EE120 Fall 2014 Problem Set 10, Problem 5. version 1.1

In [341]:
```
%pylab
```

```
Using matplotlib backend: WXAgg
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['yscale']
`%pylab --no-import-all` prevents importing * from pylab and numpy
```

In [342]:
```
%matplotlib inline
import numpy as np
import scipy as sp
from scipy import signal
import matplotlib.pyplot as plt
np.set_printoptions(precision = 1, suppress = True, linewidth = 132)
print 'done import'
```

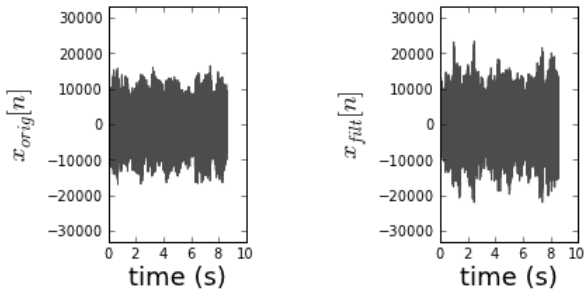
```
done import
```

In [343]:
```
# Graphing helper function
def setup_graph(fig,title='', x_label='', y_label='', xlim='', ylim='',
 fig_size=None, nplots=''):
    #fig = plt.figure()
    if fig_size != None:
        fig.set_size_inches(fig_size[0], fig_size[1])
    #ax = fig.add_subplot(nplots[0],nplots[1],nplots[2])
    ax = fig.add_subplot(nplots[0],nplots[1],nplots[2])
    ax.set_title(title)
    ax.set_xlabel(x_label,fontsize=18)
    ax.set_ylabel(y_label,fontsize=18)
    ax.set_xlim(xlim[0],xlim[1])
    ax.set_ylim(ylim[0],ylim[1])
```

Preliminary data processing: read the audio files

In [344]:
```
from scipy.io import wavfile
from scipy import signal
#### signal 2 original
rate1,data1 = wavfile.read('sig2-orig.wav')
data1_shape=shape(data1)
print('Raw signal 2 original shape: ',data1_shape)

rate2,data2 = wavfile.read('sig2-filt.wav');
data2_shape=shape(data2)
print('Raw signal 2 filtered shape: ',data2_shape)

rate3,data3 = wavfile.read('sig1-filt.wav');
data3_shape=shape(data3)
print('Raw signal 1 filtered shape: ',data3_shape)

sig2_orig = np.zeros(round(data1_shape[0]/2))
sig2_filt = np.zeros(round(data2_shape[0]))
sig1_filt_left = np.zeros(round(data3_shape[0]))
sig1_filt_right = np.zeros(round(data3_shape[0]))

# copy into individual channels for processing
#for i in range(0,int(data1_shape[0]/2)-1):
#    sig2_orig[i] = data1[i]/32768 # copy left channel
for i in range(0,size(data1)/2):
    sig2_orig[i] = data1[i]

for i in range(0,int(data2_shape[0])-1):
    sig2_filt[i] = data2[i][0] # copy the left channel
t1 = linspace(0,float(data2_shape[0])/float(rate1), data2_shape[0])

for i in range(0,int(data3_shape[0])-1):
    sig1_filt_left[i]=data3[i][0]
    sig1_filt_right[i]=data3[i][1]
t2 = linspace(0,float(data3_shape[0])/float(rate3),data3_shape[0])

print('Post-Process signal 2 original shape: ',shape(sig2_orig))
print('Post-Process signal 2 filtered shape: ',shape(sig2_filt))
print('Post-Process signal 1 filtered shape: ',shape(sig1_filt_left))

wavfile.write('sig2_orig_2.wav',rate1,sig2_orig)
wavfile.write('sig2_filt_2.wav',rate2,sig2_filt)

fig = plt.figure()
setup_graph(fig,title='', x_label='time (s)', y_label='$x_{orig}[n]$',
xlim=[0, 10], ylim=[-33000,33000], fig_size=(6,3), nplots=[1,3,1])
_ = plt.plot(t1, sig2_orig)
setup_graph(fig,title='', x_label='time (s)', y_label='$x_{filt}[n]$',
xlim=[0, 10], ylim=[-33000,33000], fig_size=(6,3), nplots=[1,3,3])
_ = plt.plot(t1, sig2_filt)
```

```
('Raw signal 2 original shape: ', (756473,))
('Raw signal 2 filtered shape: ', (378236, 2))
('Raw signal 1 filtered shape: ', (297224, 2))
('Post-Process signal 2 original shape: ', (378236,))
('Post-Process signal 2 filtered shape: ', (378236,))
('Post-Process signal 1 filtered shape: ', (297224,))
```



