Lab 1 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
```

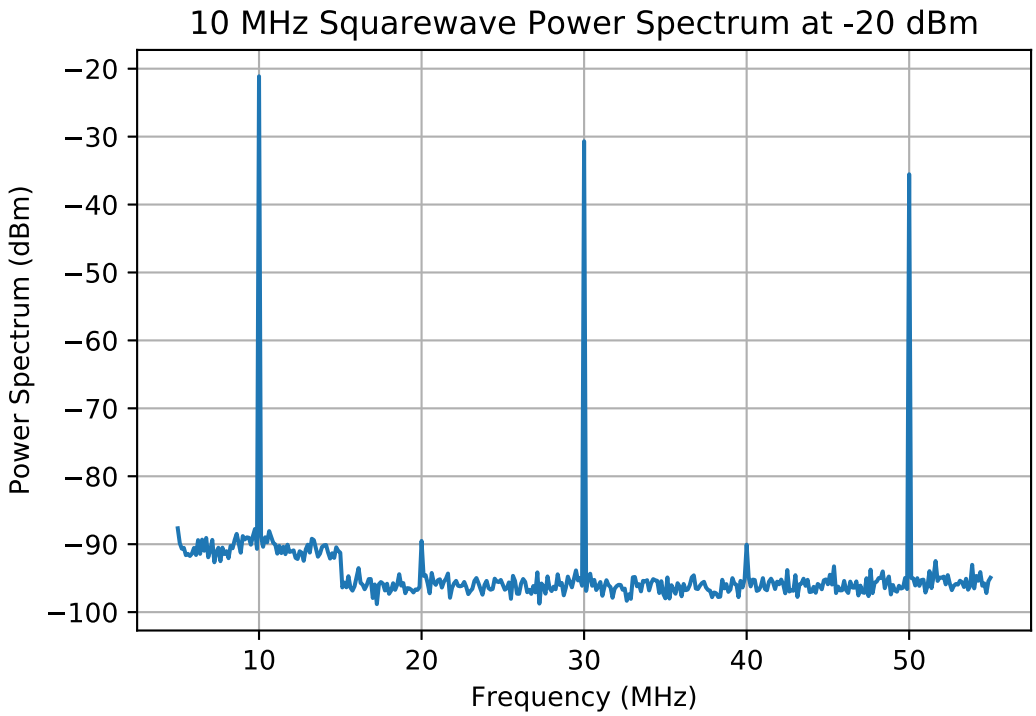
```
pylab.rcParams['savefig.dpi'] = 100 # default 72
#pylab.rcParams['figure.figsize'] = (6.0, 4.0) # default (6,4)
#%config InlineBackend.figure_formats=['png'] # default for inline viewing
%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
#%config InlineBackend.figure_formats=['pdf']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

Importing FieldFox CSV Files

```
# Skip the first 32 rows, then skip the last row that contains 'END'
f, Sx = loadtxt('10MHz_sq_-20dBm.csv',delimiter=',',skiprows=32,
                usecols=(0,1),comments='END',unpack=True)
```

```
plot(f/1e6,Sx)

xlabel(r'Frequency (MHz)')
ylabel(r'Power Spectrum (dBm)')
title(r'10 MHz Squarewave Power Spectrum at -20 dBm')
grid()
```



Search for Peaks Using `scipy.signal.find_peaks`

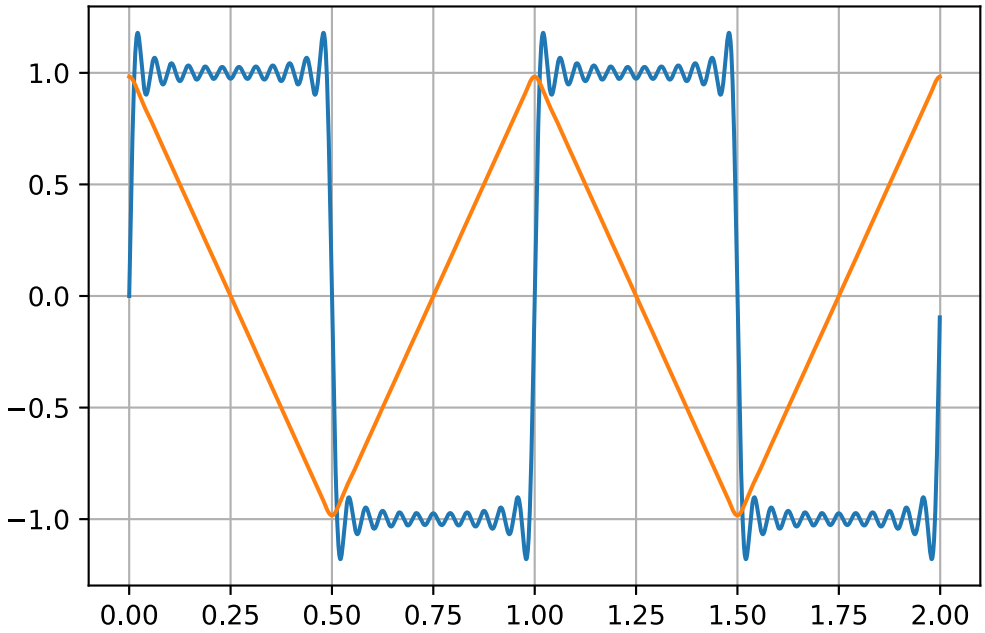
```
peak_idx = signal.find_peaks(Sx,height=(-60,)) # peak threshold = -60 dBm
for k, k_idx in enumerate(peak_idx[0]):
    print('Peak at %6.4f MHz of height %6.4f dBm'% (f[k_idx]/1e6,Sx[k_idx]))
```

```
Peak at 10.0000 MHz of height -21.1254 dBm
Peak at 30.0000 MHz of height -30.6877 dBm
Peak at 50.0000 MHz of height -35.5452 dBm
```

Square and Triangle Wave Expansions

```
A = 1
f0 = 1
Nmax = 25
n = arange(1,Nmax,2)
fn = n*f0
Xn = 2*A/(1j*pi*n)
Yn = 4*A/((n*pi)**2)
t = arange(0,2,.001)
x = ss.fs_approx(Xn,fn,t)
y = ss.fs_approx(Yn,fn,t)
plot(t,x)
plot(t,y)

grid();
```



Filter Design by the Insertion Loss Method

From circuits and systems we know that a linear system having input and output will have a *system function* that relates the input to the output. For circuit synthesis using lumped L and C elements in particular, the system function will a rational is s

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_0 + b_1 s + \cdot + b_N s^N}{a_0 + a_1 s + \cdot + a_N s^N} \tag{1}$$

where N is the filter order.

