Lab6_notebook_sample

April 25, 2022

1 Lab 6 Sample Notebook: RTL-SDR

```
# Some installs are required to use the RTLSDR dongle in Python:
# rtlsdr is a community package for connecting to the dongle
pip install pyrtlsdr
# A helper package that is integrated with scikit-dsp-comm
# This package will soon contain capture, but at the moment it is missing
pip install sdr_helper
```

[1]:
```
from numpy import *
from matplotlib.pyplot import *
%matplotlib inline
import sk_dsp_comm.sigsys as ss
import sk_dsp_comm.sdr_helper as sdrh
import sk_dsp_comm.fir_design_helper as fir_d
import sk_dsp_comm.digitalcom as dc
import scipy.signal as signal
from IPython.display import Audio, display
from IPython.display import Image, SVG

%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
#%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

[2]:
```
# package to allow interfacing directly to the RTLSDR radio hardware
import rtlsdr as rtlsdr1
```

[3]:
```
def capture(Tc,fo=88.7e6,fs=2.4e6,gain=40,device_index=0):
    """
    Capture an array of complex radio samples:

    capture(Tc,fo=88.7e6,fs=2.4e6,gain=40,device_index=0)
    """
    # Setup SDR
    sdr1 = rtlsdr1.RtlSdr(device_index) #create a RtlSdr object
    #sdr.get_tuner_type()
    sdr1.sample_rate = fs
    sdr1.center_freq = fo
    #sdr.gain = 'auto'
    sdr1.gain = gain
    # Capture samples
    Nc = np.ceil(Tc*fs)
    x = sdr1.read_samples(Nc)
    sdr1.close()
    return x
```

1.0.1 Capture 5 Seconds of I/Q Samples at 88.7 MHz (KCME, Colorado Springs)

```
[4]: Tc = 5
     # Tc, fo=88700000.0, fs=2400000.0, gain=40, device_index=0
     x = capture(Tc, fo=88700000.0, fs=2400000.0, gain=40)
```

```
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
```

1.0.2 Archive the capture in a wave file

Here we use the function `complex2wav()` to repurpose wav audio file standard as a means to store a complex signal array (real and imaginary parts) into the left and right audio channels. In the below notice that the sampling rate of 2.4 Msps is stored in the wav file header.

```
[19]: # sdrh.complex2wav('KCME_IQ.wav',2400000,x)
      sdrh.complex2wav('KCME_IQ_test2022.wav',2400000,x)
```

```
Saved as binary wav file with (I,Q)<=>(L,R)
```

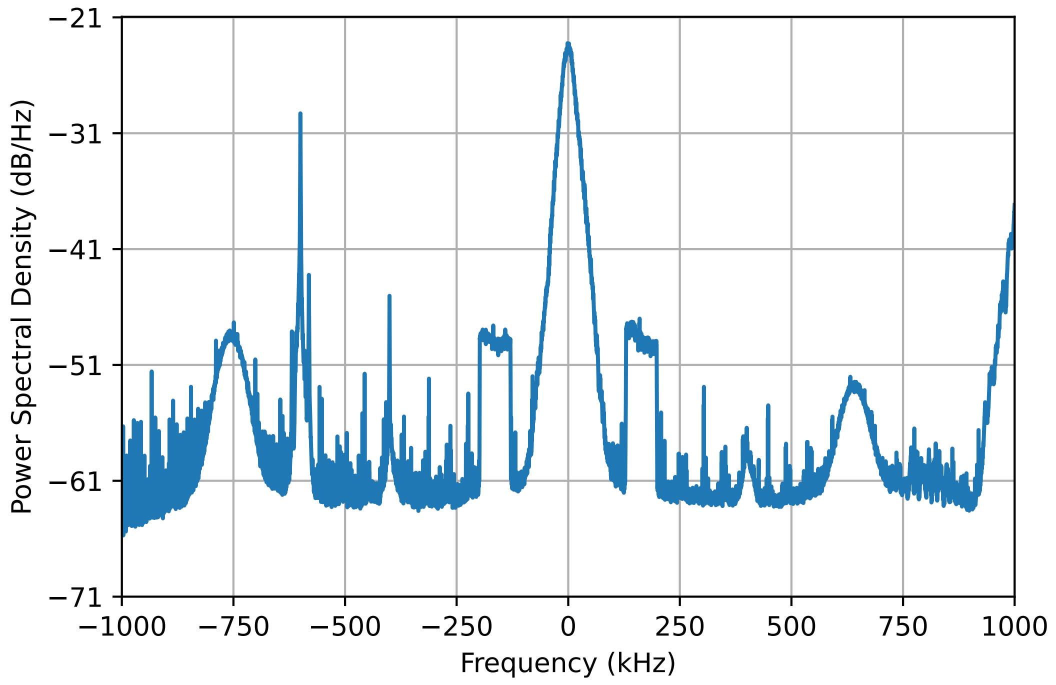
1.0.3 Restore the IQ Samples into a Complex Baseband Array

If need be, restore an archived capture back into the workspace. The sampling rate will be 2.4 Msps as that is how it was originall captured and saved.

