Lab 3 Sample Notebook

```
%pylab inline
#%pylab notebook
#%matplotlib qt
import sk_dsp_comm.sigsys as ss
import scipy.signal as signal
import ipywidgets as widgets
from ipywidgets import interact, interactive, fixed, interact_manual
from IPython.display import Audio, display
from IPython.display import Image, SVG
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
pylab.rcParams['savefig.dpi'] = 100 # default 72
#pylab.rcParams['figure.figsize'] = (6.0, 4.0) # default (6,4)
#%config InlineBackend.figure_formats=['png'] # default for inline viewing
%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
#%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
#<div style="page-break-after: always;"></div> #page breaks after in Typora
```

Support Functions

```
def line_spectra_dBm(fn,Xn,floor_dBm = -60,R0 = 50,lwidth=2):
    """
    Plot the one-sided line spectra from Fourier coefficients in dBm.

    Inputs
    ------
    fn = harmonic frequencies correspinding to Xn's
    Xn = pulse train FS coefficients
    floor_dBm = spectrum floor in dBm
    R0 = impedance (default 50 ohms)

    Returns
    -------
    Sx_dBm = One-sided bandpass lines as dBm levels

    Mark Wickert February 2019
    """
```

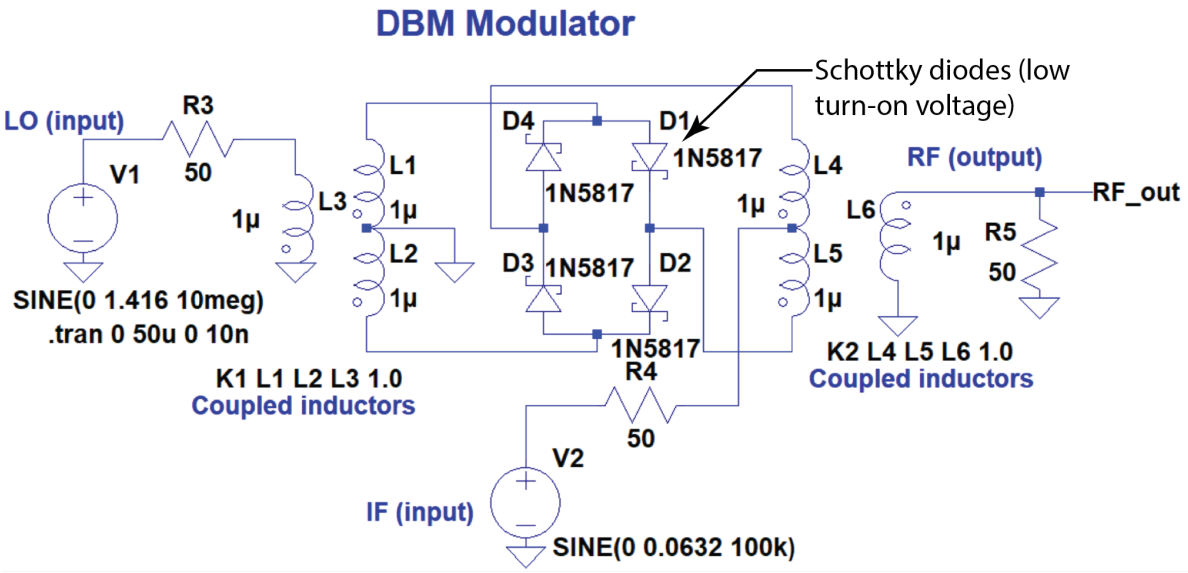
```
    Xn_dBm = abs(Xn*sqrt(1000/(2*R0)))
    Xn_dBm[0] = abs(Xn[0])*sqrt(1000/(R0))

    ss.line_spectra(fn,Xn_dBm,mode='magdB',sides=1,lwidth=lwidth,floor_dB=floor_dBm)
    ylabel(r'Power (dBm)')
    xlabel(r'Frequency (Hz)')
    title(r'Line Spectra PSD')
    Sx_dBm = 20*log10(2*Xn_dBm)
    Sx_dBm[0] -= 6.02
    return Sx_dBm
```

Double Balanced Mixer Modeling

Consider both LTSpice and a behavioral level model in Python.

