# Project 4: GPS Signal Power Spectral Density

## Objectives

1. See examples of how the signal power spectrum is spread by different modulations
2. Generate sample-based power spectral density plots for several different GPS signals

## Collaboration

This is an individual lab. You are allowed to discuss any aspect of the lab with other students, and you may look at each other’s source code for debugging purposes. However, your programming must be your own (i.e., you may not copy or transcribe someone else’s program, in part or in whole).

## Overview

This week, you have seen examples of the different PRN codes used for GPS signals. In this project, you will be generating simulated signals with simulated PRN codes (with the chips being a random binary sequence, rather than actual PRN codes), and then calculating the power spectrum of the signals using standard signal processing techniques.

## Task 1: Generate Simulated C/A Code Signal

For this task, you should generate a simulated C/A code signals. For this simulation, you can go ahead and treat the C/A code as a random binary codes (values of 1 and -1) which does not repeat [start with np.random.rand()]. The fact that the C/A code repeats every millisecond will have an effect on the power spectral density, in that it will become a line spectrum rather than a continuous spectrum, but for the purposes of this project, we are not going to worry about that. Also, you will be simulating the GPS signal not at a center frequency of 1575.42 MHz, but at a lower frequency (for computational reasons—simulation of the true sampled signal at RF would require an extremely high sample rate and many samples.

I recommend that you sample the signal using the following “settings”:

Center frequency (f\_0): 30MHz (rather than 1575.42MHz for GPS)

Sampling frequency (f\_s): 200MHz

Length of sampled data to simulate (t\_max): 0.1 seconds

You should generate a carrier signal at the center frequency, and then modulate it with sampled random binary sequences (which essentially means do an element-by-element multiply between the two, as long as the sampled random binary sequences are implemented as +1/-1 values).

Note that you need to generate samples of the random binary sequences, taking into account both the chipping rates and the sampling frequency. One way to do this is to generate a time vector, which gives the time of each sample:

time\_array = np.arange(0, t\_max, 1/f\_s)

Then, what you need to figure out is the chip number for each time value in this vector, which can be calculated based on the time array and the chipping rate (f\_chip):

chip\_num = np.floor(time\_array \* f\_chip).astype(‘int’)

Finally, you can the generate the sampling of the PRN codes using:

prn\_sampled = prn\_code[chip\_num]

where prn\_code is the random binary sequence representing the PRN code sequence.

When you are done, you should have two signals:

1. Carrier only
2. C/A code signal modulated onto the carrier

## Task 2: Calculate Power Spectral Density for C/A Code

Now, take the signals calculated in Task 1 and calculate a power spectral density (PSD). Please use the FFT command to accomplish this. For example:

fft\_signal = np.fft.fft(signal)/len(signal)

fft\_signal = abs(fft\_signal[range(int(len(signal)/2))])

will produce a normalized fft output of signal and retrieve the correct power values for plotting. The values for the frequency axis can be obtained by:

len\_time = len(time\_array)

values = np.arange(int(len\_time/2))

time\_period = len\_time / f\_s

freq\_axis = values / time\_period

In order to clear up the noise in the C/A code PSD plot, I recommend that you apply a windowing filter (from the scipy.signal library), which replaces each point with an average of the nearest N points:

fft\_filtered = scipy.signal.filtfilt(np.ones(N), 1, fft\_signal)

I recommend using an N of 500. (Note that you probably don’t want to do this with the carrier-only plot).

Plot the power spectral density of both the carrier-only signal, as well as the C/A code signal modulated onto the carrier. The C/A code modulation plot should have a similar nature to the power spectrum shown in the slides (although the slides are logarithmic scale in the y-axis).

## Task 3: Calculate Power Spectral Density for P(Y) Code

Now, repeat the process, except simulate P(Y) code, rather than C/A code. The main difference is that the chipping rate should be 10x faster. However, the L1 P(Y) code is also 3dB less power than the C/A code, so you should multiply the signal by 1/sqrt(2) in order to properly scale it before calculating the spectrum. Plot this in a separate plot from the C/A code. Does it look much different? Make sure you observe the scale on the x-axis. (There is no need to replot the carrier-only from here on out).

## Task 4: Calculate Power Spectral Density for M-Code

Repeat the process, only implement a simulated M-Code modulation. M-Code is a BOC(10,5) signal, which means that it is modulated by two different signals:

1. A square wave at 101.023x106 (equal length 1 and -1 portions within the square wave period)
2. A PRN code with a chipping rate of 51.023x106. (For this project, you can simulate the M-Code PRN code with random values, just like with the C/A and P codes).

More details on an M-Code signal can be found at:

<https://gssc.esa.int/navipedia/index.php/Binary_Offset_Carrier_(BOC)>

Plot the M-Code power spectral density, after applying the window filter, as with the C/A code and P code examples.

## What to Turn In

For this project, turn in the following set of plots, properly labeled:

1. PSD of carrier-only signal
2. PSD of window filtered C/A code, P code, and M code all on one plot, plotted on a linear scale.
3. PSD of window filtered C/A code, P code, and M code all on one plot, plotted on a logarithmic scale on the y-axis. Keep the x-axis linear (you can use semilogy from matplotlib.pyplot to do this). Set the y-axis limits to generate a plot that looks similar to the power spectral density plot of the signals given in the notes.
4. Also turn in the Python code used to generate the signals and plots.

Proper labeling means that you (1) say what the axis is with words (e.g., “Frequency”) and (2) provide the units (e.g., “MHz”). Do this for both the x axis and the y axis.

## Grading

You will be graded primarily upon accuracy and correct labeling.