Digital Signal Processing Laboratory
Department of Electrical & Computer Engineering

Digital signal processing

Lab 7

Filter design

Heraklion 2025

Dr. Konstantinos Karampidis

Introduction

Infinite Impulse Response (IIR) filters are filters whose impulse response is not finite.

The following three types of analog low-pass filters are commonly used:

- Butterworth filters
- Chebyshev filters
- Elliptic filters

Butterworth filters

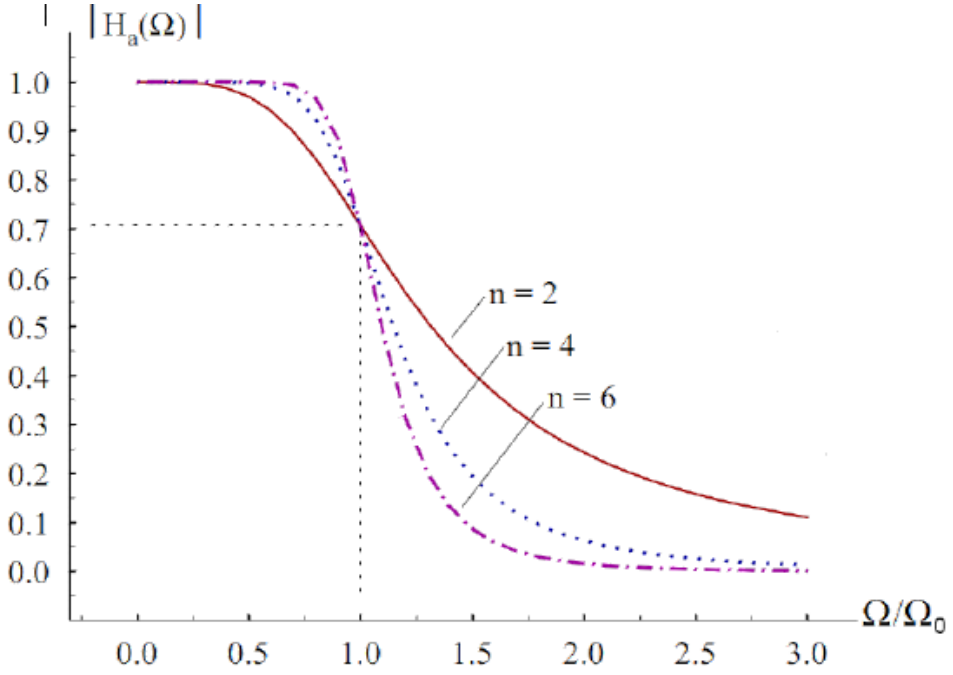
A low-pass Butterworth filter is an all-pole filter and the square of the response amplitude is given by:

$$\left|H_a(j\Omega)\right|^2 = \frac{1}{1+\left(\frac{j\Omega}{j\Omega_0}\right)^{2N}}$$

The parameter N corresponds to the order of the filter (it is the number of poles of the transfer function). The filter allows the passage of frequencies up to the cut-off frequency Ω_0 (the frequency at which the attenuation is -3db).

Butterworth filters

A Butterworth filter is completely determined by the order N and the cut-off frequency Ω_0.
Increasing N (the order of the filter) improves its performance as a low-frequency filter relative to the ideal one.



Butterworth filter construction

The development of the theoretical infrastructure for the construction of Butterworth filters is divided into two parts:

- Creation of the equations for the calculation of N and Ω_0.
- Calculation of the transfer function H(s) of the analog filter.

Butterworth filter - Implementation

Filter specifications are usually given in terms of frequency and signal attenuation in decibels (db) as follows:

- Passband: Ω_{pass}
- Stopband: Ω_{stop}
- Maximum attenuation (in db) in the passband A_{pass}
- Minimum attenuation (in db) in the stopband: A_{stop}

Once N and Ω have been determined, the transfer function can be found by spectral factorization.