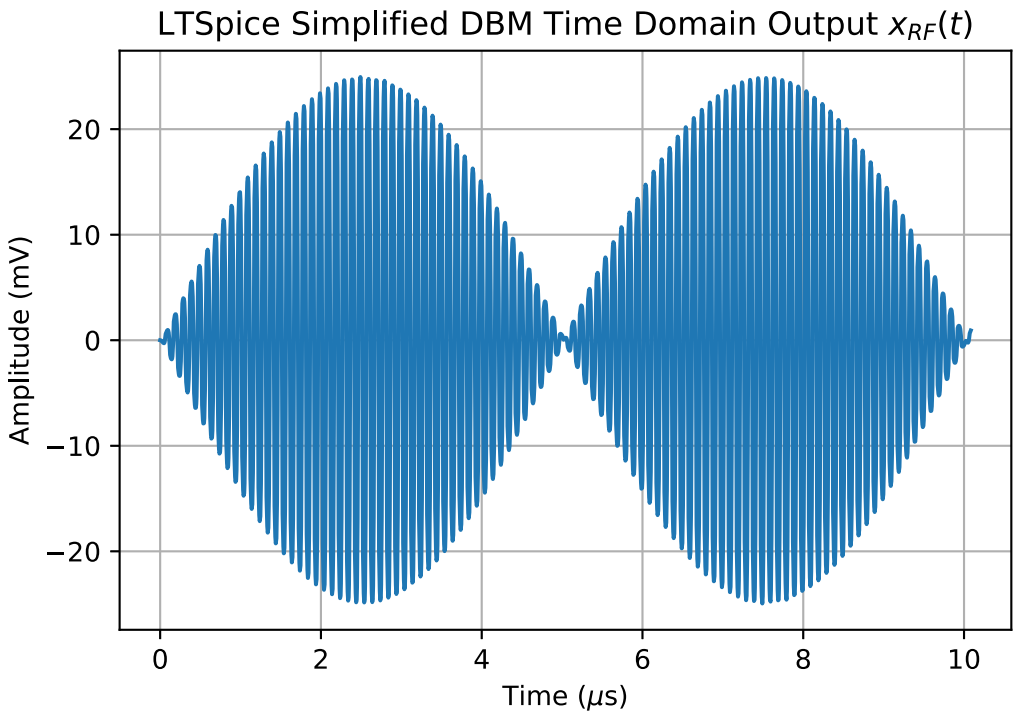
LTSpice DBM Circuit Simulation

Import time domain and spectrum results from a simplified DBM circuit simulation.



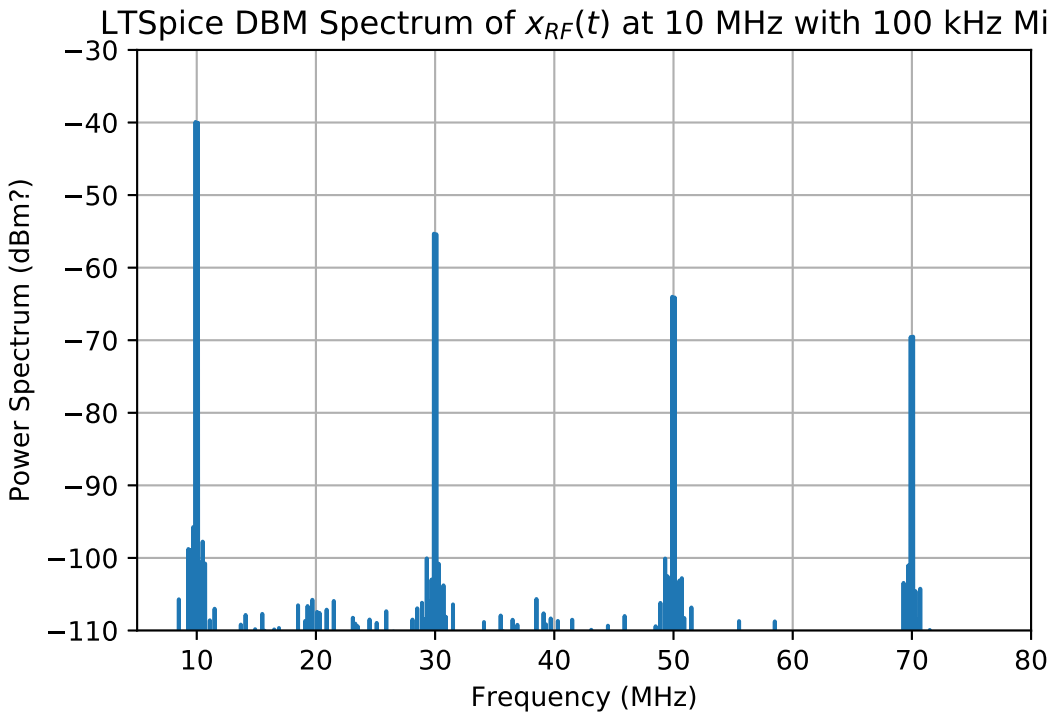

```
# Skip the first 32 rows, then skip the last row that contains 'END'
t_spice, RF_out = loadtxt('DBM_mod_time.csv',delimiter=',',skiprows=1,
                  usecols=(0,1),unpack=True)
```

```
plot(t_spice[:3200]*1e6,RF_out[:3200]*1e3)
title(r'LTSpice Simplified DBM Time Domain Output $x_{RF}(t)$')
ylabel(r'Amplitude (mV)')
xlabel(r'Time ($\mu$s)')
grid();
```




```
# Skip the first 32 rows, then skip the last row that contains 'END'
f_spice, RF_out_psd = loadtxt('DBM_mod_spec.csv',delimiter=',',skiprows=1,
                usecols=(0,1),unpack=True)
```

```
plot(f_spice/1e6,RF_out_psd)
title(r'LTSpice DBM Spectrum of $x_{RF}(t)$ at 10 MHz with 100 kHz Mix')
ylabel(r'Power Spectrum (dBm?)')
xlabel(r'Frequency (MHz)');
ylim([-110,-30])
xlim([5,80])
grid();
```



