both switch from f=2MHz to f=3MHz but the phase jump will result in all sorts of noisy emissions in the frequency domain and a real circuit will have a hard time jumping so abruptly besides.

We suggest starting with random binary sequence, switching at 1Mbit/s, then writing a loop to go through that sequence to generate a time-domain waveform with something like `cos(2pi*(f_RF+f_modulation)).`

Another good way is to calculate the phase of the signal over time and take `cos()` of that afterward.

FFT:

The code included with the matlab help page on setting up FFT works great:
`https://www.mathworks.com/help/matlab/ref/fft.html`
But try to plot double-sided instead of single-sided FFT. There's some sample code in `http://inst.eecs.berkeley.edu/~ee290c/sp17/hw/MatlabSignalsAndNoise.m` too -- but most of that file isn't applicable to this homework.

Filtering:

Any low-pass (bandpass?) filter should be fine; experiment with what you can generate with matlab's `fdatool` GUI. A good place to start is the Butterworth. Once you design a filter, export it with File > Generate MATLAB Code > Filter Design Function. Save the .m, open it, and copy that code into yours.

Visualize a filter with `fvtool(Hd)`, where `Hd` is the object created by `dfilt.dffir()`. Apply it to a signal with `filter(Hd,mySignal)`.

Getting started:
If you're lost, here's the first few lines of our sample program:

```
% Simulation parameters
```

```
N=2^13;                      % Number of points to simulate
fs=256e6;                    % Sampling frequency (N/(fs/datarate) must
be integer)
deltat=1/fs;
deltaf=fs/N;
time=0:deltat:(N-1)*deltat; % Simulation time vector
datarate=1e6; % 1Msymbol/s
deltafsk=0.5*datarate/2;
f_RF=24e6;

% Generate random binary bits
bits=round(rand(1,N/(fs/datarate)));
```

Assignment 2: Redo #1 with a Gaussian filter applied to the bitstream. This means a freq vs time plot would look like a smooth change from one freq to another instead of sudden jumps between +250kHz and -250kHz. Note this is different from *phase discontinuities* -- now you're trying to eliminate *frequency discontinuities* too. One way to do this is take your vector of frequency vs time representing 0101... and apply a gaussian filter.

Matlab's Gaussian filter can be invoked with `gaussdesign()`. The first parameter for this function is BT, or B*T -- this is the filter bandwidth times the bit period. Bit period you already know, and the filter should be the total +/- frequency spacing. The second parameter is "span" and we suggest leaving it at its default of 3. The third parameter depends on your code.

A huge demo of Gaussian filters for various bandwidths and ideal vs nonideal approximations will pop up if you run `GaussianFilterExample`. FYI the "a" term in Figures 2 and 4 is b*t.

Assignment 3: Add noise to your gaussian bitstream, then redo #2. Make some noise with:

```
noise = sigma*rand(size(mySignal));
```

And add it to your gaussian signal:

```
newsignal = mySignal+noise;
```

This is called AWGN -- additive white gaussian noise. Try a few different `sigma` values (0.1 is a good place to start). How large can it be before you can't see the signal in your freq vs time plot after step 4 anymore?

IMPORTANT NOTE:
This is a plot-heavy writeup so use plots and axis labels efficiently, e.g., `subplot()` -- I don't want to see a 50-page word doc with 1 barely-labeled plot per page.