Amplitude and phase response can be determined directly from $H(s)$ with $s \to j2\pi f$. To obtain the group delay we need a function for numerically computing group delay from phase:

```
def grp_delay_s(H,f):
    """
    Group delay from frequency response

    Since this calculation involves finding the derivative of the
    phase response, care must be taken at phase wrapping points
    and when the phase jumps by +/-pi, which occurs when the
    amplitude response changes sign. Since the amplitude response
    is zero when the sign changes, the jumps do not alter the group
    delay results.

    Mark Wickert February 2019, based on portions of sk_dsp_comm.iir_design_helper
    """
    w = 2*pi*f
    theta = np.unwrap(np.angle(H))
    # Since theta for an FIR filter is likely to have many pi phase
    # jumps too, we unwrap a second time 2*theta and divide by 2
    theta2 = np.unwrap(2*theta)/2.
```

```
    #theta_dif = np.diff(theta2)
    #f_diff = np.diff(f)
    Tg = -np.diff(theta2)/np.diff(w)
    # For gain almost zero set groupdelay = 0
    idx = np.nonzero(np.ravel(20*np.log10(H[:-1]) < -400))[0]
    Tg[idx] = np.zeros(len(idx))
    max_Tg = np.max(Tg)
    #print(max_Tg)
    return Tg
```

RC Lowpass Group Delay Verify

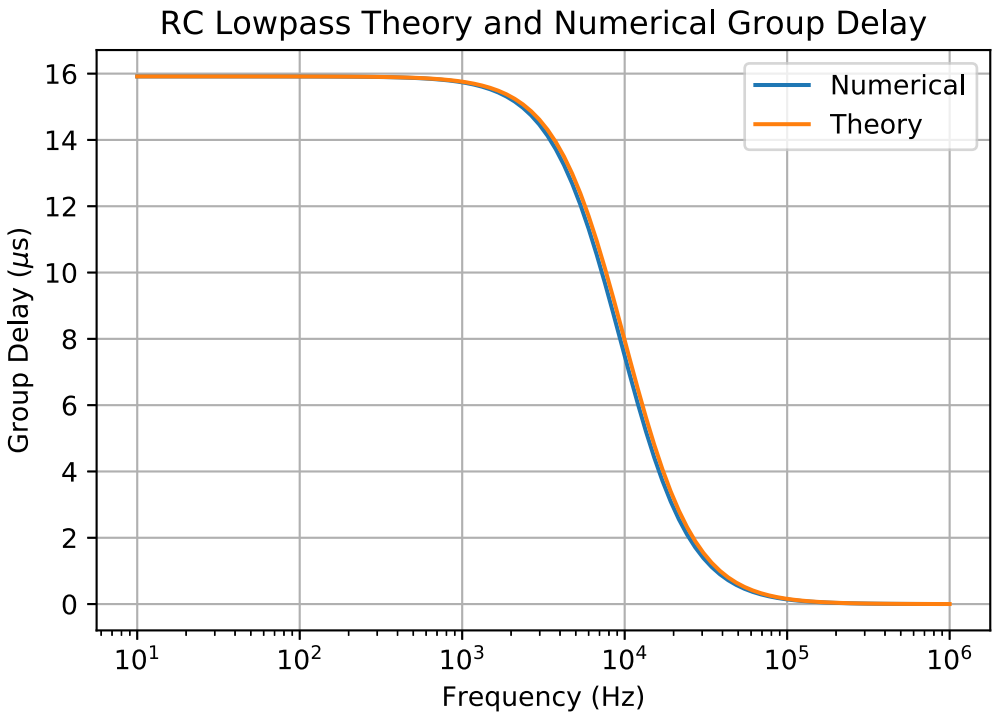
To better understand group delay, consider a simple test case to validate the numerical evaluation performed by `grp_delay_s` and theory. An RC lowpass filter embedded in a zero ohm source resistance and infinite resistance load, has system function

$$H(f) = \frac{1}{1 + j2\pi fRC} = \frac{1}{1 + jf/f_c} \tag{2}$$

where f_c is the filter 3dB frequency. So

$$T_g = \frac{-1}{2\pi} \cdot \frac{d}{df} - \arctan(f/f_c) = \frac{1}{2\pi} \cdot \frac{f_c}{f^2 + f_c^2} \text{ (s)} \tag{3}$$

```
f_RC = logspace(1,6,100)
fc = 1e4
b_RC = array([2*pi*fc])
a_RC = array([1, 2*pi*fc])
w, H_RC = signal.freqs(b_RC,a_RC,2*pi*f_RC)
#semilogx(f_RC,20*log10(abs(H_RC)))
T_g_RC = grp_delay_s(H_RC,f_RC)
T_g_RC_thy = fc/(2*pi)*1/(f_RC**2 + fc**2)
semilogx(f_RC[:-1],T_g_RC*1e6)
semilogx(f_RC,T_g_RC_thy*1e6)
title(r'RC Lowpass Theory and Numerical Group Delay')
ylabel(r'Group Delay ($\mu$s)')
xlabel(r'Frequency (Hz)')
legend((r'Numerical',r'Theory'))
grid();
```

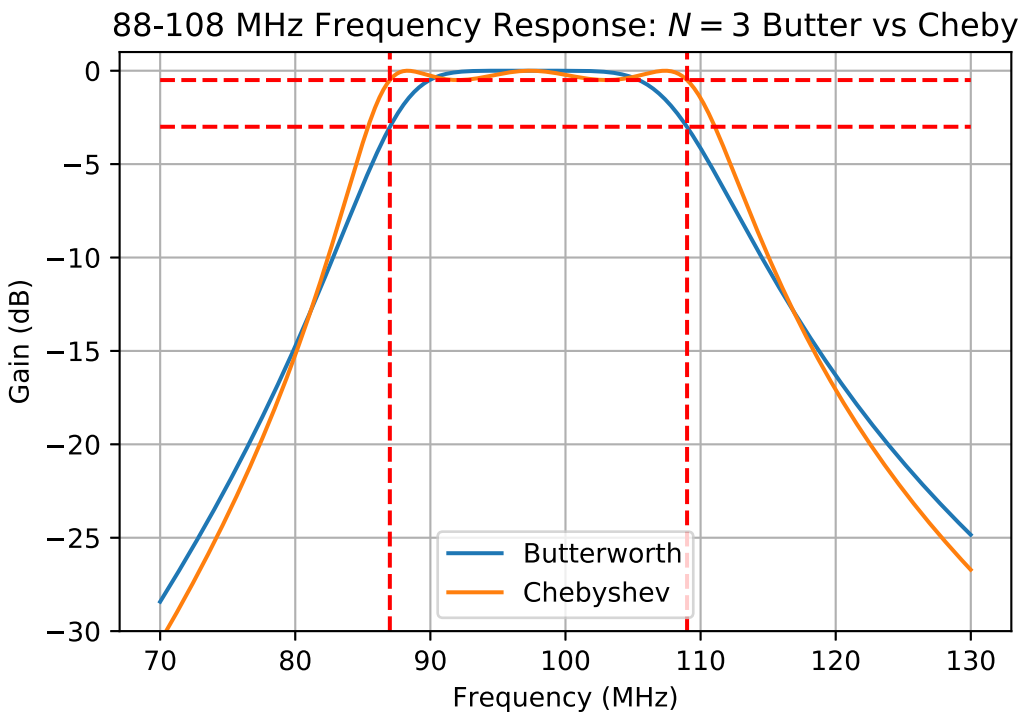


- ## Example: 88-108 MHz $N = 3$ Bandpass

```
# b, a = signal.butter(N, Wn, btype='low', analog=False, output='ba'), N is the number of
lowpass poles
b_but, a_but = signal.butter(3, 2*pi*array([87e6,109e6]), btype='bandpass', analog=True,
output='ba')
# b_cheb, a_cheb =  signal.cheby1(N, rp, Wn, btype='low', analog=False, output='ba')
b_cheb, a_cheb = signal.cheby1(3, 0.5, 2*pi*array([87e6,109e6]), btype='bandpass',
analog=True, output='ba')
print(b_cheb)
print(a_cheb)
```