Python Behavioral Level Modeling

The behavioral level model is motivated by the understanding that the mixer high level signal, typically the LO port, drives the diode ring into switching mode. The LO signal then appears approximately as a squared wave signal multiplying the low level input signal at the IF port or RF port. The first step in development of this model is to understand a diode clipper built using Schottky diodes.

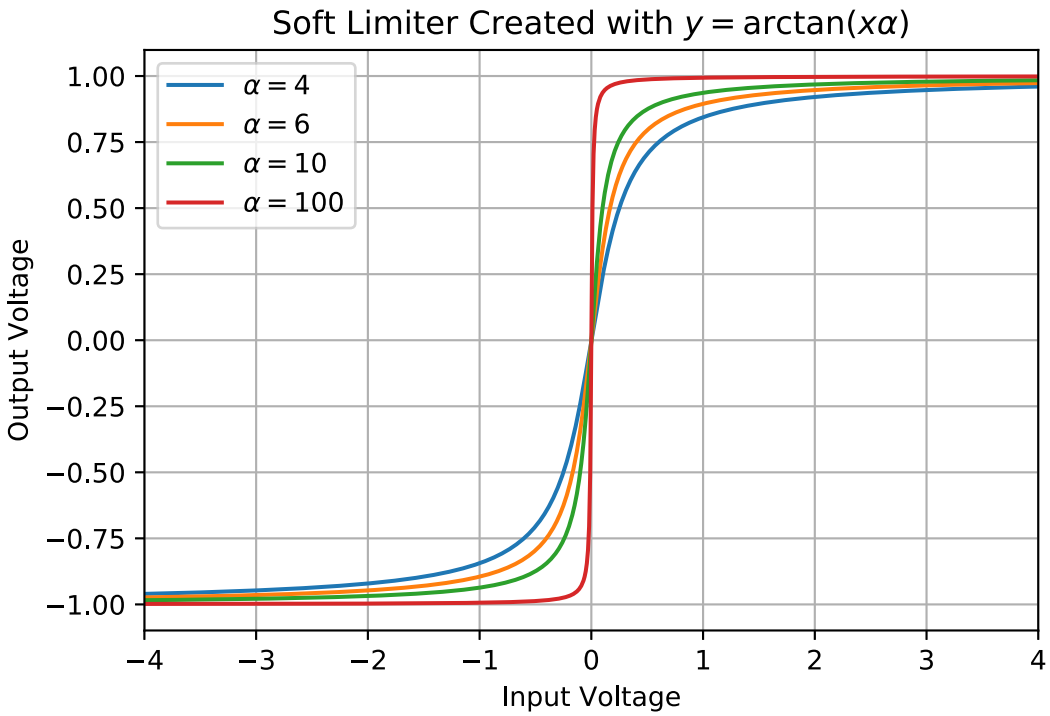
Soft Limiter and a Diode Clipping Model

Use the $\arctan()$ function to approximate the *soft* clipping action of the Schottky diode ring found in the DBM.

