Systems and Signals 414 Practical 5: Comb Filter

Aim: Design of a simple comb filter, and investigation of the frequency behaviour of systems and signals.

Handing in: Hand in via your `stnumber@sun.ac.za` email address. (To restate, *use your student number email address*! Do not use a special alias email address such as `johnsmith@sun.ac.za`.) Attach both your Jupyter `.ipynb` notebook file, and an `.html` copy of your output (`File` → `Download as` → `.html`). Email this to `sfstreicher+ss414@gmail.com` with subject as `prac5`. In summary, your email should look like this:

```
Recipient : sfstreicher+ss414@gmail.com
Subject   : prac5
Attachment: c:\...\filename.ipynb
Attachment: c:\...\filename.html
```

You may send your work multiple times; only the last submission will be marked. The process is automated (so we will not actually read the emails; i.e. trying to contact us using `sfstreicher+ss414@gmail.com` is futile). Make sure your `.ipynb` file returns your intended output when run from a clean slate!

Task: Do the following assignment using Jupyter. Document the task indicating your methodology, theoretical results, numerical results and discussions. Graphs should have labelled axes with the correct units indicated.

Preamble code:

In [1]:

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

pl.rcParams['figure.figsize'] = (9,2)

def setup_plot(title, y_label, x_label):
    pl.box(False)
    pl.margins(*(pl.array(pl.margins())+0.05))
    pl.title(title)
    pl.ylabel(y_label)
    pl.xlabel(x_label)
```

In [2]:

```
#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)
    zerosplot = pl.plot(zeros.real, zeros.imag,  'o', markersize=9, color='none',
                             markeredgecolor=polesplot[0].get_color(),
                             )

    #Draw overlap text
    overlap_txt = []
    def draw_overlap_text():
        for txt in overlap_txt:
            try:    txt.remove()
            except: txt.set_visible(False)
        del overlap_txt[:]

        poles_pixel_positions = ax.transData.transform(np.vstack(polesplot[0].get_data
()).T)
        zeros_pixel_positions = ax.transData.transform(np.vstack(zerosplot[0].get_data
()).T)

        for (zps_pixels, zps) in [(poles_pixel_positions, poles), (zeros_pixel_position
s, zeros)]:
            superscript = np.ones(len(zps))
            for i in range(len(zps)):
                for j in range(i+1,len(zps)):
                    if superscript[i]!=-1:
                        if np.all(np.abs(zps_pixels[i] - zps_pixels[j]) < 0.9):
                            superscript[i]+=1;
                            superscript[j]=-1;
            for i in range(len(zps)):
                if superscript[i] > 1:
                    txt = pl.text(zps[i].real, zps[i].imag,
                            r'${}^{%d}$'%superscript[i], fontsize=20
                            )
                    overlap_txt.append(txt)
    draw_overlap_text()

    #Reset when zooming
    def on_zoom_change(axes): draw_overlap_text()
```

```
    ax.callbacks.connect('xlim_changed', on_zoom_change)
    ax.callbacks.connect('ylim_changed', on_zoom_change)
```

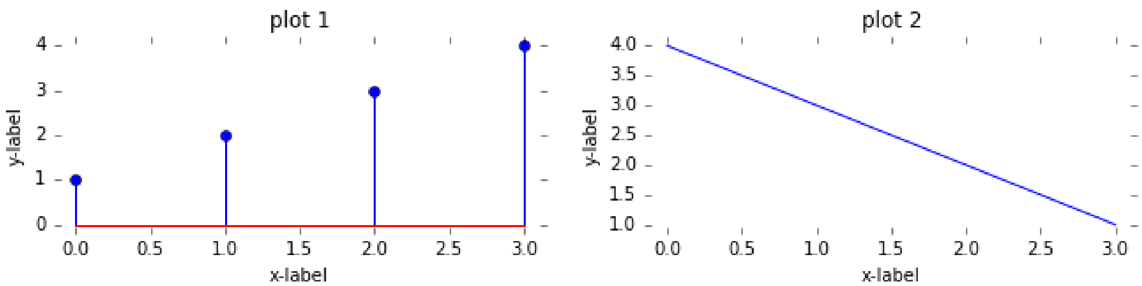
In [3]:

```
import os
import urllib
import scipy.io
from scipy.io import wavfile

#Download yesterday.wav from courses.ee.sun.ac.za and return it as a numpy array
def yesterday_wav():
    url = 'http://courses.ee.sun.ac.za/Stelsels_en_Seine_414/content/yesterday.wav'
    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
    signal_array = signal_array/np.max([np.max(signal_array), -np.min(signal_array)])
    return sample_frequency, signal_array
```

Subplot example:

In [4]:

```
#Example of subplotting
pl.figure(figsize=(11,2))
pl.subplot(1, 2, 1)
setup_plot('plot 1', 'y-label', 'x-label')
pl.stem([1,2,3,4])

pl.subplot(1, 2, 2)
setup_plot('plot 2', 'y-label', 'x-label')
pl.plot([4,3,2,1]);
```



Useful functions:

`signal.freqz(b, a, ..., whole=False)`
We use this function for plotting purposes only. It returns x-y coordinates to help illustrate the frequency response of a filter of type:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots}$$

`signal.lfilter(b, a, x, ...)`
Given a signal $x[n]$, this function will apply the input filter on the input signal, i.e. $x[n] * h[n]$, and return the output signal. Note the input filter is of type:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots}$$

`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 accordance to:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + \dots} = k \frac{(z - z_0)(z - z_1)\dots}{(z - p_0)(z - p_1)\dots}$$

`signal.zpk2tf(z, p, k)`
The reverse conversion as provided by `signal.tf2zpk`.

`np.unwrap(p, ...)`
Use this function to remedy angular wrapping such as the sawtooth effect you would see with linearly-increasing angles wrapped between $-\pi$ and π.

`zplane(z, p)`
A custom function to plot the poles and zeros parameters $\mathbf{z}$, $\mathbf{p}$ using matplotlib.

`yesterday_wav()`
A custom function to downloads the file `yesterday.wav` to the current working directory (but only if it is not already downloaded) and then read the file into a numpy array. The function returns the sample frequency f_s and the numpy array $x[n]$.

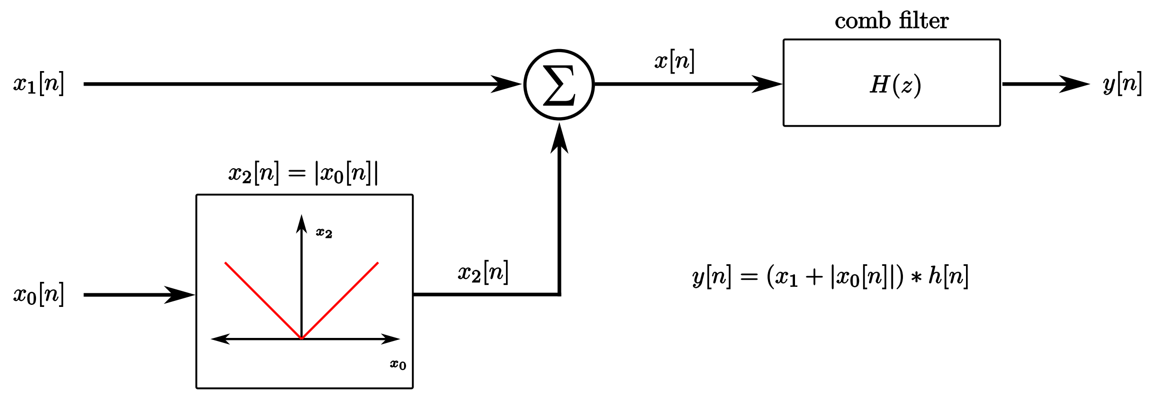
`IPython.display.Audio(x, ..., rate=fs)`
This function allows for a numpy array $x[n]$ to interpreted as an audio signal with sample frequency f_s, and be played in an embedded media player. Note that this function call must be the last line within a code block.

Question 1: All zeros comb filter

Consider the setup below with $x_1[n]$ represents a desired signal and $x_2[n]$ an interfering signal:

In [5]:

```
IPython.display.display(IPython.display.SVG('texdoc.svg'))
```



Given that signal $x_1[n]$ is an audio-snippet from *Yesterday* by *The Beetles* with sample frequency $f_s = 44100$ Hz (see the provided function `yesterday_wav`), and $x_0[n]$ is a sinusoid with frequency $f_0 = 2205$ Hz, also with sample frequency $f_s = 44100$ and with the same length as $x_1[n]$, do the following:

1. Generate the signals $x_0[n]$, $x_1[n]$, $x_2[n]$ and $x[n]$. As a single subplot, (a) plot the signal $x_1[n]$ over time $3\text{s} \leq t < 3.005\text{s}$ using `pl.plot`, and (b) plot the full sampled magnitude spectra $|X_1[k]|$ with f_ω on the x-axis (also using `pl.plot`). Make sure your plots are labeled correctly, and make use of the provided `setup_plot` function to ensure that the plots are zoomed such that the first and last sample are clearly visible (this is unfortunately necesary due to a flaw in matplotlib). Repeat this subplotting for $x_2[n]$ and $x[n]$ (including their respective magnitude spectra).
2. Use `IPython.display.Audio` to listen to the audio signals $x_1[n]$, $x_2[n]$, and $x[n]$.
3. Now design an all zeros comb filter $H(z)$ to rid $x[n]$ of the interfering signal $x_2[n]$. Provide the transfer functions and difference equations of both the prototype filter and the resulting comb filter. Plot the pole-zero patterns of the resulting comb filter by using the given `zplane` function.
4. Plot the frequency responses of both the prototype filter and the resulting comb filter with f_ω on the x-axis by making use of `signal.freqz`. Hint: By default `signal.freqz` uses a frequency range of $0 \leq f_\omega < \frac{1}{2}$. To cover the whole unit circle ($0 \leq f_\omega < 1$), set the keyword argument `whole=True`.
5. Filter $x[n]$ with the designed comb filter to obtain $y[n]$. Plot this signal in the same manner as prescribed in subquestion 1, and use `IPython.display.Audio` to listen to $y[n]$.

Question 2: Comb filter with resonant poles

1. Repeat subquestions 3-5 of Question 1, but now design a comb filter with zeros and resonant poles.
2. What effect does the resonant poles have with regards to signal $y[n]$? Can you hear an audible difference between the signals $x_1[n]$ and $y[n]$ using the filter design of Question 1? How about when using the filter design of Question 2?