```
[ ]: # fs, x = sdrh.wav2complex('KCME_IQ.wav')
     fs, x = sdrh.wav2complex('KCME_IQ_test2022.wav')
```

1.0.4 Look at the Complex Baseband Spectrum from the RTL-SDR

```
[5]: psd(x,2**14,2400);
     xlim(-1000,1000);
     xlabel('Frequency (kHz)');
```



1.0.5 Process Samples Through a Mono FM Demodulator

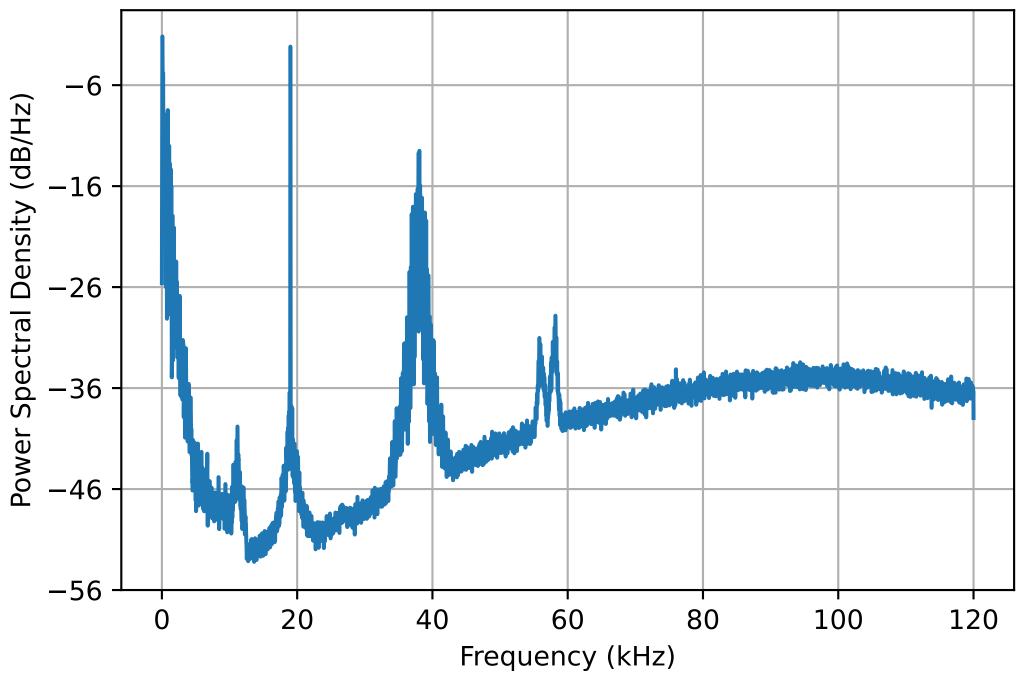
[6]:
```
# z_bb, z_out = sdrh.mono_fm(x,fs=2.4e6,file_name='KCME_test2022_new.wav')
z_bb, theta, y_lpr, y_lmr, z_out = \
              sdrh.stereo_fm(x,fs=2.4e6,file_name='KCME_test2022_new.wav')
```