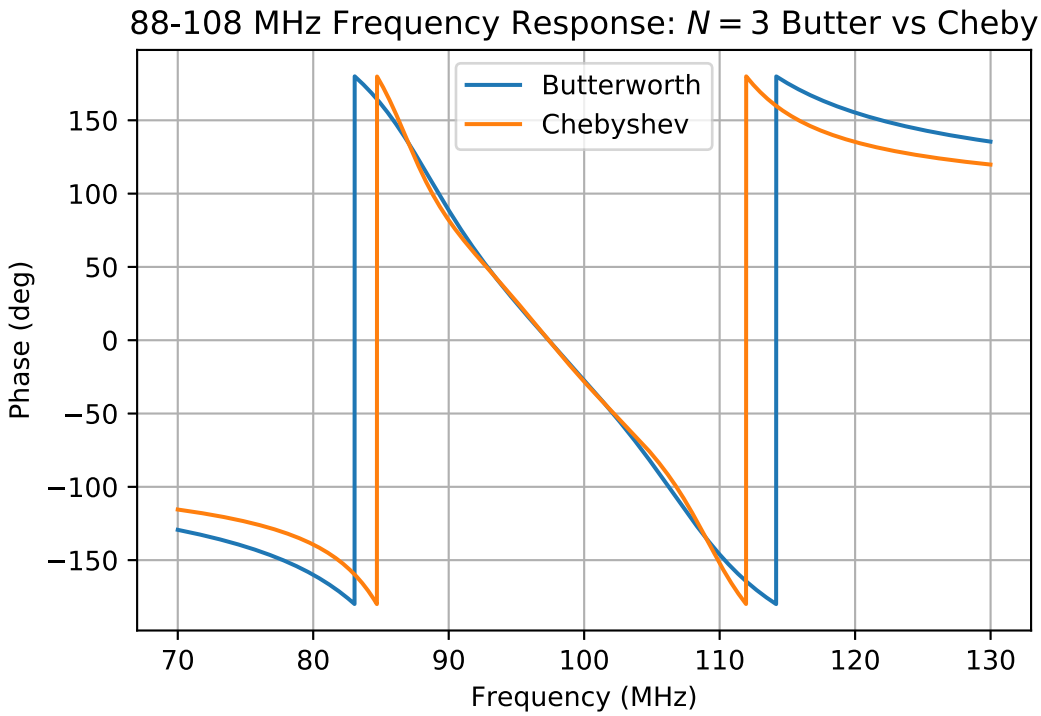
```
[1.89031812e+24 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[1.00000000e+00 1.73190256e+08 1.15244960e+18 1.31566119e+26
 4.31446976e+35 2.42736133e+43 5.24706521e+52]
```

```
f = arange(70, 130, .01) # In MHz
w, H_but = signal.freqs(b_but,a_but,2*pi*f*1e6)
w, H_cheb = signal.freqs(b_cheb,a_cheb,2*pi*f*1e6)
plot(f,20*log10(abs(H_but)))
plot(f,20*log10(abs(H_cheb)))
plot([70,130],[-.5,-.5],'r--'); plot([70,130],[-3,-3],'r--');
plot([87,87],[-30,1],'r--'); plot([109,109],[-30,1],'r--');
ylim([-30,1])
title(r'88-108 MHz Frequency Response: $N=3$ Butter vs Cheby')
ylabel(r'Gain (dB)')
xlabel(r'Frequency (MHz)')
legend((r'Butterworth',r'Chebyshev'))
grid();
```

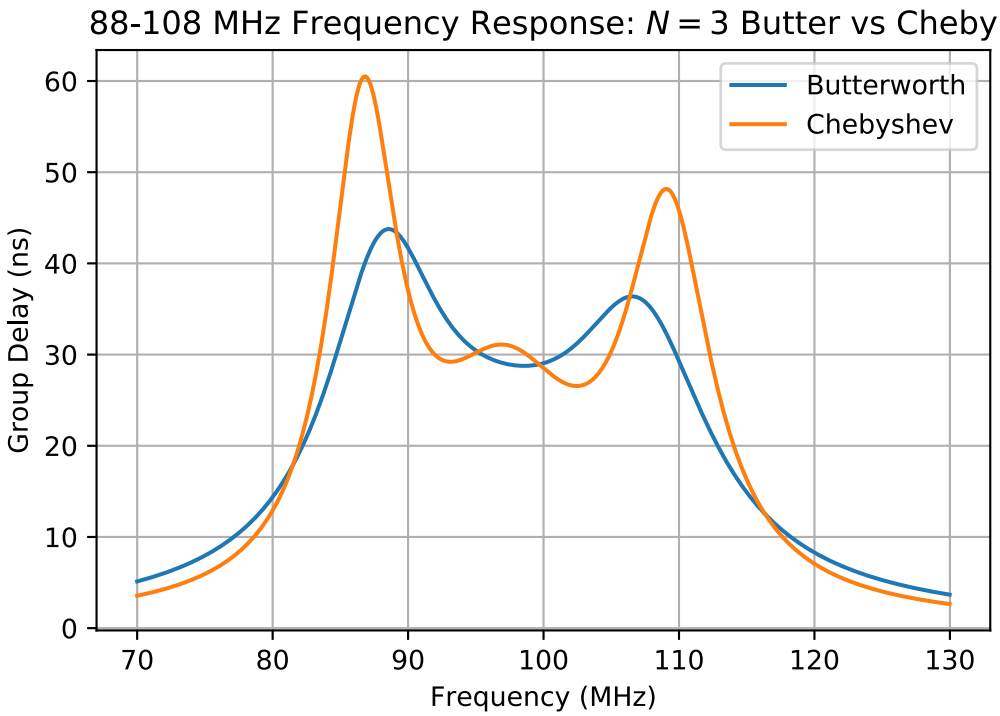


In the above the dashed lines mark the critical frequencies and pass band attenuation levels at the upper and lower band edges; Butterworth -3dB and Chebyshev -0.5 dB.

```
plot(f,angle(H_but)*180/pi)
plot(f,angle(H_cheb)*180/pi)
title(r'88-108 MHz Frequency Response: $N=3$ Butter vs Cheby')
ylabel(r'Phase (deg)')
xlabel(r'Frequency (MHz)')
legend((r'Butterworth',r'Chebyshev'))
grid();
```




```
Tg_but = grp_delay_s(H_but,f*1e6) # frequency in Hz
Tg_cheb = grp_delay_s(H_cheb,f*1e6)
plot(f[:-1],Tg_but*1e9) # frequency axis needs to be reduced by 1 index
plot(f[:-1],Tg_cheb*1e9)
title(r'88-108 MHz Frequency Response: $N=3$ Butter vs Cheby')
ylabel(r'Group Delay (ns)')
xlabel(r'Frequency (MHz)')
legend((r'Butterworth',r'Chebyshev'))
grid();
```