$$y = \frac{2}{\pi}\arctan(\alpha \cdot x) \tag{1}$$

First consider the input/output characteristic:

```
x = arange(-5,5,.01)

for k in range(4):
    alpha = [4,6,10,100]
    y = arctan(x*alpha[k])*2/pi
    plot(x,y)
xlim([-4,4])
title(r'Soft Limiter Created with $y = \arctan(x\alpha)$')
ylabel(r'Output Voltage')
xlabel(r'Input Voltage')
legend((r'$\alpha = 4$',r'$\alpha = 6$',r'$\alpha = 10$',r'$\alpha = 100$'))
grid();
```

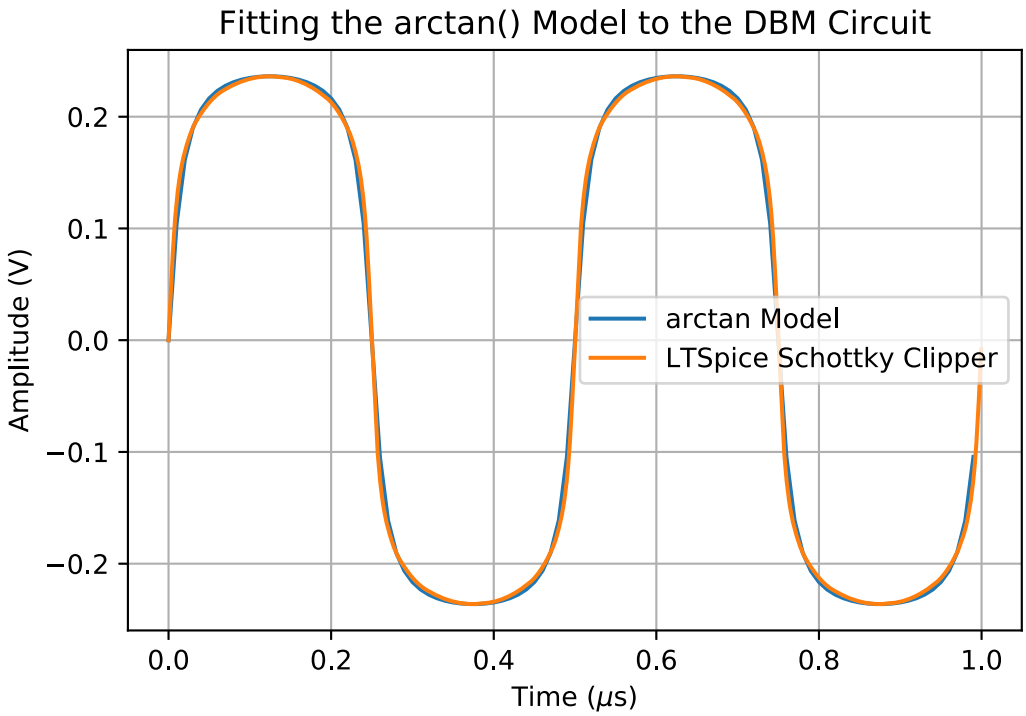


Simulate the simple diode clipper in LTSpice when driven by a single sinusoid. LTSpice simulation results are imported and then a reasonable α value is chosen for a +7 dBm LO drive sinuoid.

```
# Skip the first 32 rows, then skip the last row that contains 'END'
t_clip, clip_out = loadtxt('DiodeClipper.csv',delimiter=',',skiprows=1,
                usecols=(0,1),unpack=True)
```

```
# Behavioral level model using arctan as an amplitude limiter
t_behav = arange(0,1e-3,1e-5)
x_behav = 1.416*sin(2*pi*2000*t_behav)
y_behav = arctan(x_behav*4.0)*2/pi
y_behavs = y_behav*max(clip_out)/max(y_behav)
```

```
plot(t_behav*1e3,y_behavs)
plot(t_clip[:-1]*1e3,clip_out[:-1])
title(r'Fitting the $\arctan()$ Model to the DBM Circuit')
ylabel(r'Amplitude (V)')
xlabel(r'Time ($\mu$s)')
legend((r'$\arctan$ Model',r'LTSpice Schottky Clipper'))
#ylim ([-.4,.4])
grid();
```



From the above we see that $\alpha = 4.0$ is a good fit for the +7 dBm LO input. Define the DBM model with function `DBM_model()` having parameters `alpha` to control the soft limiter threshold and `conv_loss` to model the mixer conversion loss from the low level input relative to the desired sum and difference frequency outputs. Understand that the loss reflects the fact that the mixer converts the input signal IF/RF to additional output frequencies (harmonic and intermod terms).