```
Done!
```

[7]:
```
# Audio('KCME.wav')  # From file
# Audio('KCME_test2022_mono.wav')  # From file
# Audio('KCME_test2022_stereo.wav')  # From file
```

[7]:
```
<IPython.lib.display.Audio object>
```

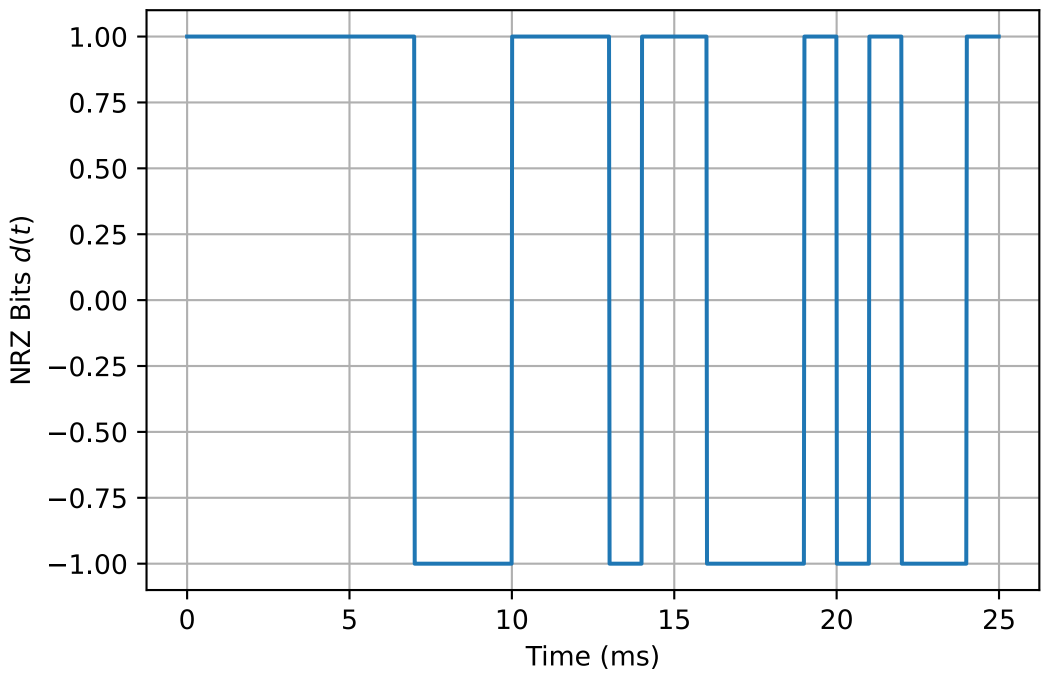
[8]:
```
psd(z_bb,2**14,2400/10.);
xlabel('Frequency (kHz)');
```



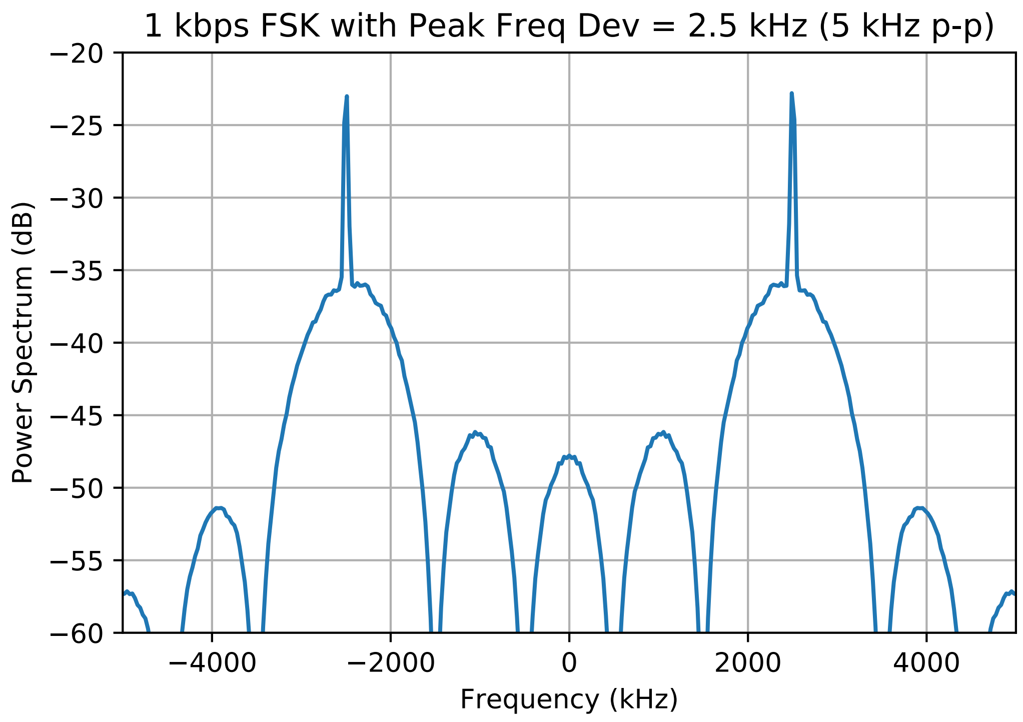
1.0.6 FSK Simulation

[38]:
```
Ns = 120; Nbits = 1000; Rb = 1000; Df = 2500.0; fs = Rb*Ns;
data = ss.pn_gen(Nbits,7)
#data = ss.pn_gen(Nbits,7)
waveform = signal.lfilter(ones(Ns),1,ss.upsample(data,Ns))
x, b = ss.nrz_bits2(data,Ns,'rect')
n = arange(Ns*Nbits)
xc = exp(1j*2*pi*Df*x/fs*n);
```

[39]:
```
plot(n[:3000]/fs*1000,x[:3000])
xlabel(r'Time (ms)')
ylabel(r'NRZ Bits $d(t)$')
grid();
```



[6]:
```
Pxc,fxc = ss.my_psd(xc,2**12,fs);
plot(fxc,10*log10(Pxc))
xlim([-5000,5000])
ylim([-60,-20])
title(r'1 kbps FSK with Peak Freq Dev = 2.5 kHz (5 kHz p-p)')
xlabel(r'Frequency (kHz)')
ylabel(r'Power Spectrum (dB)')
grid();
savefig('FSK1kbps2_5.pdf')
```



1.0.7 FSK Capture $\Delta f = 2.5$ kHz

[333]:
```
Tc = 20 # or 10s
# Tc, fo=88700000.0, fs=2400000.0, gain=40, device_index=0
x_FSK2_5 = capture(Tc, fo=70000000.0, fs=2400000.0, gain=40)
```

[ ]:
```
psd(x_FSK2_5,2**14,2400);
xlim(-50,50);
xlabel('Frequency (kHz)');
```

Save and Restore IQ Data

[113]:
```
sdrh.complex2wav('FSK_70M_IQ2_5.wav',2400000,x_FSK2_5)
```