Python implementation

In Python, the function that gives us the filter order (N) and cutoff frequency (Ω_0) is the ***buttord***, which we can use as follows:

```
N, Wo=signal.buttord(Wpass, Wstop, Apass, Astop, analog=False, fs=None)
```

After calculating N and Ω_o we use the ***butter*** function, which returns the coefficients of the numerator (b) and denominator (a) of the transfer function of the low-pass filter we want to implement.

```
b, a=signal.butter (N, Wo, btype='low', analog=False, output='ba', fs=None)
```

Example 1

Implement an analogue low-pass Butterworth filter that allows the passage of frequencies up to 4KHz with a maximum attenuation of 1db in this band, and from 8KHz onwards has an attenuation of at least 40db.

Solution

Ω_{pass} (passband) = 4000Hz = 2*π*4000rad/sec

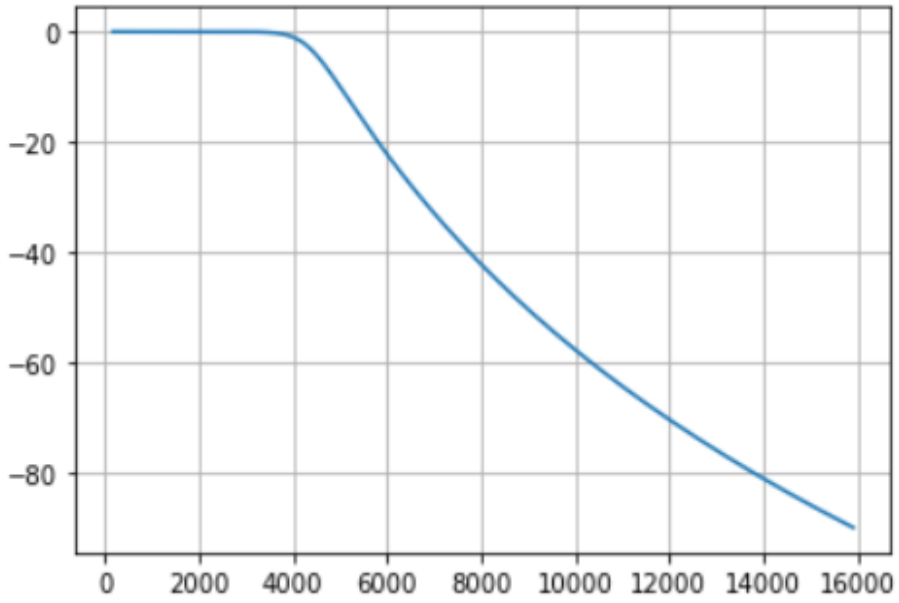
Ω_{stop} (stopband) = 8000Hz = 2*π*8000rad/sec

A_{pass} (Maximum attenuation in the pass band) = 1db

A_{stop} (Minimum attenuation in the stop band) = 40db

Example 1

```
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
Ap= 1
As = 40
Wp = 4000
Ws = 8000
omega_p = 2*np.pi*(Wp)
omega_s = 2*np.pi*(Ws)

N, Wn = signal.buttord(omega_p,
omega_s, Ap, As, analog=True)
b, a = signal.butter(N, Wn, 'low', True)
w, h = signal.freqs(b,a)

plt.plot(w/(2*np.pi), 20*np.log10(abs(h)))
plt.grid()
plt.show()
```



The attenuation in db at the frequency Ω of a Butterworth filter can be calculated in the following ways:

$$A(\Omega) = 10\log_{10}\left|H_a(j\Omega)\right|^2 db$$

$$A(\Omega) = 20\log_{10}\left|H_a(j\Omega)\right| db$$

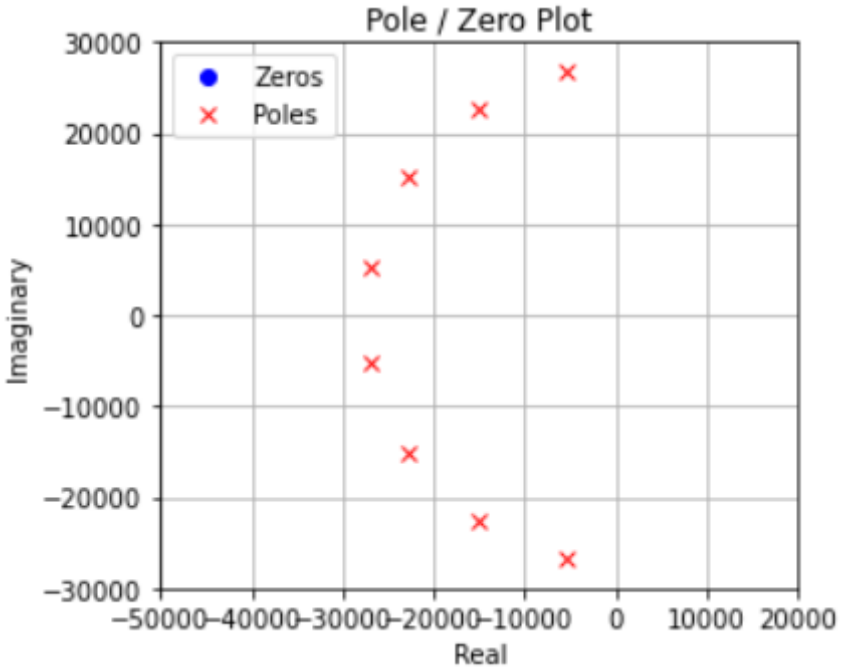
Therefore the command to design the filter can be written in the following ways:

```
plot(W/(2*pi), 10*log10(abs(H)**2))
plot(W/(2*pi), 20*log10(abs(H)));
```

How many poles do we expect to have to the filter we designed?

To graphically represent the poles of the filter we designed, we need to complete the following code:

```
z, p, c = scipy.signal.tf2zpk(b,a)
plot_z_plane(z,p,show_roc=1)
```



```
def plot_z_plane(zeros, poles, show_roc=1):
    fig, ax = plt.subplots(figsize=(7, 7), facecolor='white')
    # Maximum pole for ROC calculation and axis limits
    max_p = np.max(np.abs(poles)) if len(poles) > 0 else 0
    # ROC Plotting (Only if show_roc == 1)
    if show_roc == 1:
        roc_limit = max(2.5, max_p + 1.0)
        # Green area
        roc_area = plt.Circle((0, 0), roc_limit, color='lightgreen', alpha=0.2, label=f'ROC: $|z|
> {max_p:.2f}$', zorder=1)
         ax.add_artist(roc_area)
    # White "hole" (on top of ROC, below the X markers)
    white_center = plt.Circle((0, 0), max_p, color='white', zorder=2)
    ax.add_artist(white_center)
    # Unit Circle
    theta = np.linspace(0, 2*np.pi, 200)
    ax.plot(np.cos(theta), np.sin(theta), color='black', linestyle='--',
    linewidth=1, label='Unit Circle', zorder=3)
    # Poles and Zeros
    ax.scatter(np.real(zeros), np.imag(zeros), s=150, marker='o',
    facecolors='none', edgecolors='blue', linewidth=2,
    label='Zeros', zorder=4)
```

```
ax.scatter(np.real(poles), np.imag(poles), s=150, marker='x',
color='red', linewidth=2, label='Poles', zorder=4)
# Automatic axis adjustment
all_coords = np.concatenate(([1.2], np.abs(poles), np.abs(zeros)))
limit = np.max(all_coords) + 0.3
ax.set_xlim([-limit, limit])
ax.set_ylim([-limit, limit])
ax.axhline(0, color='black', lw=1, zorder=3)
ax.axvline(0, color='black', lw=1, zorder=3)
ax.set_aspect('equal')
ax.legend(loc='upper right', frameon=True).set_zorder(5)
ax.grid(True, linestyle=':', alpha=0.5, zorder=0)
plt.show()
```

Example 2

Implement an analog low-pass Butterworth filter that allows the passage of frequencies up to 200KHz with a maximum attenuation of 3db in this band, while from 400KHz and above it has an attenuation of at least 60db.