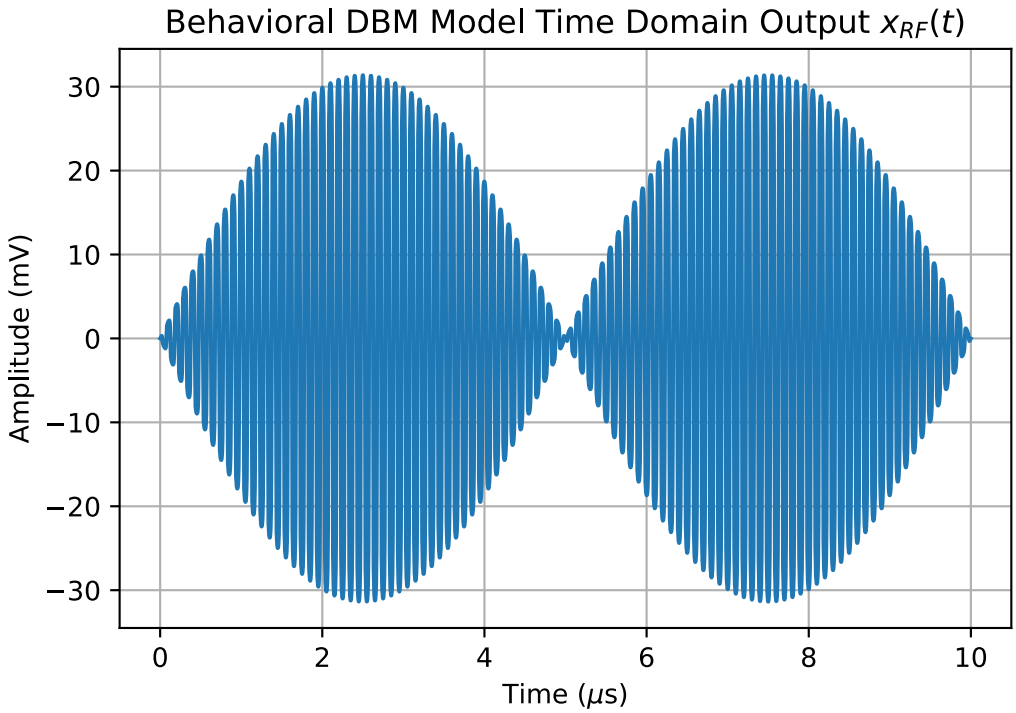
```
def DBM_model(x_LO,x_IF_RF,conv_loss_dB = 5.0, alpha = 4.0, ):
    """

    Mark Wickert February 2019
    """
    x_LO_clip = 10**(-conv_loss_dB/20)*arctan(x_LO*alpha)*2/pi
    x_IF_RF_clip = 10**(-conv_loss_dB/20)*arctan(x_IF_RF*alpha)*2/pi
    x_out = x_LO_clip*x_IF_RF_clip
    return x_out, x_LO_clip, x_IF_RF_clip
```

