Systems and Signals 414 Practical 4: Poles and zeros

Aim: Understand the effect of poles and zeros on the impulse response of LTI systems, both in the time and frequency domains.

Hand in: Please hand in this notebook as a PDF file on sunlearn by Sunday, 22 April at 23:55. To save your notebook to a PDF, you can print the notebook from your browser and then choose to `Save as PDF`. (If you are doing the practical on a machine with LaTeX, you can also select `File`$\rightarrow$ `Download as` $\rightarrow$ `PDF via LaTeX (.pdf)` directly in the notebook). After exporting your notebook, upload the PDF by clicking on `Practical 3 submission` on sunlearn and following the steps. You may submit your work multiple times; only the last submission will be marked. **No late submissions will be accepted.**

Task: Do the following assignment using Jupyter. Document the task indicating your methodology, theoretical results, numerical results and discussions as necessary. Your graphs should have labeled axes with the correct units indicated. If you get stuck with a Numpy or Scipy function, go look up the usage at https://docs.scipy.org (https://docs.scipy.org). Also take a look at the provided coding examples.

Preamble code and helper functions:

In [25]:

```
#%matplotlib notebook
%matplotlib inline

#All the necessary imports
import pylab as pl
import numpy as np
from scipy import signal
import IPython.display

#Nicer matplotlib dimensions
pl.rcParams['figure.figsize'] = (9,3)

#A helper-function to setup a proper plot
def setup_plot(title, y_label='', x_label='', newfig=True):
    if newfig:
        pl.figure()
    pl.margins(*(pl.array(pl.margins())+0.05))
    pl.title(title)
    pl.ylabel(y_label)
    pl.xlabel(x_label)

#Download yesterday.wav from courses.ee.sun.ac.za and return it as a numpy a
rray
def download_and_load_audio(url, mono=True, factor_of_2_length=True):
    import os
    import urllib
    import scipy.io
    from scipy.io import wavfile

    filename = os.path.split(url)[-1]
    #Download if path does not already exist
    if not os.path.isfile(filename):
        urllib.request.urlretrieve(url, filename)
    sample_frequency, signal_array = wavfile.read(filename)
    #Normalise signal and return
    if mono and len(signal_array.shape)==2:
        signal_array = np.sum(signal_array, axis=1)
    signal_array = signal_array/np.max([np.max(signal_array),
                                        -np.min(signal_array)])
    if factor_of_2_length:
        signal_array = signal_array[:2**np.floor(np.log2(len(signal_array)))
                                    .astype('int')]

    return sample_frequency, signal_array

#Adapted from https://gist.github.com/endolith/4625838
def zplane(zeros, poles):
    """
    Plot the complex z-plane given zeros and poles.
    """
    zeros=np.array(zeros);
    poles=np.array(poles);
    ax = pl.gca()

    # Add unit circle and zero axes
    unit_circle = pl.matplotlib.patches.Circle((0,0), radius=1, fill=False,
                             color='black', ls='solid', alpha=0.6)
    ax.add_patch(unit_circle)
    pl.axvline(0, color='0.7')
    pl.axhline(0, color='0.7')

    #Rescale to a nice size
    rscale = 1.2 * np.amax(np.concatenate((abs(zeros), abs(poles), [1])))
    pl.axis('scaled')
    pl.axis([-rscale, rscale, -rscale, rscale])

    # Plot the poles and zeros
    polesplot = pl.plot(poles.real, poles.imag, 'x', markersize=9)
```

Functions necessary for this practical

Scipy provides a good selection of signal processing tools in scipy.signal. Note the following functions important for this practical:

- `signal.freqz(b, a, whole=True, ...)`
  We use this function for plotting purposes only. It returns ω coordinates to illustrate the frequency response of a given filter. Use the keyword arguments `whole=True` to return for $\omega = 0$ up to $\omega = 2\pi$. The filter is of type:
  $$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots}$$
- `signal.lfilter(b, a, x, ...)`
  Given a signal $x[n]$, this function will apply the difference equation given by $\mathbf{b}$ and $\mathbf{a}$ on the input signal and returns the result (this is what you had to painstakingly code up in Practicals 1 and 2). Note the difference equation is of type:
  $$a_0 y[n] = -a_1 y[n-1] - a_2 y[n-2] \ldots + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] \ldots$$
- `signal.tf2zpk(b, a)`
  This returns poles and zeros parameters $\mathbf{z}$, $\mathbf{p}$, and k as output from filter parameters $\mathbf{b}$ and $\mathbf{a}$ as input, with the filter given by:
  $$H(z) = \frac{b_0 z^M + b_1 z^{M-1} + b_2 z^{M-2} + \ldots}{a_0^N + a_1 z^{N-1} + a_2 z^{N-2} + \ldots} = k \frac{(z - z_0)(z - z_1)\ldots}{(z - p_0)(z - p_1)\ldots}$$
- `signal.zpk2tf(z, p, k)`
  The reverse conversion as provided by `signal.tf2zpk`.
- `np.unwrap(p, ...)`
  Use this function to remedy unwanted angular wrapping between $-\pi$ and π for linearly-increasing angles. You may have noticed this wrapping as the sawtooth-type effect on your phase plots.