Frequency Response from Network Analyzer `s2p` Files

The code cells below contain code for designing and then analyzing filters with component tolerances and finite inductor Q. Code cells for bandpass filters are first, with lowpass designs following.

```
def lc_butter_bpf(N,f1,f2,R):
    """
    Rg,L,C,RL = lc_butter_bpf(N,f1,f2,R): lumped element
    bandpass filter design using a maximally flat or Butterworth
    lowpass prototype. The LC ladder network is assumed to begin
    with a parallel LC resonator circuit followed by a series LC resonator,
    followed by a another parallel LC resonator circuit, etc.
    The filter is assumed to be doubly terminated, that is
    a resistive source impednace is required as well as
    a resistive load termination.

     N = lowpass filter order (bandpass design is of order 2N
    f1 = lower 3dB bandedge frequency in Hz
    f2 = upper 3dB bandedge frequency in Hz
     R = impedance scaling factor in ohms

    Rg = scaled source termination resistance in ohms
     L = vector of inductance values in Henries [parallel series parallel ...]
     C = vector of capacitance values in Farads [parallel series parallel ...]
    RL = scaled load resistance in ohms (same as Rg for Butterworth)

    The filter topology for N = 3 is the following:

         o---Rg---o----|---o----L2----o----C2-----o---|----o---    - +
                       |                              |            |
```

```
          +             |                               |             |
                    ----o----                       ----o----         |
                    |       |                       |       |         |
      Input          |       |                       |       |  cont.  |
      Source        L1      C1                      L3      C3   ...  RL  Output
                    |       |                       |       |         |
                    |       |                       |       |  N > 3  |
          -         ----o----                       ----o----         |
                        |                               |             |
          0-------------|-------------------------------|----o---     | -
                                      Ground
```

```
    Mark Wickert April 2001, translated to Python December 2018
    """
    Rg = R
    RL = R

    #Normalized (R=1) lowpass g values for Butterworth prototype
    k = arange(N)
    g = 2*sin((2*k+1)/(2*N)*pi)

    #Scale impedance level of all g values
    g[0::2] = g[0::2]/R # Capacitor scaling
    g[1::2] = g[1::2]*R # Inductor scaling

    #Lowpass to bandpass element transformation
    L = zeros(N)
    C = zeros(N)

    #Detect input type: f1 & f2 or f0 & w
    if f2 < 1:
        f0 = f1
        w = f2
        f1 = f0/2*(sqrt(4 + w**2) - w)
        f2 = f0/2*(sqrt(4 + w**2) + w)
        print('***** f1 = %10.3g Hz and f2 = %10.3g Hz *****' % (f1,f2))
    else:
        f0 = sqrt(f1*f2) # Center frequency in Hz
        w = (f2 - f1)/f0 # Fractional bandwidth
        print('*****  f0 = %10.3g Hz and w = %5.3f *****' % (f0,w))

    #C to parallel LC resonators
    L[0::2] = w/(2*pi*f0*g[0::2])
    C[0::2] = g[0::2]/(2*pi*f0*w)
    #L to series LC resonators
    L[1::2] = g[1::2]/(2*pi*f0*w)
    C[1::2] = w/(2*pi*f0*g[1::2])
    return Rg,L,C,RL


def lc_cheby_bpf(N,RdB,f1,f2,R):
    """
    Rg,L,C,RL = lc_cheby_bpf(N,RdB,f1,f2,R): lumped element
    bandpass filter design using a Chebyshev
```

```
lowpass prototype. The LC ladder network is assumed to begin
with a parallel LC resonator circuit followed by a series LC resonator,
followed by a another parallel LC resonator circuit, etc.
The filter is assumed to be doubly terminated, that is
a resistive source impednace is required as well as
a resistive load termination.

  N = lowpass filter order (bandpass design is of order 2N
RdB = passband ripple in dB
 f1 = lower 3dB bandedge frequency in Hz
 f2 = upper 3dB bandedge frequency in Hz
  R = impedance scaling factor in ohms

 Rg = scaled source termination resistance in ohms
  L = vector of inductance values in Henries [parallel series parallel ...]
  C = vector of capacitance values in Farads [parallel series parallel ...]
 RL = scaled load resistance in ohms (same as Rg only for N odd)

The filter topology for N = 3 is the following:

    o---Rg---o----|---o----L2----o----C2-----o---|----o---    - +
                  |                              |            |
    +             |                              |            |
               ----o----                       ----o----      |
               |       |                       |       |      |
 Input         |       |                       |       |  cont.  |
 Source        L1      C1                      L3      C3   ...  RL  Output
               |       |                       |       |      |
               |       |                       |       |  N > 3  |
    -          ----o----                       ----o----      |
                   |                              |            |
    0--------------|------------------------------|----o---    | -
                                  Ground
Mark Wickert April 2001, translated to Python December 2018
"""
#Convert Ripple in dB (RdB) to eps2 = epsilon^2
eps2 = 10**(RdB/10) - 1
Rg = R

if int(N/2)*2 == N: # If N is even RL is not equal to R
    RL = R*(1 + 2*eps2 + 2*sqrt(eps2*(1 + eps2)))
else:
    RL = R

#Normalized (R=1) lowpass g values for Chebyshev prototype
k = arange(N)
beta = log((sqrt(1+eps2)+1)/(sqrt(1+eps2)-1));
a = sin((2*k+1)/(2*N)*pi)
b = (sinh(beta/(2*N)))**2 + (sin((k+1)*pi/N))**2
g = zeros(N)
g[0] = 2*a[0]/(sinh(beta/(2*N)))
for i in range(1,N):
    g[i] = 4*a[i-1]*a[i]/(b[i-1]*g[i-1])
```

```
    #Scale impedance level of all g values
    g[0::2] = g[0::2]/R # Capacitor scaling
    g[1::2] = g[1::2]*R # Inductor scaling

    #Lowpass to bandpass element transformation
    L = zeros(N)
    C = zeros(N)

    #Detect input type: f1 & f2 or f0 & w
    if f2 < 1:
        f0 = f1
        w = f2
        f1 = f0/2*(sqrt(4 + w**2) - w)
        f2 = f0/2*(sqrt(4 + w**2) + w)
        print('***** f1 = %10.3g Hz and f2 = %10.3g Hz *****' % (f1,f2))
    else:
        f0 = sqrt(f1*f2) # Center frequency in Hz
        w = (f2 - f1)/f0 # Fractional bandwidth
        print('*****  f0 = %10.3g Hz and w = %5.3f *****' % (f0,w))

    #C to parallel LC resonators
    L[0::2] = w/(2*pi*f0*g[0::2])
    C[0::2] = g[0::2]/(2*pi*f0*w)
    #L to series LC resonators
    L[1::2] = g[1::2]/(2*pi*f0*w)
    C[1::2] = w/(2*pi*f0*g[1::2])
    return Rg,L,C,RL


def lc6_Ymodel_bpf(f,L,C,f0,Q=[1000,1000,1000],R0=50):
    """
    Full loss LC Bandpass modeling using an admittance ladder network

        The filter topology is the following:

         o---R0---o-----|-----o----Y2-----o-----|---------o---- +
                        |                      |         |
         +              |                      |         |
                        o                      o         |
                        |                      |         |
      Input             |                      |         |
      Source           Y1                     Y3        R0  Output
                        |                      |         |
                        |                      |         |
         -              o                      o         |