Calculate the Input/Output FFTs (FFT takes quite some time)

In [345]:
```
import numpy as np
sig2_orig_fft=np.fft.fft(sig2_orig,300000)
print('FFT of Signal 2 original computed')
sig2_filt_fft=np.fft.fft(sig2_filt,300000)
print('FFT of Signal 2 filtered computed')
w=linspace(0,2*pi,len(sig2_orig_fft))

sig1_filt_fft1=np.fft.fft(sig1_filt_left)
print('FFT of Signal 1 filtered computed')
sig1_filt_fft2=np.fft.fft(sig1_filt_right)
w_sig1=linspace(0,2*pi,len(sig1_filt_fft1))
print 'shape of orig, filt', np.shape(sig2_orig_fft), np.shape(sig2_fil
t_fft)
```

```
FFT of Signal 2 original computed
FFT of Signal 2 filtered computed
FFT of Signal 1 filtered computed
shape of orig, filt (300000,) (300000,)
```

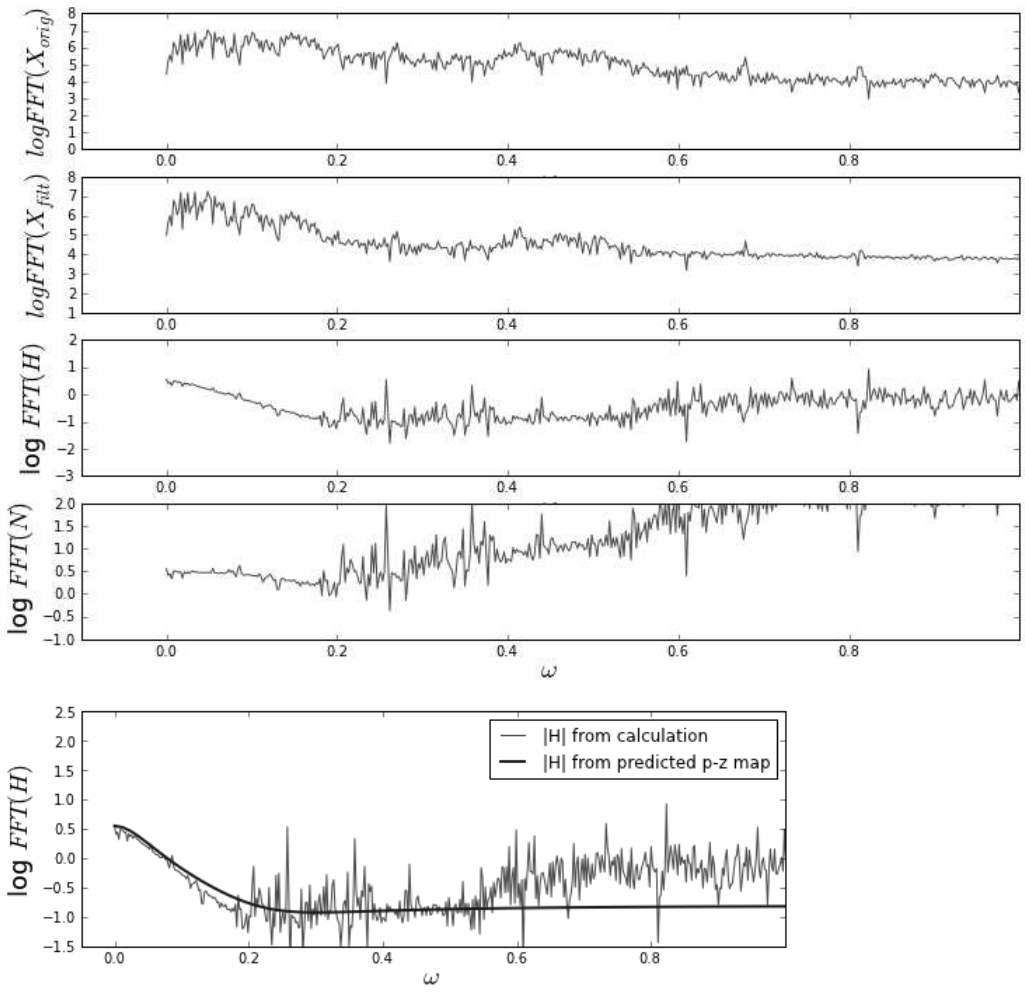
Plot Transfer Function Freq. Response and Estimate zero locations from it. \(H[k] = Y[k]/X[k]\)