Questions

Question 1: Poles and zeros investigation

Consider the LTI system (filter) with transfer function $H(z)$ given by

$$H(z) = \frac{1 - 2\cos(\theta_1)r_1 z^{-1} + r_1^2 z^{-2}}{1 - 2\cos(\theta_2)r_2 z^{-1} + r_2^2 z^{-2}},$$

with the following default parameter values:

$$\theta_1 = \frac{3}{8}2\pi,\ \ \theta_2 = \frac{1}{8}2\pi,\ \ r_1 = 0.95,\ \ r_2 = 0.95,$$

1. Determine the locations of the poles and zeros of $H(z)$ in terms of θ_1, θ_2, r_1, r_2 by hand. Substitute the parameters with their default values and plot a pole-zero diagram. You may use the given `zplane` function to accomplish this.
2. Keeping the other parameters at their default values, vary r_1 over the intervals $r_1 = \{0.0, 0.5, 0.8, 1.0, 1.05\}$ and follow the subquestions. Plot the A, B, and C plots on the same figure (refer to the sub-plotting code example), label your plots clearly, and indicate the varying parameter and value in the plot title.
   A. Investigate the effect of this variation on the placement of poles and zeros using `zplane`.
   B. Investigate the effect of this variation on both the magnitude and phase responses of the LTI system with `signal.freqz`. Use linearly scaled axes for frequencies and phases, and decibel scaling $20\log_{10}(|A|)$ for amplitudes (note `np.log10`). Note and use the function `np.unwrap` to lesser the discontinuity of your phase plots.
   C. Investigate the effect of this variation on the impulse response of the system using `signal.lfilter`.
   D. Explain your observations in view of the locations of the poles and zeros of $H(z)$.
3. Repeat Question 2 but now only vary r_2 over the intervals $r_2 = \{0.0, 0.5, 0.8, 1.0, 1.05\}$, and setting all other parameters to their default values.
4. Repeat Question 2 but now only vary θ_1 over the intervals $\theta_1 = \{0, \frac{1}{8}2\pi, \frac{1}{4}2\pi, \frac{3}{8}2\pi, \frac{1}{2}2\pi\}$, and setting all other parameters to their default values..
5. Repeat Question 2 but now only vary θ_2 over the intervals $\theta_2 = \{0, \frac{1}{8}2\pi, \frac{1}{4}2\pi, \frac{3}{8}2\pi, \frac{1}{2}2\pi\}$, and setting all other parameters to their default values.
6. Now let $r_2 = 1.0$, while the other parameters take on their default values.
   A. Using `signal.lfilter`, determine the output of the system when sinusoids with frequencies $\omega_1 = 0.11(2\pi)$, $\omega_2 = 0.125(2\pi)$ and $\omega_3 = 0.135(2\pi)$ are applied to it (separately).
   B. Explain your observations in view of a plot of the poles and zeros of $H(z)$.

Question 2: Digital filter investigation

Now consider the LTI system (filter) with transfer function $H(z)$ given by

$$H(z) = \frac{0.0038 + 0.0001z^{-1} + 0.0051z^{-2} + 0.0001z^{-3} + 0.0038z^{-4}}{1 - 3.2821z^{-1} + 4.2360z^{-2} - 2.5275z^{-3} + 0.5865z^{-4}},$$

that is

```
b = [0.0038, 0.0001, 0.0051, 0.0001, 0.0038]
a = [1, -3.2821, 4.2360, -2.5275, 0.5865]
```

1. Plot the pole-zero diagram for this system. Hint: `signal.tf2zpk` might be helful.
2. Determine the system's magnitude and phase response using `signal.freqz`. Can you explain the frequency response of the system from the poles and zeros?
3. What type of filter is this system? Verify your answer by filtering the following signal consisting of sinusoids spaced out in frequency:
   ```
   n = np.arange(1000)
   x = np.sum(np.cos(fwi*2*np.pi*n) for fwi in np.linspace(0, 0.15, 16))
   ```
   Plot your signal and filtered results in both the discrete-time and frequency domain. You can also use a sampling frequency of $f_s = 8$ kHz and listen to the inputs and outputs using `IPython.lib.display.Audio`

Additional Question 1: Digital filter design

Note that this question is not compulsory.

Consider the discrete-time signal $x[n]$ by loading in `burnafterreading.wav` as a numpy array using the `download_and_load_audio` function.

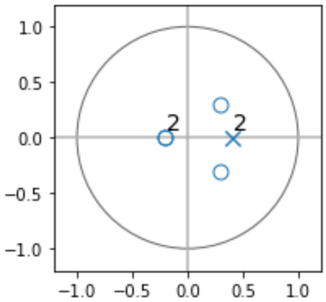
Someone found this audio clip on a compact disc lying on the floor and it appears that the audio clip was sabotaged to deter eavesdroppers. Load and listen to $x[n]$ using `IPython.display.Audio` and plot the frequency in order to get a feeling for how the signal is corrupted. You are curious to find out what valuable information might be on that cd and decided to design a filter to filter out the noisy spectrum.

Using pole and zero placements, design any filter of your choice to remove the noisy spectrum of the corrupted signal, and apply this filter on $x[n]$. Document your findings.