                        |                      |         |
         0--------------|----------------------|---------o----- -
                                    Ground

     f = frequency ndarray
     L = Inductance ndarray
     C = Capacitance ndarray
```

```
    f0 = Center frequency or cutoff frequency for finite Q losses
     Q = quality factor of inductors
    R0 = characteristic impedance, e.g., 50 Ohms

    Mark Wickert January 2019
    """
    # Rs and Rp from inductor Q and center frequency
    Rp1 = 2*pi*f0*L[0]*Q[0]
    Rs2 = 2*pi*f0*L[1]/Q[1]
    Rp3 = 2*pi*f0*L[2]*Q[2]
    # Define s along the jw axis
    s = 1j*2*pi*f
    G0 = 1/R0
    # Admittance values
    Y1 = s*C[0] + 1/(s*L[0]) + 1/Rp1
    Y2 = 1/(1/(s*C[1]) + s*L[1] + Rs2)
    Y3 = s*C[2] + 1/(s*L[2]) + 1/Rp3
    # Calculate H(s)
    H = 2*G0*Y2/((G0+Y1+Y2)*(G0+Y2+Y3)- Y2**2)
    return H
```

```
def lc_butter_lpf(N,fc,R):
    """
    Rg,L,C,RL = lc_butter_lpf(N,fc,R): lumped element
    lowpass filter design using a maximally flat or Butterworth
    lowpass prototype. The LC ladder network is assumed to begin
    with a parallel C followed by a series L,
    followed by a another parallel C, etc.
    The filter is assumed to be doubly terminated, that is
    a resistive source impednace is required as well as
    a resistive load termination.

     N = lowpass filter order (bandpass design is of order 2N
    fc = 3dB cutoff frequency in Hz
     R = impedance scaling factor in ohms

    Rg = scaled source termination resistance in ohms
     L = vector of inductance values in Henries [series ...]
     C = vector of capacitance values in Farads [parallel ...]
    RL = scaled load resistance in ohms (same as Rg for Butterworth)

    The filter topology for N = 3 is the following:

         o---Rg---o----|---o----L2----o---|----o--------o +
                       |                  |             |
         +             |                  |             |
                       o                  o             |
                       |                  |             |
      Input            |                  |   cont.     |  Load
      Source           C1                 C3            RL
                       |                  |             |
                       |                  |             |
```

```
          -             o                   o             |
                        |                   |             |
          0-------------|-------------------|----o--------o -
                                          Ground
```

```
    Mark Wickert December 2018
    """
    Rg = R
    RL = R

    #Normalized (R=1) lowpass g values for Butterworth prototype
    k = arange(N)
    g = 2*sin((2*k+1)/(2*N)*pi)

    #Scale impedance level of all g values
    g[0::2] = g[0::2]/R # Capacitor scaling
    g[1::2] = g[1::2]*R # Inductor scaling

    #Unit cutoff Lowpass to Lowpass element transformation
    C = g[0::2]/(2*pi*fc)
    L = g[1::2]/(2*pi*fc)
    return Rg,L,C,RL


def lc_cheby_lpf(N,RdB,fc,R):
    """
    Rg,L,C,RL = lc_cheby_lpf(N,RdB,fc,R): lumped element
    lowpass filter design using a Chebyshev
    lowpass prototype. The LC ladder network is assumed to begin
    with a parallel Cr followed by a series L, etc.
    The filter is assumed to be doubly terminated, that is
    a resistive source impednace is required as well as
    a resistive load termination.


      N = lowpass filter order (bandpass design is of order 2N
    RdB = passband ripple in dB
     fc = 3dB cutoff frequency in Hz
      R = impedance scaling factor in ohms

    Rg = scaled source termination resistance in ohms
     L = vector of inductance values in Henries [series ...]
     C = vector of capacitance values in Farads [parallel ...]
    RL = scaled load resistance in ohms (same as Rg for Butterworth)

    The filter topology for N = 3 is the following:
```

```
          o---Rg---o----|---o----L2----o---|----o--------o +
                        |                  |             |
          +             |                  |             |
                        o                  o             |
                        |                  |             |
      Input             |                  |   cont.     |  Load
      Source            C1                 C3            RL
```

```
                        |                   |             |
                        |                   |             |
             -          o                   o             |
                        |                   |             |
             0----------|-------------------|----o--------o -
                                          Ground
    Mark Wickert December 2018
    """
    #Convert Ripple in dB (RdB) to eps2 = epsilon^2
    eps2 = 10**(RdB/10) - 1
    Rg = R

    if int(N/2)*2 == N: # If N is even RL is not equal to R
        RL = R*(1 + 2*eps2 + 2*sqrt(eps2*(1 + eps2)))
    else:
        RL = R

    #Normalized (R=1) lowpass g values for Chebyshev prototype
    k = arange(N)
    beta = log((sqrt(1+eps2)+1)/(sqrt(1+eps2)-1));
    a = sin((2*k+1)/(2*N)*pi)
    b = (sinh(beta/(2*N)))**2 + (sin((k+1)*pi/N))**2
    g = zeros(N)
    g[0] = 2*a[0]/(sinh(beta/(2*N)))
    for i in range(1,N):
        g[i] = 4*a[i-1]*a[i]/(b[i-1]*g[i-1])

    #Scale impedance level of all g values
    g[0::2] = g[0::2]/R # Capacitor scaling
    g[1::2] = g[1::2]*R # Inductor scaling

    #Unit cutoff Lowpass to Lowpass element transformation
    C = g[0::2]/(2*pi*fc)
    L = g[1::2]/(2*pi*fc)
    return Rg,L,C,RL


def lc5_Ymodel_lpf(f,L,C,f0,Q=[1000,1000],R0=50):
    """
    Full loss LC Lowpass modeling using an admittance ladder network

    The filter topology is the following:

         o---R0---o-|--o----Y2----o--|--o----Y4----o--|----------|  +
                    |               |               |          |
         +          |               |               |          |
                    o               o               o          |
                    |               |               |          |
      Input         |               |               |          |
      Source        Y1              Y3              Y5         R0  Output
                    |               |               |          |
                    |               |               |          |
         -          o               o               o          |
```

```
                  |                |                |          |
        0---------|----------------|----------------|--------- |  -
                                  Ground

     f = frequency ndarray
     L = Inductance ndarray
     C = Capacitance ndarray
    f0 = Center frequency or cutoff frequency for finite Q losses
     Q = quality factor of inductors
    R0 = characteristic impedance, e.g., 50 Ohms

    Mark Wickert January 2019
    """
    # Rs and Rp from inductor Q and cutoff frequency
    Rs2 = 2*pi*f0*L[0]/Q[0]
    Rs4 = 2*pi*f0*L[1]/Q[1]

    # Define s along the jw axis
    s = 1j*2*pi*f
    G0 = 1/R0
    # Admittance values
    Y1 = s*C[0]
    Y2 = 1/(s*L[0] + Rs2)
    Y3 = s*C[1]
    Y4 = 1/(s*L[1] + Rs4)
    Y5 = s*C[2]
    # Calculate H(s)
    Num = 2*G0*Y2*Y4
    Den = Y1*(Y2*(Y4+Y5+G0) + Y3*(Y4+Y5+G0) + Y4*(Y5+G0)) \
        + Y2*(Y3*(Y4+Y5+G0) + Y4*(Y5+2*G0) + G0*(Y5+G0)) \
        + G0*(Y3*(Y4+Y5+G0) + Y4*(Y5+G0))
    H = Num/Den
    return H
```

The RF Board Filters

- Two bandpass designs
- One lowpass design

The design requirements are summaried in the table below:

Filter Type	Description
70 MHz BPF	$N = 3$ lowpass prototype, with 0.5 dB passband ripple and 0.5 dB bandwidth of 10 MHz centered on 70 MHz. The fractional bandwidth is thus $w = \mathrm{BW}/f_0 = 10/70 = 14.28\%$, where $\mathrm{BW} = f_2 - f_1$ and $f_0 = \sqrt{f_1 f_2}$.
88-108 MHz BPF	$N = 3$ lowpass prototype, with 0.5 dB passband ripple, lower band edge of 87 MHz, upper band edge of 109 MHz. The fractional bandwidth is thus $w = (109 - 87)/\sqrt{87 \times 109} = 22.6$ %.
1 MHz LPF	$N = 5$ lowpass prototype, with 0.5 dB passband ripple and cutoff frequency of 10 MHz.

70 MHz Bandpass

Lumped Network Synthesis

```
f0 = 70e6 # Hz
BW = 10e6 # Hz
FB = BW/f0 # fractional bandwidth
Rg,L_BPF70,C_BPF70,RL = lc_cheby_bpf(3,0.5,f0,FB,50)
```

```
***** f1 =   6.52e+07 Hz and f2 =   7.52e+07 Hz *****
```

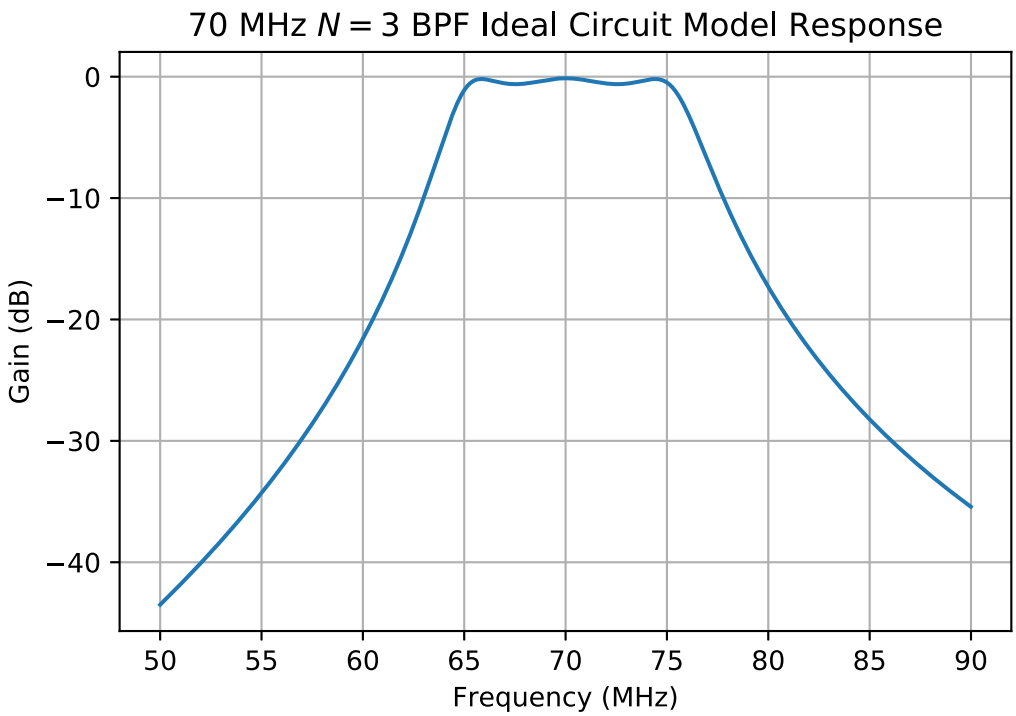
```
L_BPF70*1e9 # nH
```

```
array([ 10.17384147, 872.71954662,  10.17384147])
```

```
C_BPF70*1e12 # pF
```

```
array([508.11172543,   5.92337844, 508.11172543])
```

```
f = arange(50,90,.01)*1e6
H_BPF70 = lc6_Ymodel_bpf(f,L_BPF70,C_BPF70,f0,Q=[1000,1000,1000])
plot(f/1e6,20*log10(abs(H_BPF70)))
#semilogx(f,20*log10(abs(H)))
ylabel(r'Gain (dB)')
xlabel(r'Frequency (MHz)')
title(r'70 MHz $N=3$ BPF Ideal Circuit Model Response')
grid();
```



FM Broadcasting Receiver 88-108 MHz Bandpass Filter

```
f1 = 87e6
f2 = 109e6
Rg,L_FM,C_FM,RL = lc_cheby_bpf(3,0.5,f1,f2,50)
```

```
*****  f0 =   9.74e+07 Hz and w = 0.226 *****
```

```
(109-87)/sqrt(87*109)*100
```

```
22.591746464819778
```

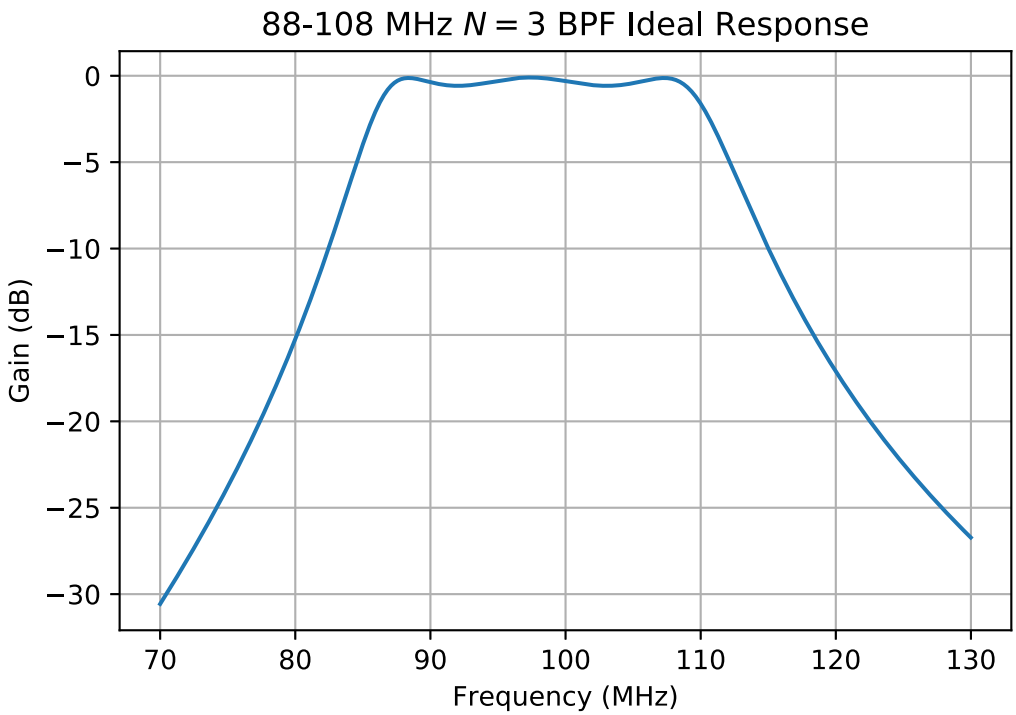
```
L_FM*1e9 # nH
```

```
array([ 11.5653286 , 396.69070301,  11.5653286 ])
```

```
C_FM*1e12 # pF
```

```
array([230.9598752 ,   6.73352521, 230.9598752 ])
```

```
f = arange(70,130,.01)*1e6
H_FM = lc6_Ymodel_bpf(f,L_FM,C_FM,f0,Q=[1000,1000,1000])
plot(f/1e6,20*log10(abs(H_FM)))
#semilogx(f,20*log10(abs(H)))
ylabel(r'Gain (dB)')
xlabel(r'Frequency (MHz)')
title(r'88-108 MHz $N=3$ BPF Ideal Response')
grid();
```