A Quick Simulation

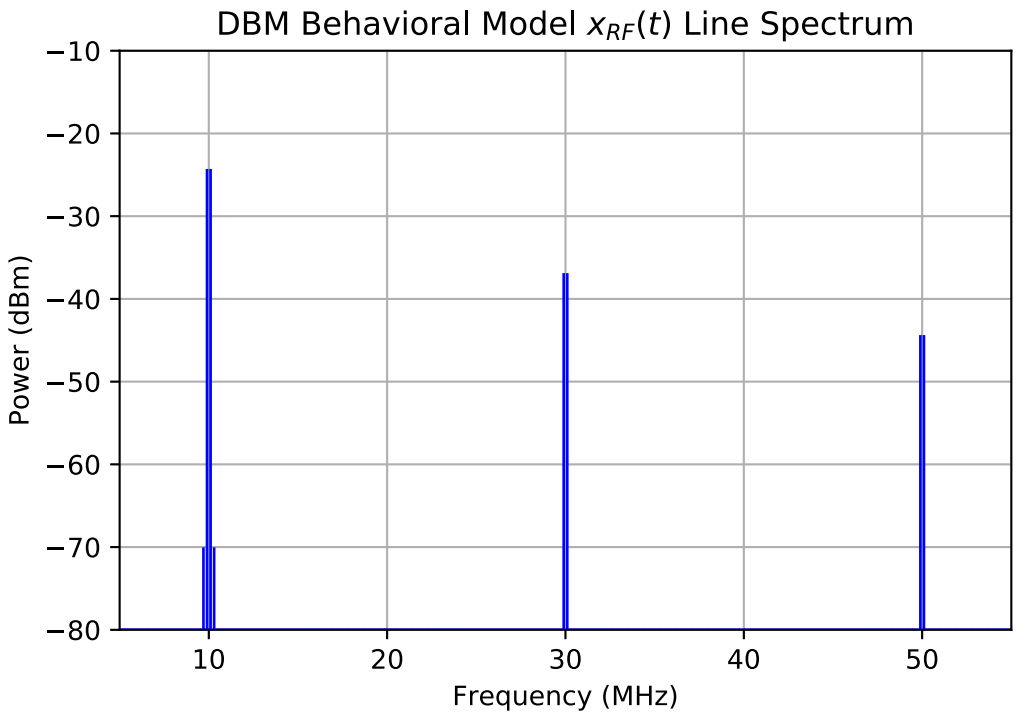
We will later calculate approximate Fourier series coefficients in order to plot the power spectrum. With $f_{\rm LO} = 10$ MHz and $f_{\rm IF} = 100$ kHz, we know that `x_RF` will be periodic with period of 1/100e3 = 10 μs. Thus in the following simulation we choose a sampling rate of 1 GHz for a high fidelity waveform and a simulation period of 10 μs.

```
t = arange(0,10e-6,1e-9)
x_LO = 1.416*cos(2*pi*10e6*t)
x_IF = 0.0632*sin(2*pi*100e3*t)
x_RF, x_LO_clip, x_IF_clip = DBM_model(x_LO, x_IF,conv_loss_dB=6.5)
```

```
plot(t*1e6,x_RF*1e3)
#plot(t*1e6,x_LO_clip*1e3)
#xlim([0,1])
title(r'Behavioral DBM Model Time Domain Output $x_{RF}(t)$')
ylabel(r'Amplitude (mV)')
xlabel(r'Time ($\mu$s)')
grid();
```




```
Xk, fk = ss.fs_coeff(x_RF,550,100e3)
Xn_dBm = line_spectra_dBm(fk/1e6,Xk,-80,lwidth=1)
title(r'DBM Behavioral Model $x_{RF}(t)$ Line Spectrum')
xlabel(r'Frequency (MHz)');
ylim([-80,-10])
xlim([5,55]);
```




```
for k, Xn_dBm_k in enumerate(Xn_dBm):
    if Xn_dBm_k > -50: # Threshold = -30 dBm
        print('Peak at %6.4f MHz of height %6.4f dBm'% (fk[k]/1e6,Xn_dBm[k]))
```

```
Peak at 9.9000 MHz of height -24.4505 dBm
Peak at 10.1000 MHz of height -24.4505 dBm
Peak at 29.9000 MHz of height -37.0442 dBm
Peak at 30.1000 MHz of height -37.0442 dBm
Peak at 49.9000 MHz of height -44.5326 dBm
Peak at 50.1000 MHz of height -44.5326 dBm
```