Coding examples

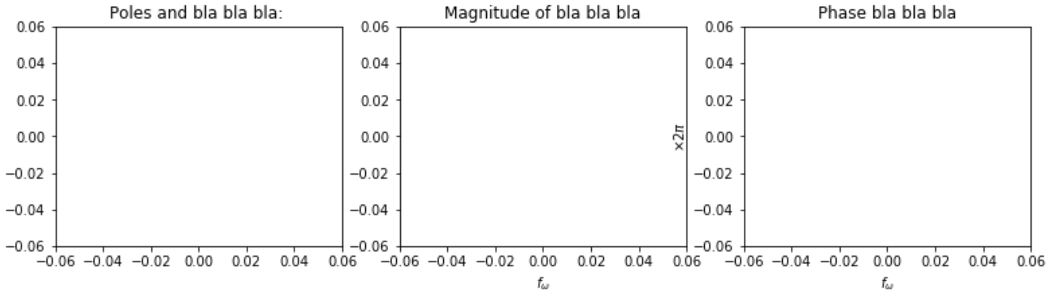
Example of poles-zero plot

In [26]:
```
setup_plot('');
zplane([0.3+0.3j, 0.3-0.3j, -0.2, -0.2], [0.4, 0.4]);
```



Example of sub-plotting

In [27]:
```
pl.figure(figsize=(13,3))

pl.subplot(1, 3, 1)
setup_plot('Poles and bla bla bla: ', newfig=False)
#plot here

pl.subplot(1, 3, 2)
setup_plot('Magnitude of bla bla bla', x_label=r'$f_\omega$', newfig=False)
#plot here

pl.subplot(1, 3, 3)
setup_plot(r'Phase bla bla bla', y_label=r'$\times 2 \pi$', x_label=r'$f_\om
ega$', newfig=False)
#plot here

#plot here
```



Note the difference between linspace and arange:

In [28]:
```
#Every second element from 0 upto (but excluding) 10
print('a)', np.arange(0,10,1))

#5 elements from 0 upto (but excluding) 10
print('b)', np.linspace(0,10,10,False))

#5 elements from 0 upto 10 (avoid!!!)
print('c)', np.linspace(0,10,10))
```

```
a) [0 1 2 3 4 5 6 7 8 9]
b) [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
c) [ 0.          1.11111111  2.22222222  3.33333333  4.44444444  5.55555556
  6.66666667  7.77777778  8.88888889 10.        ]
```

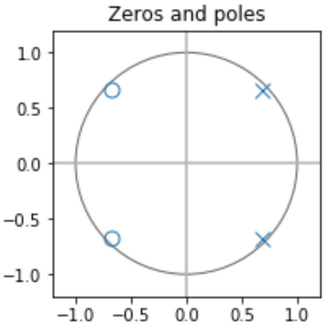
Audio player with Ipython.lib.display.Audio

In [29]:
```
example_signal = np.sin(2*np.pi*500*np.linspace(0,5,50000))
IPython.lib.display.Audio(rate=10000,
                          data=example_signal)
```

Out[29]:

Answer space:

In [30]:
```
#Q 1.1
t1 = 3/8
t2 = 1/8
r1 = 0.95
r2 = 0.95

zeros, poles, something= signal.tf2zpk([1, -2 * np.cos(2 * np.pi * t1)*r1, r
1**2], [1, -2 * np.cos(2 * np.pi * t2)*r2, r2**2])
setup_plot('Zeros and poles')
zplane(zeros, poles)

'''
We know that r1 and r2 represent the radius of the zeroes and poles respecti
vely.
Similarly we know that t1 and t2 are the angles of the zeroes and poles
'''
```