Solution

Ω_{pass} (passband) = 200KHz = 2*π*200*1000 rad/sec

Ω_{stop} (stopband) = 400KHz = 2*π*400*1000 rad/sec

A_{pass} (maximum attenuation in the pass band) = 3db

A_{stop} (Minimum attenuation in the stopband) = 60db

Example 2

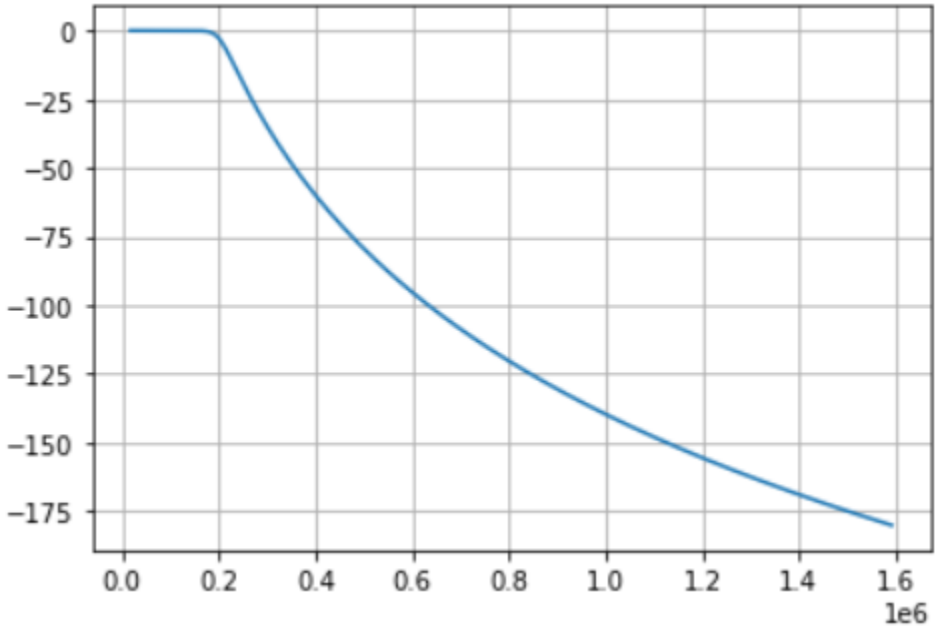
Ap= 3
As = 60

Wp = 200000
Ws = 400000
omega_p = 2*np.pi*(Wp)
omega_s = 2*np.pi*(Ws)

N, Wn = signal.buttord(omega_p, omega_s, Ap, As, analog=True)

b, a = signal.butter(N, Wn, 'low', True)
w, h = signal.freqs(b,a)

plt.plot(w/(2*np.pi), 20*np.log10(abs(h)))
plt.grid()
plt.show()



Chebyshev filters

A Chebyshev filter is a type of analogue or digital filter used to separate one frequency band from another. It is known for its steeper divergence compared to other types of filters such as Butterworth filters.

Chebyshev filters are categorized into two types: Type I and Type II.

Type I Chebyshev filters exhibit ripple in the passband, while Type II filters, also known as inverse Chebyshev filters, have ripple in the stopband.

Implementation

Similarly to buttord in chebyshev filters we have cheb1ord which we can use to use it as follows:

scipy.signal.cheb1ord(wp, ws, gpass, gstop, analog=False, fs=None) #returns order of the filter

and returns the lowest order for a Chebyshev type I filter that meets the specification.

Parameters:

wp, ws: Passband and stopband edge frequencies. For digital filters, these are in the same units as fs. By default, fs is 2 half-cycles/sample, so these are normalized from 0 to 1, where 1 is the Nyquist frequency. (wp and ws are thus in half-cycles / sample.)

Gpass: The maximum loss in the passband (dB).

Gstop: The minimum attenuation in the stopband (dB).

Analog: bool, optional: When True, return an analog filter, otherwise a digital filter is returned.

Fs: float, optional. The sampling frequency of the digital system.

Returns:

Ord:int. The lowest order for a Chebyshev type I filter that meets specs.