\(H(z) = \frac{N(z)}{(z- 0.95)^2}\), where \(N(z)\) represents the unknown zero locations. Unknown zeros are then \(N(z) = (z - 0.95)^2 \cdot Y(z)/X(z)\)

In [346]:
```
# create an empty list to contain the complex transfer function
mov_avg = np.ones(10)/10.0
length = len(sig2_orig_fft)
#print 'shape of orig, filt', np.shape(sig2_orig_fft), np.shape(sig2_fi
lt_fft)
sig2_orig_fft = np.convolve(mov_avg, sig2_orig_fft)   # try to smooth o
ut FFT
sig2_filt_fft = np.convolve(mov_avg, sig2_filt_fft)   # try to smooth o
ut FFT
#w=linspace(0,2*pi,len(sig2_orig_fft))
H_freq = np.zeros(len(sig2_orig_fft),dtype=np.complex)
N_freq = np.zeros(len(sig2_orig_fft),dtype=np.complex)

print 'shape of w, orig, filt, H_freq=', np.shape(w_sig1), np.shape(sig
2_orig_fft), np.shape(sig2_filt_fft), np.shape(H_freq)

for i in range(0,length):
    H_freq[i]=sig2_filt_fft[i]/sig2_orig_fft[i]
    N_freq[i] = (1/.05)**2 * H_freq[i] * (np.exp(1j*w[i]) - 0.95)**2.0
  # compensate for 2 known poles

print(len(w),len(abs(sig2_orig_fft)))
# Plot the FFTs: 1) Original, 2) Filtered, 3) Transfer Function
fig = plt.figure()
setup_graph(fig,title='', x_label='$ \omega$ ', y_label='$log FFT(X_{or
ig})$', xlim=[-0.1, 1], ylim=[0,8], fig_size=(12,8),nplots=[4,1,1])
_ = plt.plot(w[0:length:100],np.log10(abs(sig2_orig_fft[0:length:100]))
)
setup_graph(fig,title='', x_label='$\omega$ ', y_label='$log FFT(X_{fil
t})$', xlim=[-0.1, 1], ylim=[1,8], fig_size=(12,8),nplots=[4,1,2])
_ = plt.plot(w[0:length:100],np.log10(abs(sig2_filt_fft[0:length:100]))
)
setup_graph(fig,title='', x_label='$\omega$ ', y_label='log $FFT(H)$',
xlim=[-0.1, 1], ylim=[-3,2], fig_size=(12,8),nplots=[4,1,3])
_ = plt.plot(w[0:length:100],np.log10(abs(H_freq[0:length:100])))
setup_graph(fig,title='', x_label='$\omega$ ', y_label=' log $FFT(N)$',
 xlim=[-0.1, 1], ylim=[-1,2], fig_size=(12,8),nplots=[4,1,4])
_ = plt.plot(w[0:length:100],np.log10(abs(N_freq[0:length:100])))


# Zoomed in transfer function. Note from the first 3 figs (on top) that
 past w=0.5 the original (i.e. pre-filtered) signal is ~= 0.
# So, we'll focus on estimating H(z) in the frequency regime from 0 to
0.5 (the red curve)
fig = plt.figure()
setup_graph(fig,title='', x_label='$\omega$ ', y_label='$FFT(H)$', xlim
=[-0.05, 1], ylim=[-1.5,2], fig_size=(9,3),nplots=[1,1,1])
_ = plt.plot(w[0:length:100],np.log10(abs(H_freq[0:length:100])))

# Estimating the zero locations. From the third figure, notice that the
 conjugate zeros are around w=0.24 (notice the gentle rise in H).
# Furthermore, |H| ~= 0 at w=0.24, so the zero is close to the unit cir
cle. Lets plot a freq. response for estimated zero locations
r=0.9 # estimated a number close to the unit circle
phi=0.22 # approximately where the dip in FFT of H is
z1=complex(r*cos(phi),r*sin(phi))

zer=[1,-(z1+z1.conjugate()),z1*z1.conjugate()]
west,hest=signal.freqz(zer,[1, -2*0.95, pow(0.95,2)],whole=1) # Compute
 impulse response

# Normalized the predicted H(z) to illustrate its comparison to the cal
culated H
setup_graph(fig,title='', x_label='$\omega$ ', y_label='log $FFT(H)$',
xlim=[-0.05, 1], ylim=[-1.5,2.5], fig_size=(9,3),nplots=[1,1,1])
_ = plt.plot(west,np.log10(abs(hest)/6),'r',linewidth=2)
plt.legend(['|H| from calculation','|H| from predicted p-z map'])
```

```
shape of w, orig, filt, H_freq= (297224,) (300009,) (300009,) (300009,)
(300000, 300009)
```