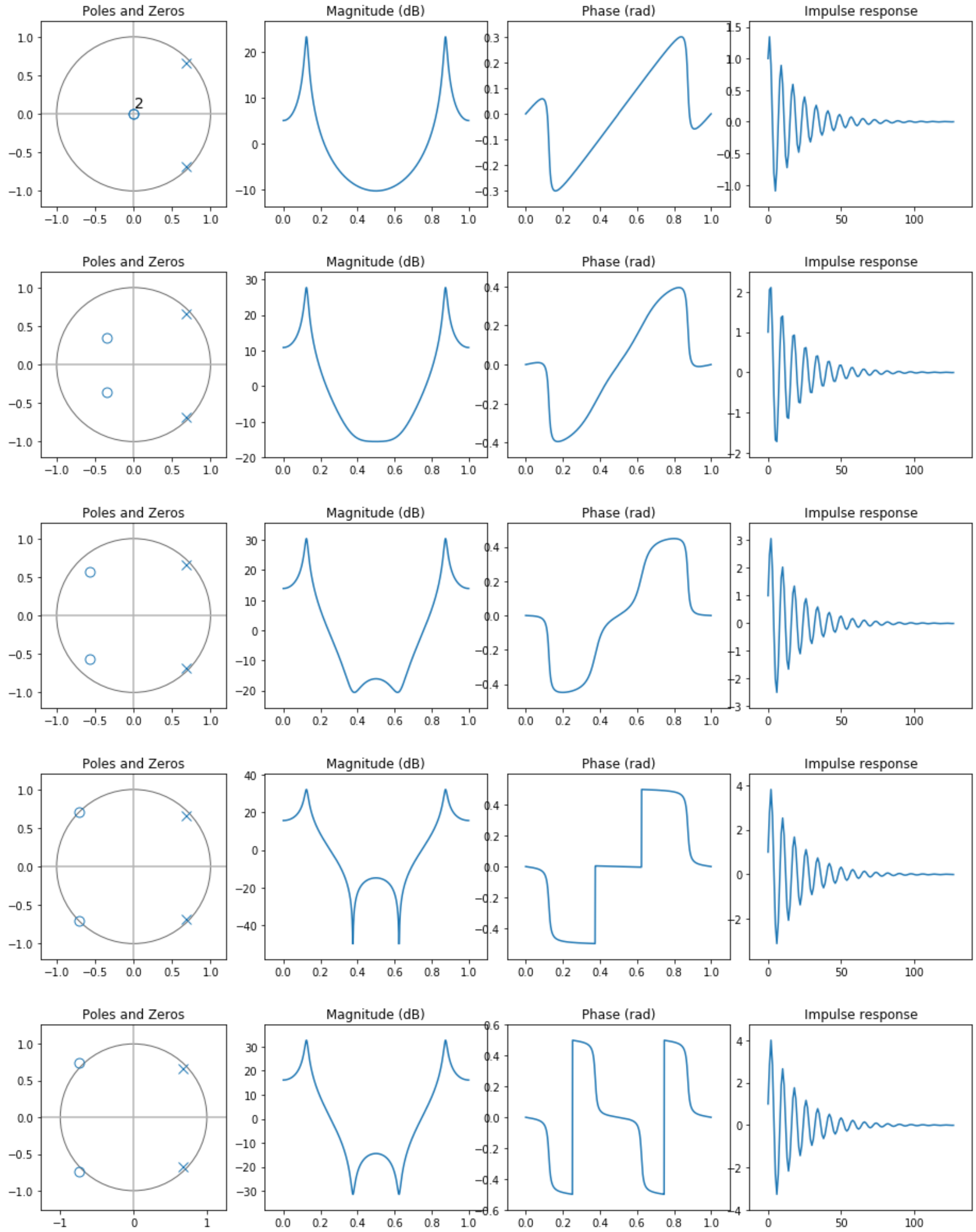


In [31]:
```
#1.2
def ploter(r1=r1, r2=r2, theta1=t1, theta2=t2):

    par1 = [1, -2*np.cos(2* np.pi * theta1)*r1, r1*r1]
    par2 = [1, -2*np.cos(2* np.pi * theta2)*r2, r2*r2]
    z, p, k = signal.tf2zpk(par1, par2)

    w, freq_response = np.array(signal.freqz(par1, par2))

    magnit_response = np.abs(np.append(freq_response, freq_response[::-1]))

    phase_response = np.angle(freq_response)
    phase_response = np.append(phase_response, -1 * phase_response[::-1]) /
(2* np.pi)

    w  = np.linspace(0, 1, 2*w.size, False)

    #Create an impulse function
    imp = np.zeros(int(w.size/8))
    imp[0] = 1

    #Generate filter response
    response = np.real(np.array(signal.lfilter(par1, par2, imp)))

    print("Theta1=",theta1," Theta2=",theta2," r1=",r1," r2=",r2)

    pl.figure(figsize=(13,3))
    pl.subplot(1, 4, 1)
    pl.tight_layout()
    setup_plot("Poles and Zeros", newfig=False)
    zplane(z, p)

    pl.subplot(1, 4, 2)
    setup_plot('Magnitude (dB)', newfig=False)
    pl.plot(w, 20*np.log10(magnit_response))

    pl.subplot(1, 4, 3)
    setup_plot('Phase (rad)', newfig=False)
    pl.plot(w, np.unwrap(phase_response))

    pl.subplot(1, 4, 4)
    setup_plot('Impulse response', newfig=False)
    pl.plot(response)


rs = np.array([0, 0.5, 0.8, 1, 1.05])
ts = np.array([0, 1/8, 1/4, 3/8, 1/2])

for r in rs:
    ploter(r)

'''
As r1 gets increased the magnitude of the frequency graph is pulled down at
theta1. The phase change also becomes very steep as the impulse response mag
nitude changes at theta1.
'''
```

```
Theta1= 0.375  Theta2= 0.125  r1= 0.0  r2= 0.95
Theta1= 0.375  Theta2= 0.125  r1= 0.5  r2= 0.95
Theta1= 0.375  Theta2= 0.125  r1= 0.8  r2= 0.95
Theta1= 0.375  Theta2= 0.125  r1= 1.0  r2= 0.95
```

```
/home/pgoos/.local/lib/python3.6/site-packages/ipykernel_launcher.py:34: Runt
imeWarning: divide by zero encountered in log10
```

```
Theta1= 0.375  Theta2= 0.125  r1= 1.05  r2= 0.95
```

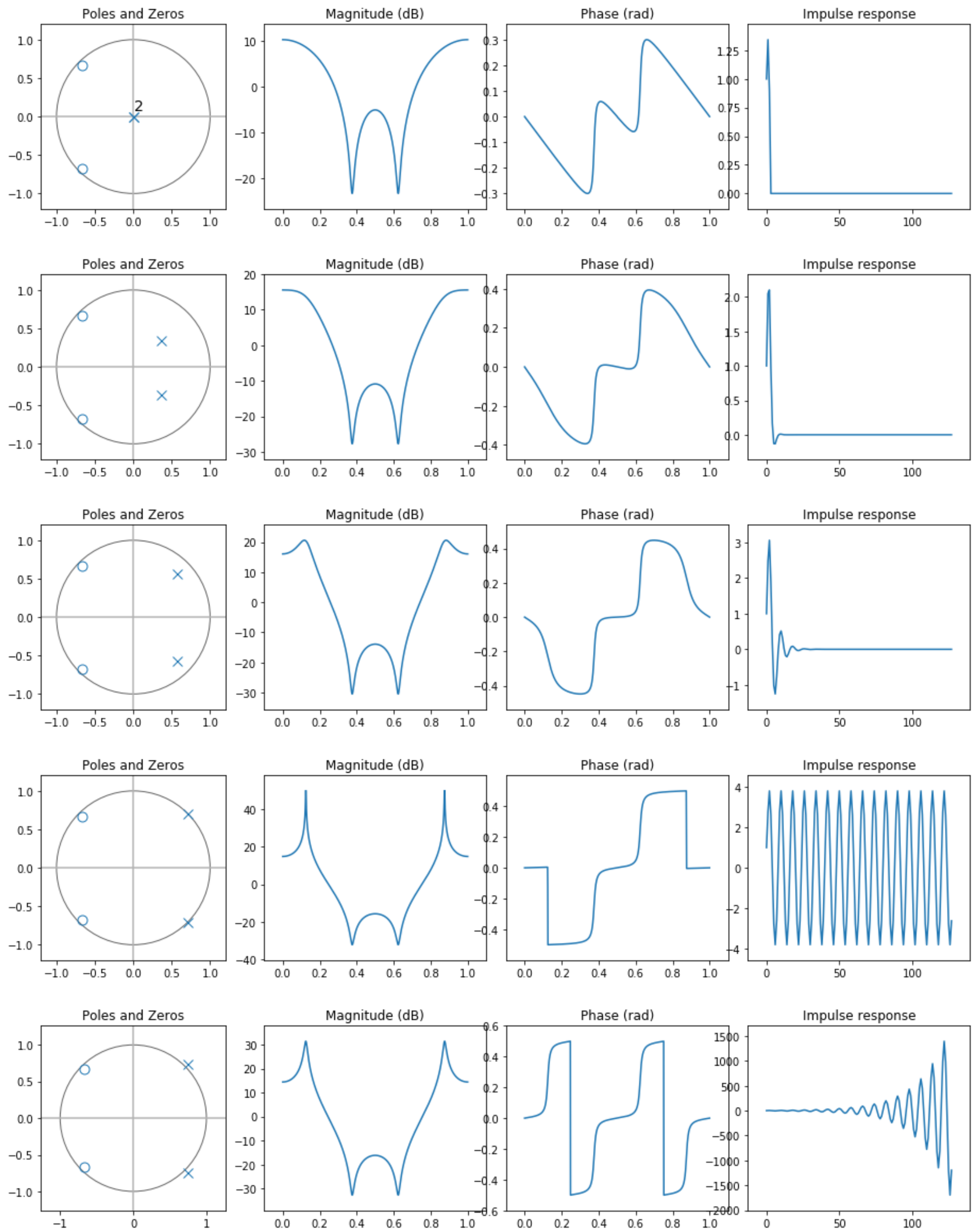


In [32]:
```
#Q 1.3
for r in rs:
    ploter(r2=r)

'''
As r2 increases the magnitude on the freq plot gets pulled up at theta2. Aga
in resulting in a sharp change in the phase at theta2. The impulse response
becomes less damped until oscilations and eventually unstable
'''
```

```
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 0.0
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 0.5
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 0.8
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 1.0
```

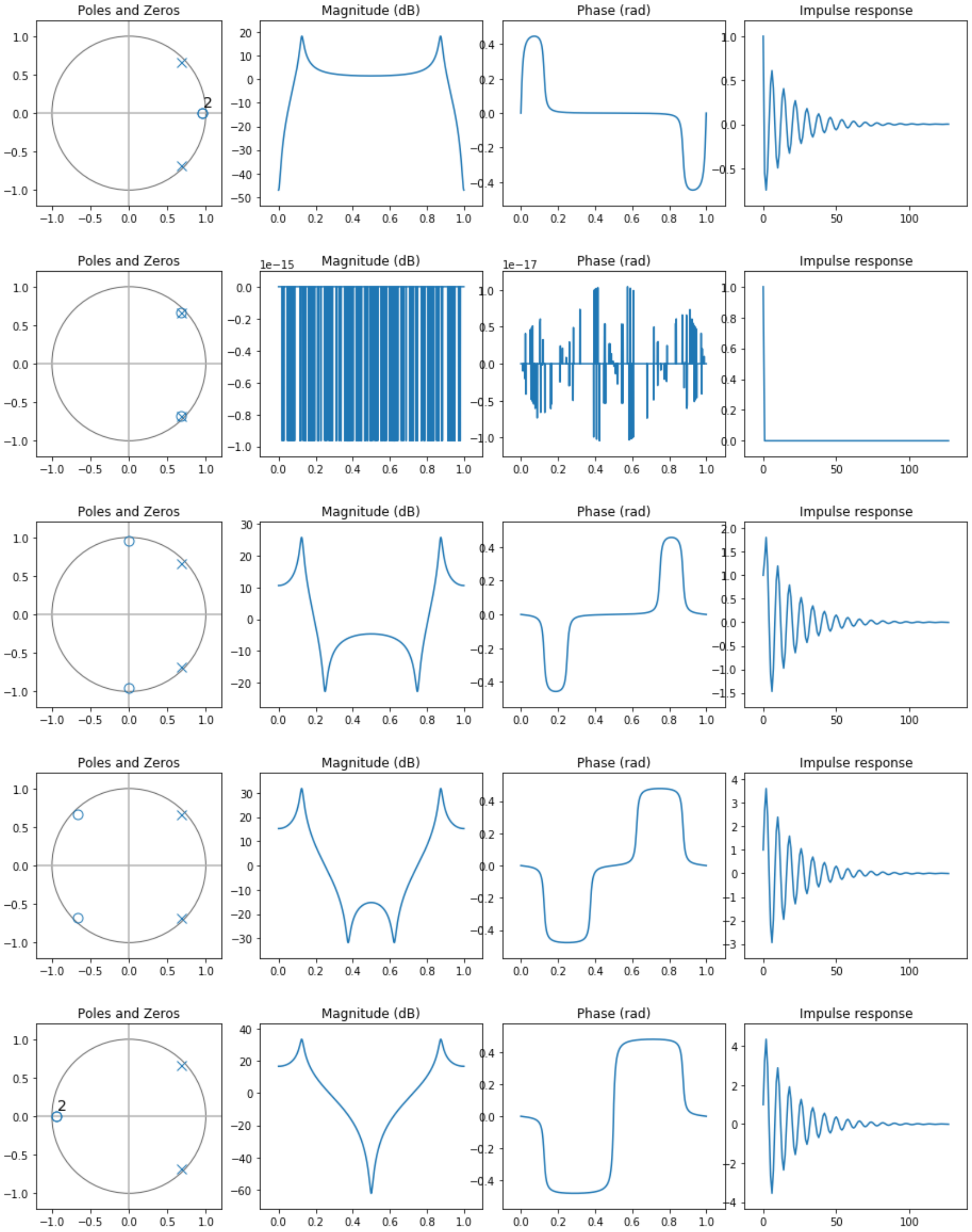
```
/usr/local/lib/python3.6/dist-packages/scipy/signal/filter_design.py:444: Run
timeWarning: divide by zero encountered in true_divide
  npp_polyval(zm1, a, tensor=False))
```

```
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 1.05
```



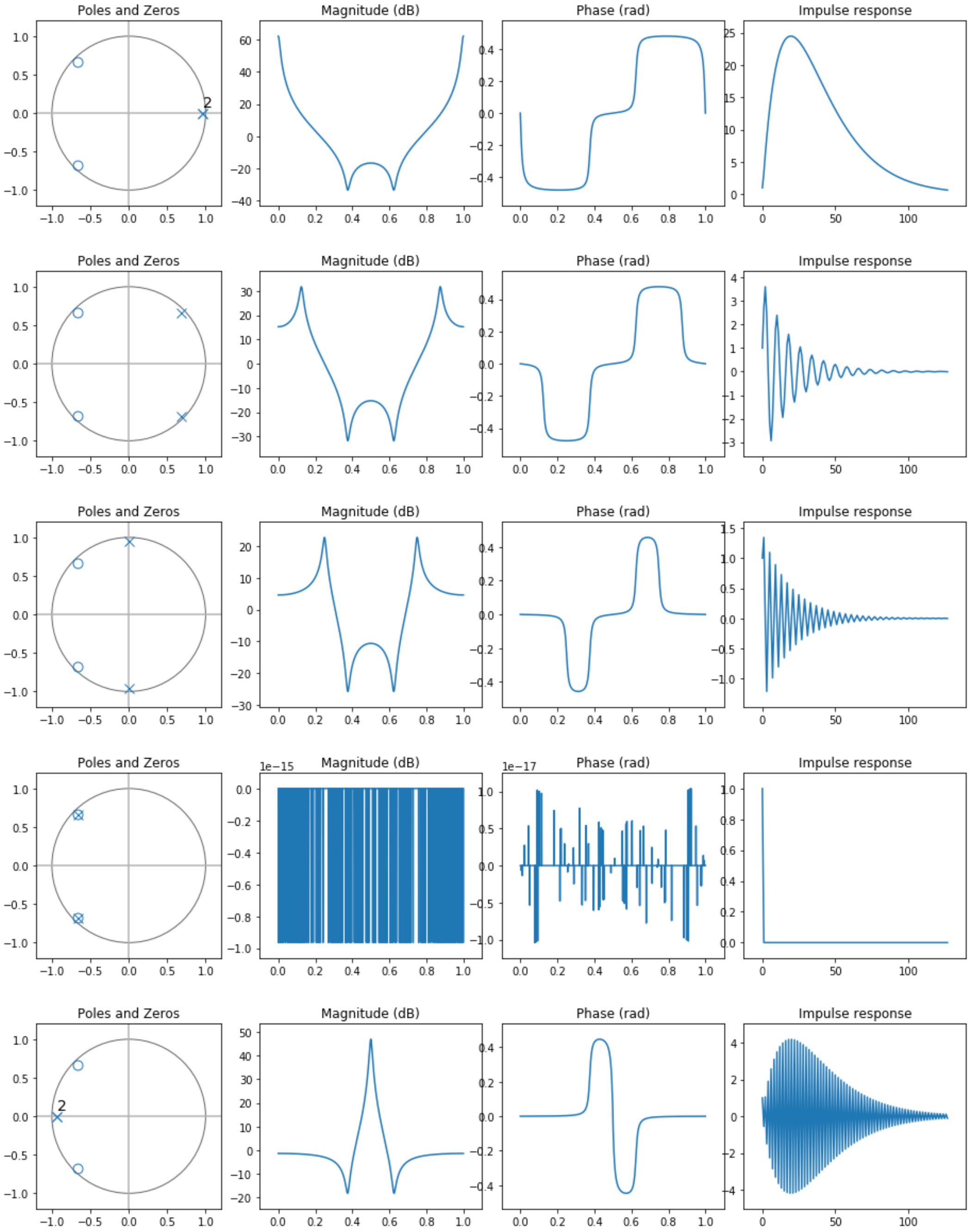
In [33]:
```
#Q 1.4
for t in ts:
    ploter(theta1=t)

'''
As theta1 increases the magnitude graph gets pulled down and they pull harde
r the closer the zeroes are to eachother
The impulse response grows as the distance between poles and zeros increases
NOTE SCALES CHANGE A LOT IN THIS QUESTION!
'''
```

```
Theta1= 0.0  Theta2= 0.125  r1= 0.95  r2= 0.95
Theta1= 0.125  Theta2= 0.125  r1= 0.95  r2= 0.95
Theta1= 0.25  Theta2= 0.125  r1= 0.95  r2= 0.95
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 0.95
Theta1= 0.5  Theta2= 0.125  r1= 0.95  r2= 0.95
```



In [35]:
```
# 1.5
for t in ts:
    ploter(theta2=t)
'''
As theta2 increases the magnitude plot gets pulled up as the poles get close
r the pill is stronger
The phase begins to narrow as the poles get closer
The impulse response becomes less damped
ONCE AGAIN, NOTICE THE SCALES
'''
```

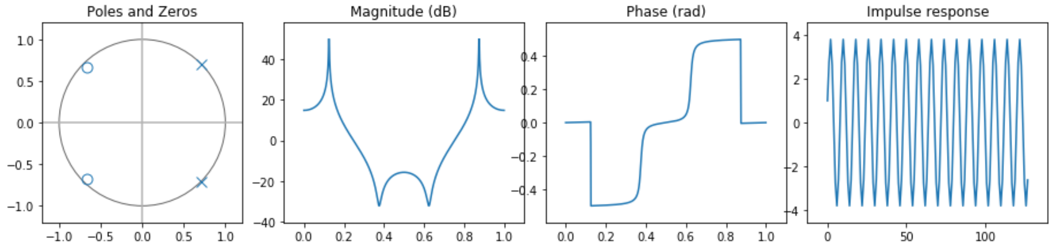
```
Theta1= 0.375  Theta2= 0.0  r1= 0.95  r2= 0.95
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 0.95
Theta1= 0.375  Theta2= 0.25  r1= 0.95  r2= 0.95
Theta1= 0.375  Theta2= 0.375  r1= 0.95  r2= 0.95
Theta1= 0.375  Theta2= 0.5  r1= 0.95  r2= 0.95
```



In [36]:
```
#Q 1.6
ploter(r2=1)
'''
As these poles & 0s are on the unit circle the magnitude tends to infinity a
t theta2.
The phase also shifts drastically at theta 2
The systems oscillates as expected
'''
```

```
/usr/local/lib/python3.6/dist-packages/scipy/signal/filter_design.py:444: Run
timeWarning: divide by zero encountered in true_divide
  npp_polyval(zm1, a, tensor=False))
```

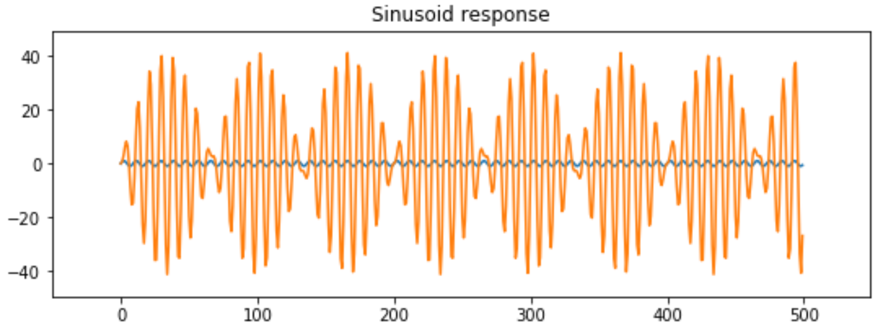
```
Theta1= 0.375  Theta2= 0.125  r1= 0.95  r2= 1
```



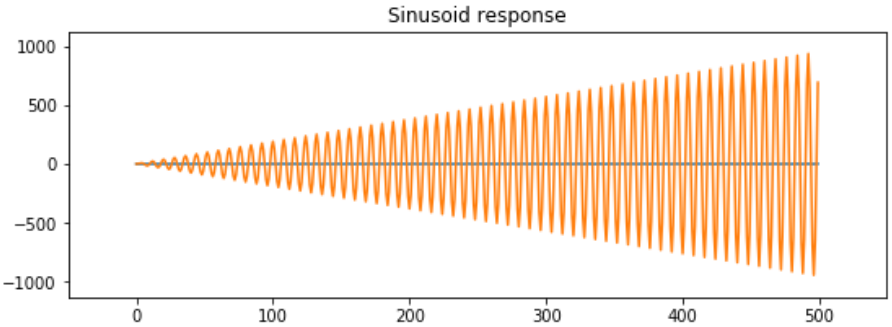
In [37]:
```
#Q 1.7
b = [1, -2*np.cos(2 * np.pi * t1)*r1, r1*r1]
a = [1, -2*np.cos(2 * np.pi * t2), 1]

ws = [0.11, 0.125, 0.135]
axis = np.linspace(0, 500, 500, False)

for count in np.arange(len(ws)):
    w = ws[count]
    sinusoid = np.sin(2 * np.pi * w * axis)
    response = signal.lfilter(b=b, a=a, x=sinusoid)

    print(w)
    pl.figure()
    setup_plot('Sinusoid response')
    pl.plot(sinusoid)
    pl.plot(np.real(response))

'''
Unstability is only present when the same frequency as theta2 is applied
'''
```

```
0.11
0.125
0.135
```