After calculating the order of the filter, we can use cheby1 as follows:

B,a = scipy.signal.cheby1(*N*, *rp*, *Wn*, *btype='low'*, *analog=False*, *output='ba'*, *fs=None*)

Parameters:

N:int. The order of the filter.

Rp: The maximum ripple allowed below unity gain in the passband. Specified in decibels, as a positive number.

Wn: array_like. A scalar or length-2 sequence giving the critical frequencies. For Type I filters, this is the point in the transition band at which the gain first drops below -*rp*.

btype{‘lowpass’, ‘highpass’, ‘bandpass’, ‘bandstop’}, optional

analog: bool, optional

output{‘ba’, ‘zpk’, ‘sos’}, optional

Returns:

b, a: ndarray, ndarray

Numerator (*b*) and denominator (*a*) polynomials of the IIR filter. Only returned if `output='ba'`.

z, p, kndarray, ndarray, float

Zeros, poles, and system gain of the IIR filter transfer function. Only returned if `output='zpk'`.

Example 1

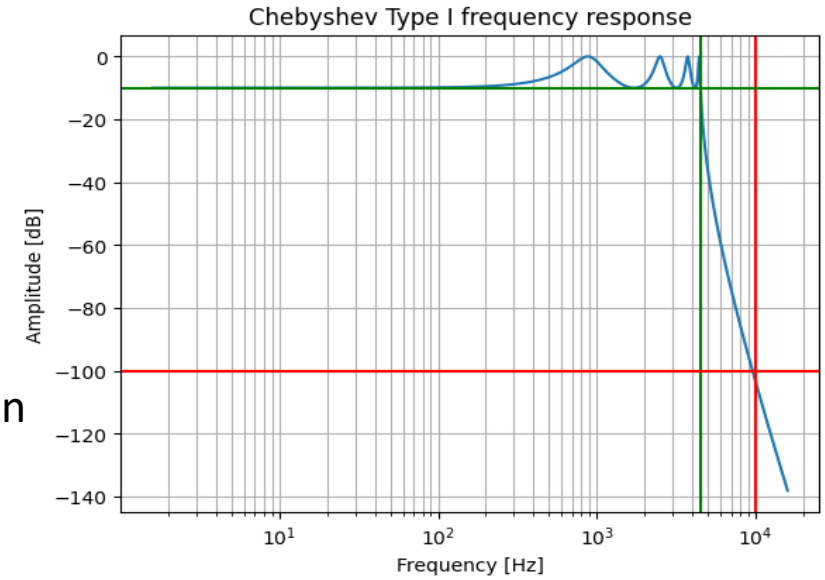
Implement an analog Chebyshev -Type I- filter that has a maximum ripple of 10 dB in the pass band up to 4500 Hz and a minimum attenuation of 100 dB in the cut-off band from 10000 Hz and above.

Solution

```
import numpy as np
from scipy.signal import cheb1ord, cheby1, freqs
import matplotlib.pyplot as plt


rp = 10  # maximum ripple in the passband (dB)
rs = 100  # minimum attenuation in the stopband (dB)
wp = 4500  # passband edge frequency (Hz)
ws = 10000  # stopband edge frequency (Hz)
```

```
# Since it's an analog filter, we directly use the angular frequencies (rad/s).
wp = 2 * np.pi * wp
ws = 2 * np.pi * ws

# Calculate the minimum order of the filter and the critical frequency
N, Wn = cheb1ord(wp, ws, rp, rs, analog=True)

# Design the Chebyshev Type I filter
b, a = cheby1(N, rp, Wn, btype='low', analog=True)

# Print filter parameters
print(f"Filter order: {N}") //Output: Filter order: 8
print(f"Critical frequency: {Wn / (2 * np.pi)} Hz")

# Frequency response
w, h = freqs(b, a, worN=np.logspace(1, 5, 1000))
```

```
# Plot the magnitude response
plt.figure()
plt.semilogx(w / (2 * np.pi), 20 * np.log10(abs(h)))
plt.title('Chebyshev Type I frequency response')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude [dB]')
plt.grid(which='both', axis='both')
plt.axvline(4500, color='green')  # passband edge
plt.axvline(10000, color='red')  # stopband edge
plt.axhline(-100, color='red')  # stopband attenuation
plt.axhline(-10, color='green')  # passband ripple
plt.show()
```



Elliptic filters

An elliptic filter is a signal processing filter with equalized ripple (equiripple) behavior in both the passband and the cutoff band.