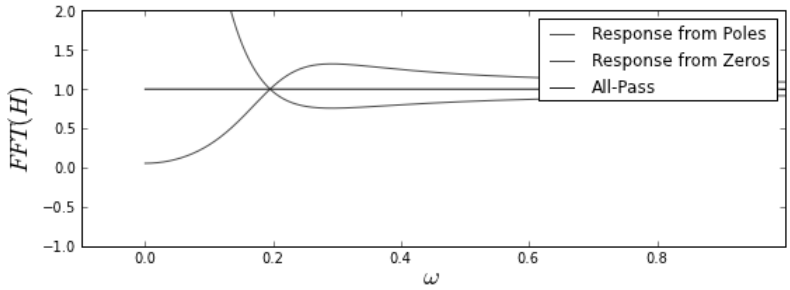
```
Out[346]: <matplotlib.legend.Legend at 0x55a6bd30>
```

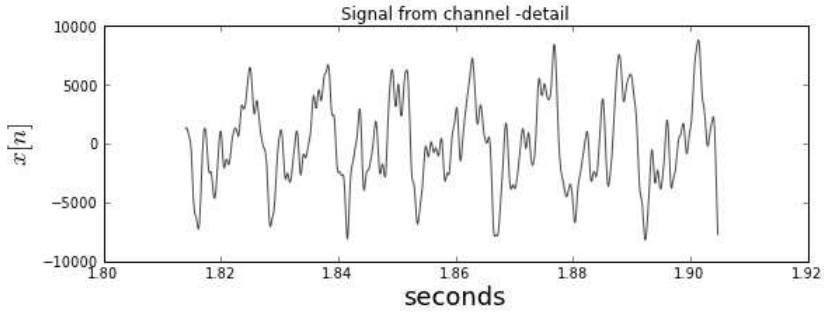


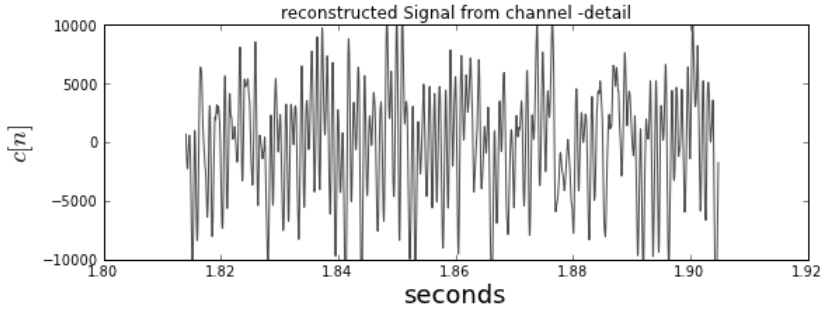
Problem 5 d. Apply Hc(z) LDE to reconstruct original signal 1