```
Saved as binary wav file with (I,Q)<=>(L,R)
```

[22]:
```
fs, x_FSK2_5 = sdrh.wav2complex('FSK_70M_IQ2_5.wav')
```

1.0.8 Processing the Capture

$N_2 = 5$ Final Decimation Stage Lowpass Filter

[27]:
```
b_FSK = fir_d.fir_remez_lpf(1000,4000,.1,60,2.4e6/10)
```

```
Remez filter taps = 205.
```

```
[ ]: # FSK demod code
```

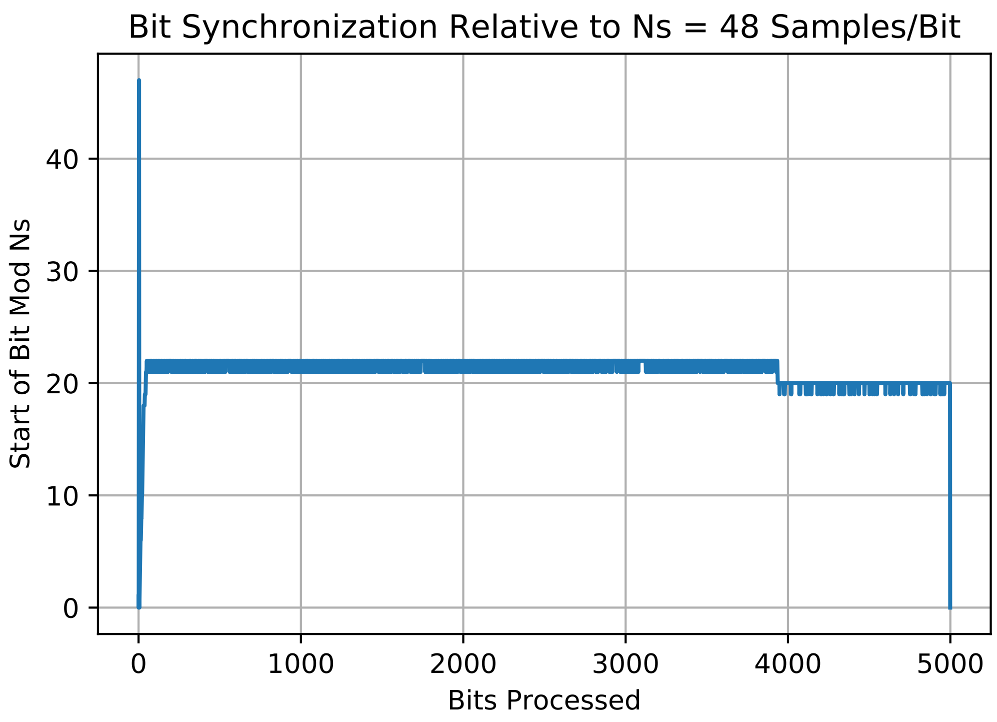
Bit Synch to Received Data Bits

```
[28]: data_hat, clk, track = sdrh.sccs_bit_sync(data_hat_FSK2_5,48)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8396/1219474186.py in <module>
----> 1 data_hat, clk, track = sdrh.sccs_bit_sync(data_hat_FSK2_5,48)

NameError: name 'data_hat_FSK2_5' is not defined
```

```
[63]: plot(track)
      title(r'Bit Synchronization Relative to Ns = 48 Samples/Bit')
      ylabel(r'Start of Bit Mod Ns')
      xlabel(r'Bits Processed')
      grid();
```



Prepare an Equal Length Stream Transmitted PN7 Bits

```
[26]: def PN7_33600A(Nbits):
          """
```

```
    Create an Nbits stream equivalent of the Keysight 33600A PN7 PBRS

    Mark Wickert April 2019
    """
    m = 7
    PN7_oct_taps = [0o211,0o217,0o235,0o367,0o277,0o325,0o203,0o313,0o345]
    taps = np.array([1, 0, 0, 0, 0, 0, 1, 1])
    sr = np.ones(m)
    # M-squence length is:
    Q = 2**m - 1
    c = np.zeros(Q)
    for n in range(Q):
        tap_xor = 0
        c[n] = sr[-1]
        for k in range(1,m):
            if taps[k] == 1:
                tap_xor = np.bitwise_xor(tap_xor,np.
 ↪bitwise_xor(int(sr[-1]),int(sr[m-1-k])))
        sr[1:] = sr[:-1]
        sr[0] = tap_xor
    PN7 = c[::-1] # reverse the sequence
    PN7_stream = zeros(Nbits)
    for n in range(Nbits):
        PN7_stream[n] = PN7[mod(n,127)]
    return PN7_stream
```

Generate Tx Reference Bits and Error Detect

[27]:
```
tx_bits = PN7_33600A(len(data_hat))
N_bits, N_errors = dc.bit_errors(tx_bits,int16((data_hat[5:]+1)/2))
print('N_bits = %d, N_errors = %d, BEP = %1.2e' % (N_bits,N_errors, N_errors/
 ↪N_bits))
```

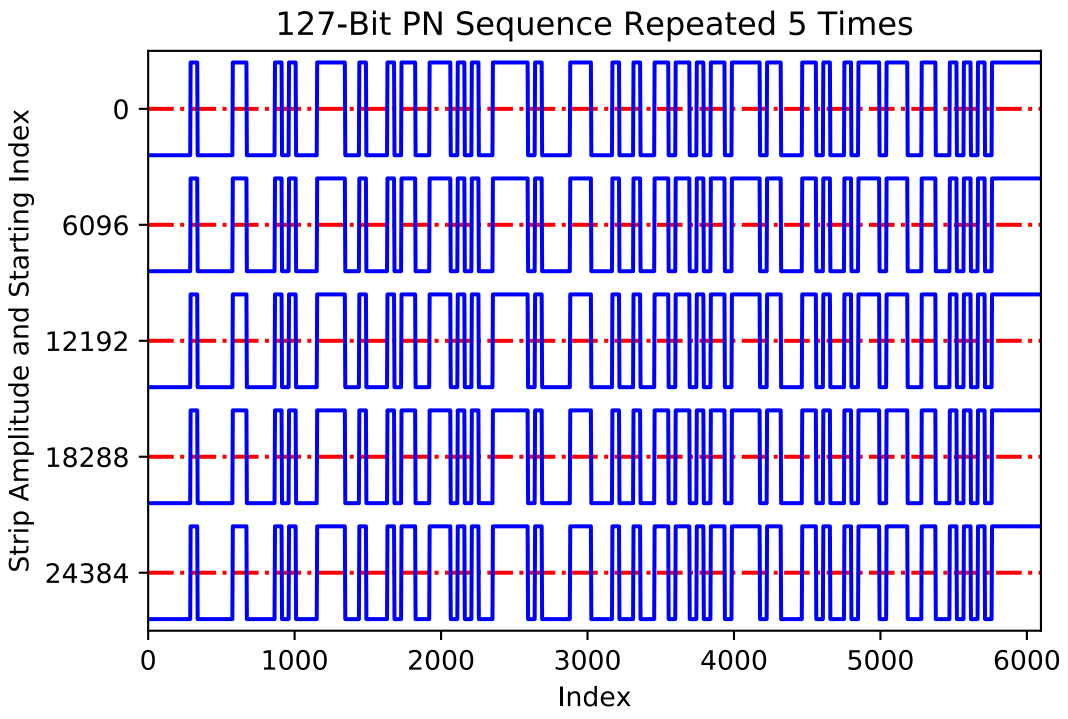
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8396/2615299031.py in <module>
----> 1 tx_bits = PN7_33600A(len(data_hat))
      2 N_bits, N_errors = dc.bit_errors(tx_bits,int16((data_hat[5:]+1)/2))
      3 print('N_bits = %d, N_errors = %d, BEP = %1.2e' % (N_bits,N_errors,␣
 ↪N_errors/N_bits))

NameError: name 'data_hat' is not defined
```

1.1 Other Support Code

1.1.1 Using strips() for Plotting a Long Waveform

[40]:
```
#data = ss.PN_gen(127*5,7)
# Make 5 periods of the 33600A PN7
data = PN7_33600A(5*127) # create 5 periods of a 127 bit sequence
x, b = ss.NRZ_bits2(data,48,'rect')
dc.strips(x,48*127);
title(r'127-Bit PN Sequence Repeated 5 Times')
savefig('strips_example.svg')
```



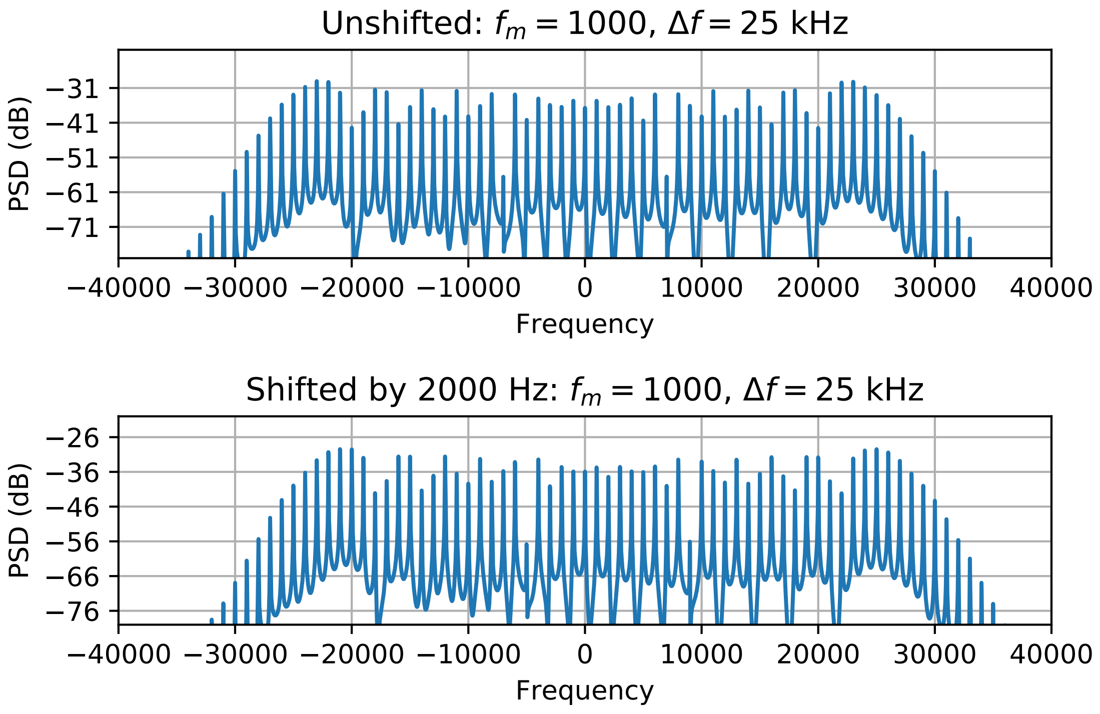
1.1.2 Sinusoidal FM Transmitter

[20]:
```
def bb_fm_tx(m,fd,fs):
    '''
    A simple complex baseband FM modulator

    Mark Wickert April 2019
    '''
    return exp(1j*2*pi*fd/float(fs)*cumsum(m))
```

[62]:
```
# Here m is a 1kHz tone with fs = 240kHz, fd = 25kHz
fm = 1000.; fs = 240e3; fd = 25e3;
```

```
n = arange(0,10000)
x_tx = bb_fm_tx(cos(2*pi*fm/fs*n),fd,fs)
subplot(211)
psd(x_tx,2**14,fs);
title(r'Unshifted: $f_m = 1000$, $\Delta f = 25$ kHz')
ylabel(r'PSD (dB)')
xlim([-40e3, 40e3])
ylim([-80, -20])
subplot(212)
# Include a tuning frequency error of +2000 Hz
x_tx *= exp(1j*2*pi*2000./fs*n)
psd(x_tx,2**14,fs);
title(r'Shifted by 2000 Hz: $f_m = 1000$, $\Delta f = 25$ kHz')
ylabel(r'PSD (dB)')
xlim([-40e3, 40e3])
ylim([-80, -20])
tight_layout()
savefig('fm_tx_Python.svg')
```



```
[ ]:
```