Python implementation

Similarly to the buttord in elliptic filters we have ellipord which we can use it as follows:
scipy.signal.ellipord(wp, ws, gpass, gstop, analog=False, fs=None)

Returns the lowest-order digital or analog elliptic filter that does not lose more than gpass (in dB) in the passband and has at least gstop attenuation (in dB) in the cutoff band.

After calculating the order of the filter we can use ellip as follows:

B,a = signal.ellip(N, rp, rs, Wn, btype='low', analog=False, output='ba', fs=None)

Parameters:

N: int (The order of the filter.)

Rp:float (The maximum ripple allowed below unity gain in the passband. Specified in decibels, as a positive number.)

Rs:float (The minimum attenuation required in the stop band. Specified in decibels, as a positive number.)

Wn: array_like (A scalar or length-2 sequence giving the critical frequencies. For elliptic filters, this is the point in the transition band at which the gain first drops below -rp.)

btype{‘lowpass’, ‘highpass’, ‘bandpass’, ‘bandstop’}, optional:The type of filter. Default is ‘lowpass’.

Analog: bool, optional (When True, return an analog filter, otherwise a digital filter is returned.)

output{‘ba’, ‘zpk’, ‘sos’}, optional. (Type of output: numerator/denominator (‘ba’), pole-zero (‘zpk’), or second-order sections (‘sos’). Default is ‘ba’ for backwards compatibility, but ‘sos’ should be used for general-purpose filtering.)
Fs: float, optional. The sampling frequency of the digital system.

Returns

b, a: ndarray. Numerator (b) and denominator (a) polynomials of the IIR filter. Only returned if output='ba'.
z, p, k : ndarray. Zeros, poles, and system gain of the IIR filter transfer function. Only returned if output='zpk'.
sos: ndarray Second-order sections representation of the IIR filter. Only returned if output='sos'.

Example 1

Design an analog elliptic filter with a maximum ripple of 1 dB in the passband, a minimum attenuation of 30 dB in the cutoff band, a passband frequency of 4000 Hz and a cutoff band frequency 1 of 8000 Hz, given a sampling frequency of 20000 Hz.

```
import numpy as np
from scipy.signal import ellipord, ellip, freqs
import matplotlib.pyplot as plt

# Define the filter specifications
fp = 4000  # Passband frequency in Hz
fs = 8000  # Stopband frequency in Hz
Rp = 1     # Passband ripple in dB
Rs = 30    # Stopband attenuation in dB
```

```
# Normalize frequencies for an analog filter (frequencies are in radians per second)
Wp = 2 * np.pi * fp
Ws = 2 * np.pi * fs

# Calculate the minimum filter order and the critical frequencies
N, Wn = ellipord(Wp, Ws, Rp, Rs, analog=True)
# Design the elliptic filter
b, a = ellip(N, Rp, Rs, Wn, btype='low', analog=True)
# Frequency response
w, h = freqs(b, a, worN=20000)
# Plot the frequency response
plt.figure()
plt.semilogx(w / (2 * np.pi), 20 * np.log10(abs(h)))
plt.title('Elliptic Lowpass Filter Frequency Response')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Amplitude [dB]')
plt.grid(which='both', axis='both')
```

```
plt.axvline(fp, color='green')  # Passband edge
plt.axvline(fs, color='red')    # Stopband edge
plt.axhline(-Rs, color='red')   # Stopband attenuation
plt.axhline(-Rp, color='green') # Passband ripple
plt.show()
# Print filter order and critical frequency
print(f"Filter order: {N}")
print(f"Critical frequency: {Wn / (2 * np.pi)} Hz")
```