In [347]:
```
from scipy.io import wavfile
from scipy import signal
# set up the inverse transfer function Hc(z)
r=0.9 # estimated a number close to the unit circle
phi=0.2 # approximately where the dip in FFT of H is
z1=complex(r*cos(phi),r*sin(phi))

zer=[1,-(z1+z1.conjugate()),z1*z1.conjugate()]

west,hest=signal.freqz(zer,[1, -2*0.95, pow(0.95,2)],(len(w_sig1)),whol
e=1) # H(z)
wc,hc=signal.freqz([1, -2*0.95, pow(0.95,2)],zer,(len(w_sig1)),whole=1)
 # Hc(z)

fig = plt.figure()
setup_graph(fig,title='', x_label='$\omega$ ', y_label='$FFT(H)$', xlim
=[-0.1, 1], ylim=[-1,2], fig_size=(9,3),nplots=[1,1,1])
_ = plt.plot(west,(abs(hest)))
setup_graph(fig,title='', x_label='$\omega$ ', y_label='$FFT(H)$', xlim
=[-0.1, 1], ylim=[-1,2], fig_size=(9,3),nplots=[1,1,1])
_ = plt.plot(wc,(abs(hc)))
setup_graph(fig,title='', x_label='$\omega$ ', y_label='$FFT(H)$', xlim
=[-0.1, 1], ylim=[-1,2], fig_size=(9,3),nplots=[1,1,1])
_ = plt.plot(wc,(abs(hc*hest)))
plt.legend(['Response from Poles','Response from Zeros','All-Pass'])

# set up 16 bit vectors to hold output
sig1_recons_left = np.zeros(len(sig1_filt_left),int16)  #output
sig1_recons_right= np.zeros(len(sig1_filt_right),int16)

y2,y1,y0 = 0.0,0.0,0.0 # initial condition
x2,x1,x0 = 0.,0.,0.
yright= np.zeros(length, int16)

y2,y1,y0 = 0.0,0.0,1.0 # initial condition
x2,x1,x0 = 0.,0.,0.
ro = 1.0/1.05 # zero radius
ctheta = np.cos(2*np.pi*1.5/44.1)
rp = 0.95 # pole radius
ctp = np.cos(2*np.pi*0.0/44.1)
yscale = 50 # normalize

length = len(sig1_filt_left)
print 'length of signal =', length

print '2rpctp, rp**2', 2*rp*ctp, rp**2
# swap poles and zeros from original function

for i in range(0,length):
#for i in range(0,1000):
    y2 = y1             # y[n-2] = y[n-1]
    y1 = y0         # y[n-1]=y[n]
    x2 = x1             # x[n-2] = x[n-1]
    x1 = x0             # x[n-1] = x
    x0 = 0.2* sig1_filt_left[i]         # x[n]
#    y0 = x0 - 2.0*ro*ctheta * x1 + ro**2 * x2 + 2.0*rp*ctp*y1 - rp**2
* y2 # filter function
# inverse filter function:
    y0 = x0 - 2.0*rp*ctp * x1 + rp**2 * x2 + 2.0*ro*ctheta*y1 - ro**2 *
 y2
#    print 'n, x[n], y[n], y[n-1], y[n-2]', i, x0,y0, y1, y2
    sig1_recons_left[i] = np.int16(yscale*y0)   # set left channel
    sig1_recons_right[i] = np.int16(yscale*y0)   # set left channel


#####  code for calculating calculating LDE on left[] and right[] (same
 for both)
y=[1.0,0.0,0.0]
x=[0,0,0]
#for i in range(3,len(t2)):
#    #y.append(x[i]-2*0.95*x[i-1]+pow(0.95,2)*x[i-2]+(z1+z1.conjugate()
)*y[i-1]-z1*z1.conjugate()*y[i-2])
#    y.append(sig1_filt_left[i]-2*0.95*sig1_filt_left[i-1]+pow(0.95,2)*
sig1_filt_left[i-2]+(z1+z1.conjugate())*y[i-1]-z1*z1.conjugate()*y[i-2]
)
# sig1_recons_left = y

#####   plot resulting output before and after
fig = plt.figure()
setup_graph(fig,title='Signal from channel -detail', x_label='seconds',
 y_label='$x[n]$', xlim=[1.80, 1.92], ylim=[-10000, 10000], fig_size=(9
 3) nplots=[1 1 1])
```

```
length of signal = 297224
2rpctp, rp**2 1.9 0.9025
```