1 MHz Lowpass Filter

```
fc = 1e6
eps_dB = 0.5
Rg,L_LPF1,C_LPF1,RL = lc_cheby_lpf(5,eps_dB,fc,50)
```

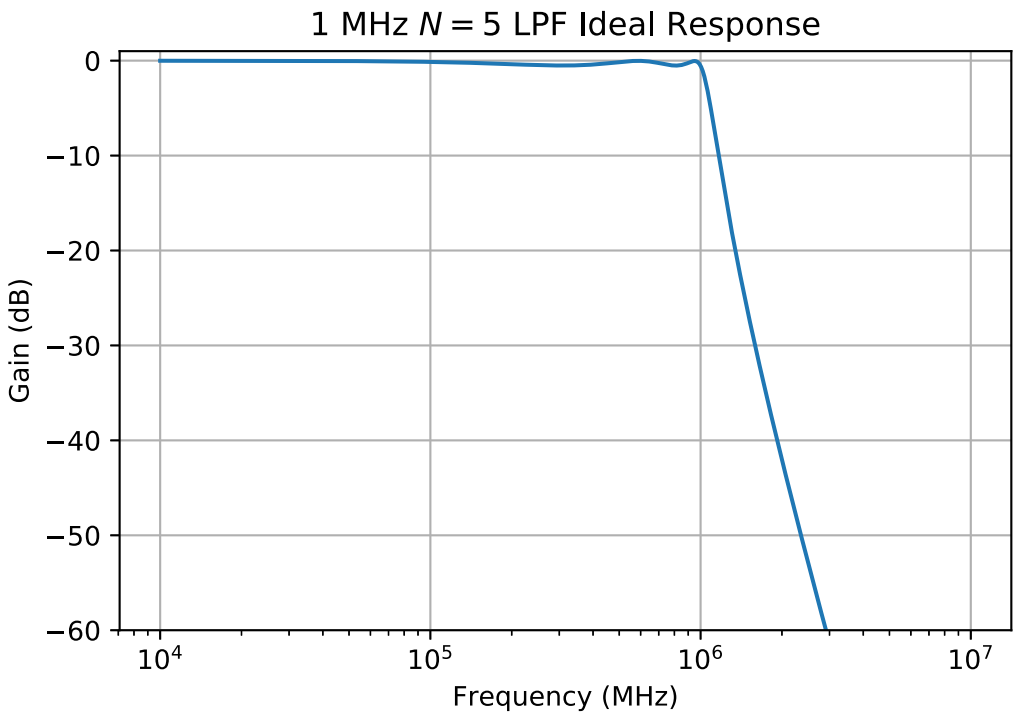
```
L_LPF1*1e6 # uH
```

```
array([9.78505867, 9.78505867])
```

```
C_LPF1*1e12 # pF
```

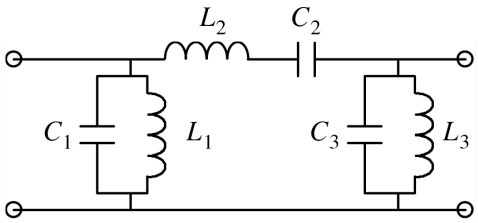
```
array([5429.63492592, 8087.70429138, 5429.63492592])
```

```
f = logspace(log10(10e3),log10(10e6),500)
H_LPF1 = lc5_Ymodel_lpf(f,L_LPF1,C_LPF1,fc,Q=[1000,1000,1000])
#plot(f/1e6,20*log10(abs(H_LPF1)))
semilogx(f,20*log10(abs(H_LPF1)))
ylim([-60,1])
ylabel(r'Gain (dB)')
xlabel(r'Frequency (MHz)')
title(r'1 MHz $N=5$ LPF Ideal Response')
grid();
```



Measurement Example Using a 30 MHz Center Frequency Bandpass Filter Design

We use the function defined in this notebook to 1.0 dB ripple Chebyshev bandpass with 1 dB passband running from 27.5 MHz to 32.5 MHz. The returned values from the function `lc_cheby_bpf(N_order,ripple_dB,f1,f2,R0)` is the source resistance `Rg`, an array of inductance values `L_BPF30`, an array of capacitance values, `C_BPF30`, and the load resistance, `RL`. Note with even order Chebyshev lowpass prototypes `RL` is not equal to `Rg`.




```
f1 = 27.5e6 # Hz
f2= 32.5e6 # Hz
# f1 = 25e6 # Hz
# f2= 35e6 # Hz
Rg,L_BPF30,C_BPF30,RL = lc_cheby_bpf(3,1.0,f1,f2,50)
```

```
*****  f0 =   2.99e+07 Hz and w = 0.167 *****
```

```
L_BPF30*1e9 # nH
```

```
array([  21.99991455, 1582.16317954,   21.99991455])
```

```
C_BPF30*1e12 # pF
```

```
array([1288.25908705,   17.91319012, 1288.25908705])
```

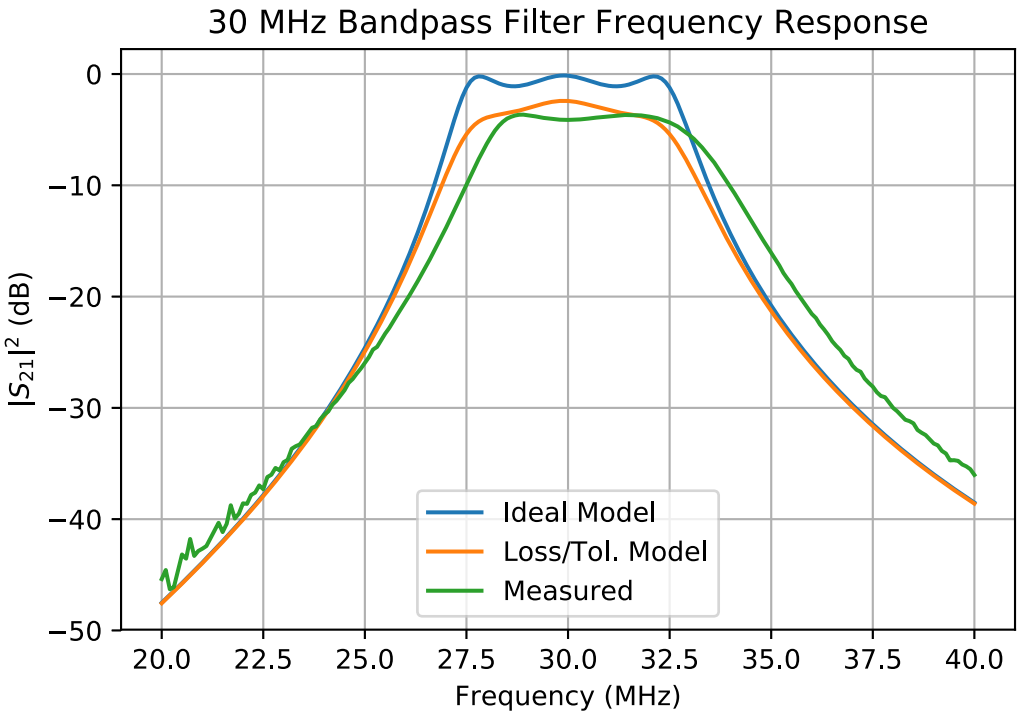
Import an `.s2p` File that Contains S_{21} and S_{11} Data

Here we import S-parameter data from the FieldFox and overlay plots of the ideal frequency response and the frequency response with the inductor Q reduced from 1000 (essentially an ideal inductor) to values of 50, 30, 50, for the three inductors L_1, L_2 and L_3.

```
# Skip the first 13 rows assuming tab spacing between entries
f, S11magdB,S11deg,S21magdB,S21deg =
loadtxt('30MHz_BPF_S2P.s2p',delimiter='\t',skiprows=13,
                                          usecols=(0,1,2,3,4),unpack=True)
IL_dB = -max(S21magdB)
print('Insertion Loss: %4.2f dB' % (IL_dB,))
H_ideal = lc6_Ymodel_bpf(f,L_BPF30,C_BPF30,sqrt(f1*f2),Q=[1000,1000,1000])
H_loss = lc6_Ymodel_bpf(f,L_BPF30,C_BPF30,sqrt(f1*f2),Q=[100,30,50])
plot(f/1e6,20*log10(abs(H_ideal)))
plot(f/1e6,20*log10(abs(H_loss)))
plot(f/1e6,S21magdB)
xlabel(r'Frequency (MHz)')
ylabel(r'$|S_{21}|^2$ (dB)')
title(r'30 MHz Bandpass Filter Frequency Response')
legend((r'Ideal Model',r'Loss/Tol. Model',r'Measured'))
```

```
grid();
```

```
Insertion Loss: 3.65 dB
```