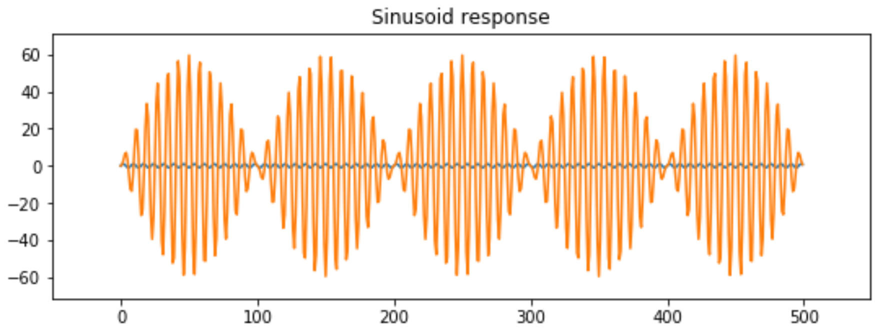
<matplotlib.figure.Figure at 0x7f3084c1f128>



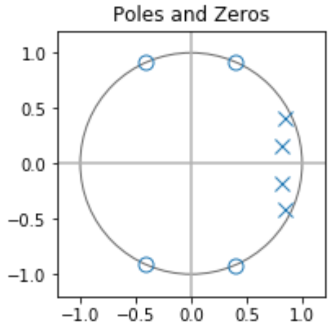
<matplotlib.figure.Figure at 0x7f3084d00128>



<matplotlib.figure.Figure at 0x7f30840da9e8>

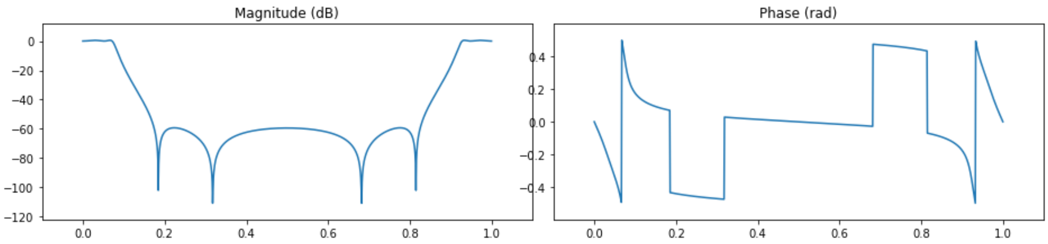


In [38]:
```
#Q 2.1
a = [0.0038, 0.0001, 0.0051, 0.0001, 0.0038]
b = [1, -3.2821, 4.2360, -2.5275, 0.5865]

zeros, poles, k = signal.tf2zpk(a, b)

setup_plot("Poles and Zeros")
zplane(zeros, poles)

'''
It looks like a LPF
'''
```



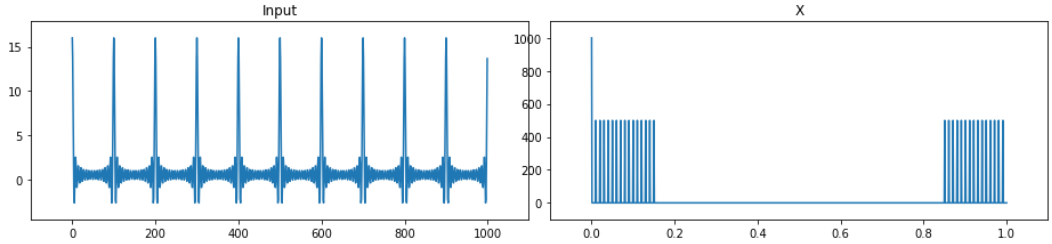
In [53]:
```
#Q 2.2
w, freq = np.array(signal.freqz(a, b))
magnit = np.abs(np.append(freq, freq[::-1]))
phase = np.angle(freq)
phase = np.append(phase, -1 * phase[::-1]) / (np.pi * 2)
w  = np.linspace(0, 1, 2*w.size, False)

pl.figure(figsize=(13,3))
pl.subplot(1, 2, 1)
pl.tight_layout()
setup_plot('Magnitude (dB)', newfig=False)
pl.plot(w, 20*np.log10(magnit))

pl.subplot(1, 2, 2)
setup_plot('Phase (rad)', newfig=False)
pl.plot(w, np.unwrap(phase))

'''
The magnitude graph pulls to -inf 4 times (basically getting rid of the sign
al)
From about fw=.1 the signal is suppressed.
'''
```

Out[53]: '\n\n'



In [50]:
```
#2.3
n = np.arange(1000)
x = np.sum(np.cos(fi*2*np.pi*n) for fi in np.linspace(0, 0.15, 16))
response = np.real(signal.lfilter(b=b, a=a, x=x))

pl.figure(figsize=(13,3))
pl.subplot(1, 2, 1)
pl.tight_layout()
setup_plot('Input', newfig=False)
pl.plot(x)
pl.subplot(1, 2, 2)
setup_plot('X', newfig=False)
pl.plot(np.linspace(0, 1, x.size), np.abs(np.fft.fft(x)))

'''
The signal is based off of sinusoids with 0<fw<0.15
'''
```

Out[50]: [<matplotlib.lines.Line2D at 0x7f3086b162e8>]



In [51]:
```
pl.figure(figsize=(13,3))
pl.subplot(1, 2, 1)
setup_plot('output', newfig=False)
pl.plot(response)
pl.subplot(1, 2, 2)
setup_plot('output FFT(x)', newfig=False)
pl.plot(np.linspace(0, 1, x.size), np.abs(np.fft.fft(response)))

'''
High frequency components have been almost completely removed
'''
```

Out[51]: [<matplotlib.lines.Line2D at 0x7f30869f62b0>]