- **Measurement Example at $f_{LO} = 10$ MHz and $f_{IF} = 100$ kHz**

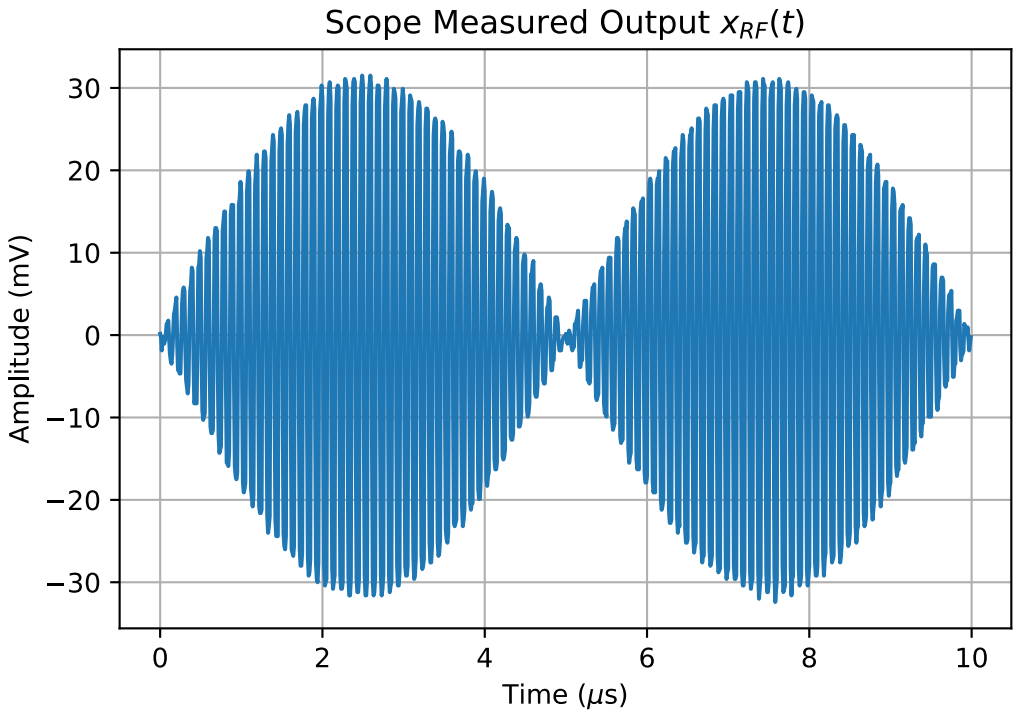
```
# Skip the first 32 rows, then skip the last row that contains 'END'
t_scope, scope_ch1, scope_ch2 = loadtxt('scope_dbm_mod.csv',delimiter=',',skiprows=2,
                usecols=(0,1,2),unpack=True)
```

```
def my_find(condition):
    """
    Find the indices where condition is true in an ndarray

    Mark Wickert February 2019
    """
    idx = np.nonzero(np.ravel(condition))[0]
    return idx
```

```
(my_find(t_scope == 0), my_find(t_scope == 10e-6))
```

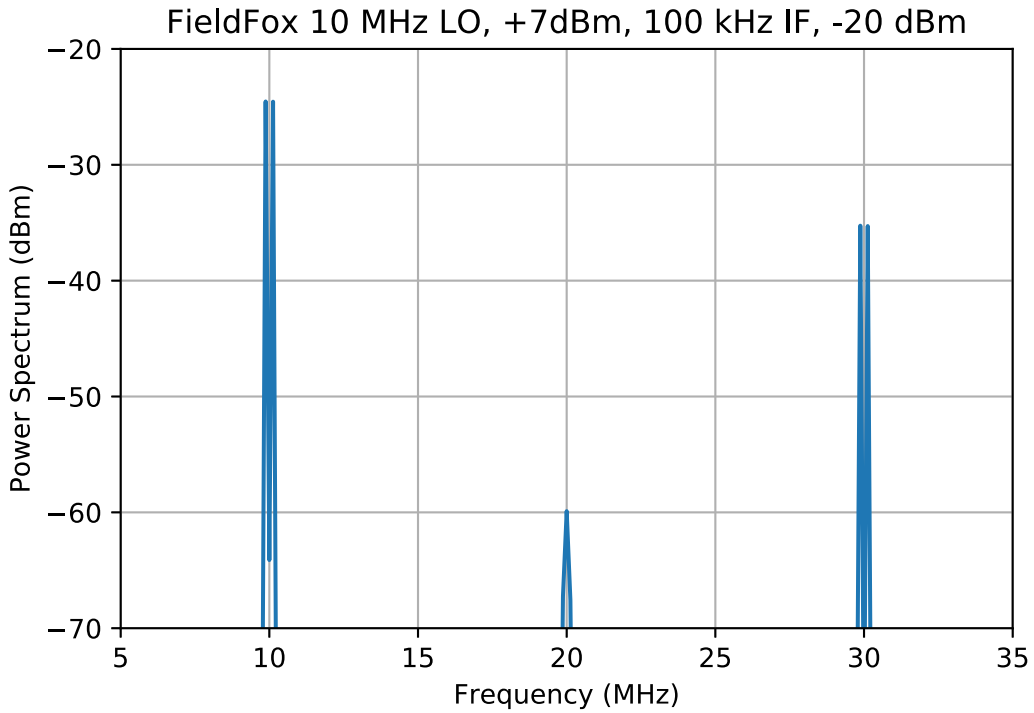
```
(array([902]), array([1902, 1903, 1904, 1905, 1906]))
```

```
plot(t_scope[902:1902]*1e6,scope_ch1[902:1902]*1e3)
title(r'Scope Measured Output $x_{RF}(t)$')
ylabel(r'Amplitude (mV)')
xlabel(r'Time ($\mu$s)')
grid();
```



```
# Skip the first 32 rows, then skip the last row that contains 'END'
f_SA, Sx_DBM_mod_10M_100k = loadtxt('DBM_MOD_10M_100K.csv',delimiter=',',skiprows=32,
                usecols=(0,1),comments='END',unpack=True)
```

```
plot(f_SA/1e6,Sx_DBM_mod_10M_100k)

xlabel(r'Frequency (MHz)')
ylabel(r'Power Spectrum (dBm)')
title(r'FieldFox 10 MHz LO, +7dBm, 100 kHz IF, -20 dBm')
ylim([-70,-20])
xlim([5,35])
grid()
```




```
peak_idx = signal.find_peaks(Sx_DBM_mod_10M_100k,height=(-75,)) # peak threshold = -30 dBm
for k, k_idx in enumerate(peak_idx[0]):
    print('Peak at %6.4f MHz of height %6.4f dBm'%
(f_SA[k_idx]/1e6,Sx_DBM_mod_10M_100k[k_idx]))
```

```
Peak at 9.8750 MHz of height -24.5360 dBm
Peak at 10.1250 MHz of height -24.5541 dBm
Peak at 20.0000 MHz of height -59.8962 dBm
Peak at 29.8750 MHz of height -35.2565 dBm
Peak at 30.1250 MHz of height -35.2895 dBm
Peak at 39.8750 MHz of height -66.5790 dBm
Peak at 40.1250 MHz of height -66.2963 dBm
Peak at 49.8750 MHz of height -41.4309 dBm
Peak at 50.1250 MHz of height -41.4799 dBm
```

More accurate frequency location can be obtained via spectral zooming and/or a smaller resolution bandwidth.

Lab Tasks Work Area

Characterizing the Power Splitter Combiner

```
# Skip the first 13 rows assuming tab spacing between entries
f_s2p, S11magdB,S11deg,S21magdB,S21deg =
loadtxt('ADP_2_1_SUM2P1.s2p',delimiter='\t',skiprows=13,
                                            usecols=(0,1,2,3,4),unpack=True)
plot(f_s2p/1e6,S21magdB)
xlabel(r'Frequency (MHz)')
ylabel(r'$|S_{21}|^2$ (dB)')
title(r'Power Splitter Sum Port to Port 1 ($S_{21}$ dB)')
ylim([-6,0])
grid();
```

```
# Skip the first 13 rows assuming tab spacing between entries
f_s2p, S11magdB,S11deg,S21magdB,S21deg =
loadtxt('ADP_2_1_ISO_P1_P2.s2p',delimiter='\t',skiprows=13,
                                            usecols=(0,1,2,3,4),unpack=True)
plot(f_s2p/1e6,S21magdB)
xlabel(r'Frequency (MHz)')
ylabel(r'$|S_{21}|^2$ (dB)')
title(r'Power Splitter Port 1 to Port Isolation ($S_{21}$ dB)')
ylim([-40,0])
grid();
```

Mixer Basics

Double Sideband Modulation and Demodulation

Part d

```
fs = 100e6 # sampling rate
fc = 1e6 # cutoff frequency
b, a = signal.cheby1(5, 1, 2*fc/fs) # digital filter design
```

```
import sk_dsp_comm.iir_design_helper as iir_h
```

```
iir_h.freqz_resp_list([b],[a],'dB',fs/1e3)
ylim([-80,0])

grid();
```

```
fc = 5e6
fm = 100e3
t = arange(0,2/(100e3),1/100e6) # Two cycles of 100 kHz with fs = 100 MHz
# Create the modulated DSB
x_LO = 1.416*cos(2*pi*fc*t)
x_IF = 0.0632*sin(2*pi*fm*t)
x_RF, x_LO_clip, x_IF_clip = DBM_model(x_LO, x_IF,conv_loss_dB=5.0)
# Create the coherently demodulated DSB
# Write code here
...
x_demod = ....
# Filter with 1 MHz LPF
y_demod = signal.lfilter(...)
plot(t/1e3,y_demod*1e3) # t in ms and V in mV
```

AM Modulation and Demodulation

Part c

Bring scope and LTSpice results together.