Using Slider Widgets to Make the Tolerance/Loss Model Approximate Measured

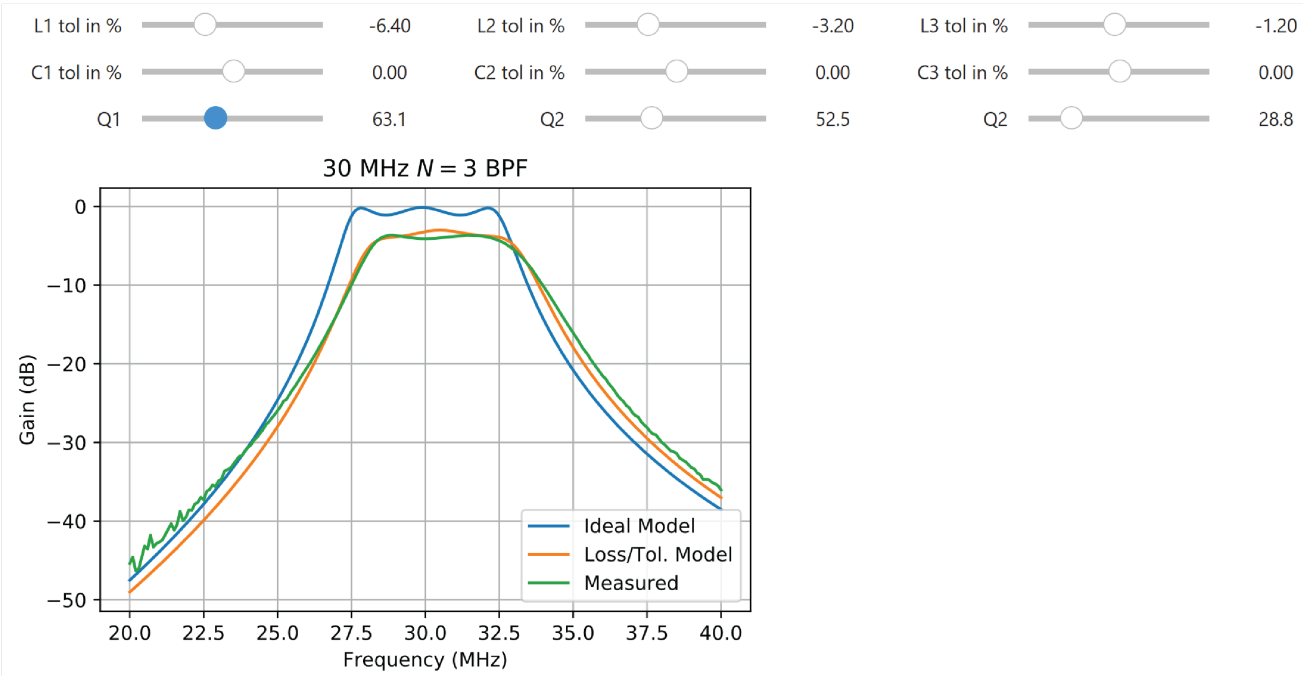
The objective here is to see if we can *tune* the element L and C away from the ideal and also adjust the inductor Q to bettwer match the measured results.

```
p_L1 = widgets.FloatSlider(value=0.,min=-20,max=20,step=0.1,description='L1 tol in %')
p_C1 = widgets.FloatSlider(value=0.,min=-5,max=5,step=0.1,description='C1 tol in %')
p_Q1 = widgets.FloatLogSlider(value=1000.,base=10,min=1,max=3,step=0.02,description='Q1')
p_L2 = widgets.FloatSlider(value=0.,min=-10,max=10,step=0.1,description='L2 tol in %')
p_C2 = widgets.FloatSlider(value=0.,min=-5,max=5,step=0.1,description='C2 tol in %')
p_Q2 = widgets.FloatLogSlider(value=1000.,base=10,min=1,max=3,step=0.02,description='Q2')
p_L3 = widgets.FloatSlider(value=0.,min=-20,max=20,step=0.1,description='L3 tol in %')
p_C3 = widgets.FloatSlider(value=0.,min=-5,max=5,step=0.1,description='C3 tol in %')
p_Q3 = widgets.FloatLogSlider(value=1000.,base=10,min=1,max=3,step=0.02,description='Q2')
uiV1 = widgets.VBox([p_L1,p_C1,p_Q1])
uiV2 = widgets.VBox([p_L2,p_C2,p_Q2])
uiV3 = widgets.VBox([p_L3,p_C3,p_Q3])
ui = widgets.HBox([uiV1,uiV2,uiV3])

def f_BPF(p_L1,p_C1,p_Q1,p_L2,p_C2,p_Q2,p_L3,p_C3,p_Q3):
    # Bring some variables into this function as globals to make the input arg
    # list shorter; other approaches work too, but this was an easy fix
    global L_BPF30, C_BPF30, f, S21magdB
```

```
    #f = arange(20,40,.01)*1e6
    # Tune the L and C values away from ideal and
    L_tol_err = array([(1+p_L1/100)*L_BPF30[0],(1+p_L2/100)*L_BPF30[1],
(1+p_L3/100)*L_BPF30[2]])
    C_tol_err = array([(1+p_C1/100)*C_BPF30[0],(1+p_C2/100)*C_BPF30[1],
(1+p_C3/100)*C_BPF30[2]])
    H_ideal = lc6_Ymodel_bpf(f,L_BPF30,C_BPF30,30e6,Q=[1000,1000,1000])
    H_loss = lc6_Ymodel_bpf(f,L_tol_err,C_tol_err,30e6,Q=[p_Q1,p_Q2,p_Q3])
    plot(f/1e6,20*log10(abs(H_ideal)))
    plot(f/1e6,20*log10(abs(H_loss)))
    plot(f/1e6,S21magdB)
    #semilogx(f,20*log10(abs(H)))
    ylabel(r'Gain (dB)')
    xlabel(r'Frequency (MHz)')
    title(r'30 MHz $N=3$ BPF')
    legend((r'Ideal Model',r'Loss/Tol. Model',r'Measured'))
    grid();

output = widgets.interactive_output(f_BPF,{'p_L1': p_L1, 'p_C1': p_C1, 'p_Q1': p_Q1,
                                           'p_L2': p_L2, 'p_C2': p_C2, 'p_Q2': p_Q2,
                                           'p_L3': p_L3, 'p_C3': p_C3, 'p_Q3': p_Q3})
display(ui, output)
```



- Create a tolerance array that multiplies the ideal L values to produce shifted values
- Use the values found using the GUI slider adjustments made above

```
# Tweak L values by -6.4%, -3.2%, and -1.2%
L_tol = array([1.0-0.064,1.0-0.032,1.0-0.012])
```

```
# Convert measured S_21 from dB & angle to complex freq resp.
```

```
H_meas = 10**(S21magdB/10)*exp(1j*S21deg*pi/180)
H_ideal = lc6_Ymodel_bpf(f,L_BPF30,C_BPF30,sqrt(f1*f2),Q=[1000,1000,1000])
Tg_H_ideal = grp_delay_s(H_ideal,f)
H_loss = lc6_Ymodel_bpf(f,L_BPF30*L_tol,C_BPF30,sqrt(f1*f2),Q=[100,30,50])
Tg_H_loss = grp_delay_s(H_loss,f)
Tg_H_meas = grp_delay_s(H_meas,f)
plot(f[:-1]/1e6,Tg_H_ideal*1e9)
plot(f[:-1]/1e6,Tg_H_loss*1e9)
plot(f[:-1]/1e6,Tg_H_meas*1e9)
ylim([0, 350])
xlim([23,38])
xlabel(r'Frequency (MHz)')
ylabel(r'Group delay (ns)')
title(r'30 MHz Bandpass Filter Frequency Response - $T_g(f)$')
legend((r'Ideal Model',r'Loss/Tol. Model',r'Measured'))
grid();
```