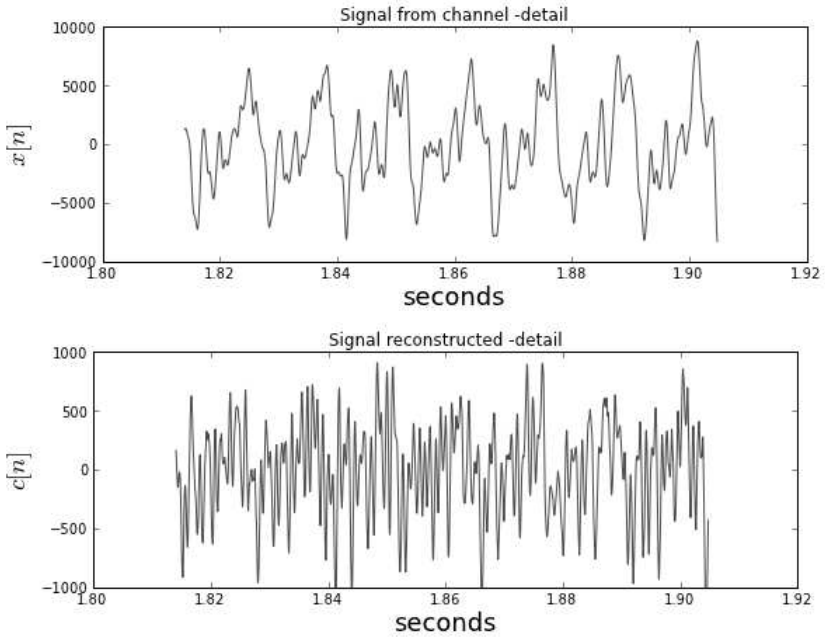






Reconstruct by taking IFFT

In [348]:
```
# set up the inverse transfer function Hc(z)
r=0.9 # estimated a number close to the unit circle
phi=0.2 # approximately where the dip in FFT of H is
z1=complex(r*cos(phi),r*sin(phi))

zer=[1,-(z1+z1.conjugate()),z1*z1.conjugate()]

west,hest=signal.freqz(zer,[1, -2*0.95, pow(0.95,2)],(len(w_sig1)),whol
e=1) # H(z)
wc,hc=signal.freqz([1, -2*0.95, pow(0.95,2)],zer,(len(w_sig1)),whole=1)
 # H_c(z)

# set up 16 bit vectors to hold output
sig1_recons_left = np.zeros(len(sig1_filt_left),int16)  #output
sig1_recons_right= np.zeros(len(sig1_filt_right),int16)

sig1_recon_freq = np.zeros(len(sig1_filt_fft1),np.complex)
for i in range(0,len(sig1_filt_fft1)):
    sig1_recon_freq[i] = sig1_filt_fft1[i] * hc[i]
sig1_recons_left=np.fft.ifft(sig1_recon_freq)

#####   plot resulting output before and after
fig = plt.figure()
setup_graph(fig,title='Signal from channel -detail', x_label='seconds',
 y_label='$x[n]$', xlim=[1.80, 1.92], ylim=[-10000, 10000], fig_size=(9
,3),nplots=[1,1,1])
_ = plt.plot(t2[80000:84000],sig1_filt_left[80000:84000])
# plot recovered signal:
fig = plt.figure()
setup_graph(fig,title='Signal reconstructed -detail', x_label='seconds'
, y_label='$c[n]$', xlim=[1.80, 1.92], ylim=[-1000, 1000], fig_size=(9,
3),nplots=[1,1,1])
_ = plt.plot(t2[80000:84000],sig1_recons_left[80000:84000])

wavfile.write('sig1_reconst2.wav',44100,np.asarray(sig1_recons_left,dty
pe=int16))
```



In [348]: