Digital Signal Processing Laboratory
Department of Electrical & Computer Engineering

Ψηφιακή Επεξεργασία Σήματος

Εργαστήριο 7

Σχεδιασμός Φίλτρων

Ηράκλειο 2025

Δρ. Κωνσταντίνος Καραμπίδης

Εισαγωγή

Τα φίλτρα IIR (Infinite Impulse Response) είναι φίλτρα των οποίων η κρουστική απόκριση δεν είναι πεπερασμένη.

Συνήθως χρησιμοποιούνται οι παρακάτω τρείς τύποι αναλογικών χαμηλοπερατών φίλτρων:

- Φίλτρα Butterworth
- Φίλτρα Chebyshev
- Ελλειπτικά φίλτρα

Φίλτρα Butterworth

Ένα χαμηλοπερατό φίλτρο Butterworth είναι ένα φίλτρο που έχει μόνο πόλους (all pole) και το τετράγωνο του πλάτους της απόκρισης δίνεται από τη σχέση:

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

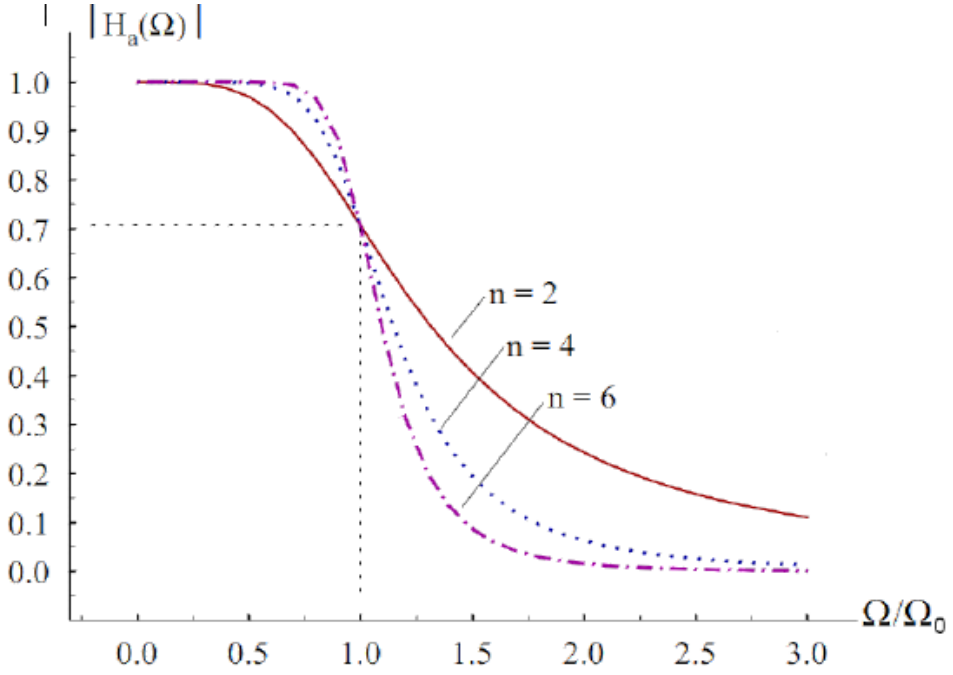
Η παράμετρος N αντιστοιχεί στη τάξη του φίλτρου (είναι ο αριθμός των πόλων της συνάρτησης μεταφοράς).

Το φίλτρο επιτρέπει τη διέλευση συχνοτήτων μέχρι τη συχνότητα αποκοπής Ω_0 (η συχνότητα στην οποία η εξασθένιση είναι -3db).

Φίλτρα Butterworth

Ένα φίλτρο Butterworth καθορίζεται πλήρως από την τάξη N και τη συχνότητα αποκοπής Ω_0.

Αυξάνοντας το N (την τάξη του φίλτρου) βελτιώνεται η απόδοση του ως φίλτρο χαμηλών συχνοτήτων σε σχέση με το ιδανικό.



Κατασκευή φίλτρων Butterworth

Η ανάπτυξη της θεωρητικής υποδομής για την κατασκευή φίλτρων Butterworth χωρίζεται σε δύο μέρη:

- Δημιουργία των εξισώσεων για τον υπολογισμό των N και Ω_0.
- Υπολογισμός της συνάρτησης μεταφοράς H(s) του αναλογικού φίλτρου.

Κατασκευή φίλτρων Butterworth

Οι προδιαγραφές του φίλτρου δίνονται συνήθως σε μορφή συχνοτήτων και εξασθένησης σήματος σε decibel (db) ως εξής:

- Ζώνη διέλευσης (passband): Ω_{pass}
- Ζώνη αποκοπής (stopband): Ω_{stop}
- Μέγιστη εξασθένιση (σε db) στη ζώνη διέλευσης :A_{pass}
- Ελάχιστη εξασθένιση (σε db) στη ζώνη αποκοπής: A_{stop}

Αφού καθοριστούν τα N και Ω η συνάρτηση μεταφοράς μπορεί να βρεθεί με φασματική παραγοντοποίηση (spectral factorization).

Υπολογισμός σε Python

Στην Python η συνάρτηση που μας δίνει την τάξη (N) του φίλτρου και τη συχνότητα αποκοπής (Ω_0) από τις προδιαγραφές είναι η **buttord**, την οποία μπορούμε να την χρησιμοποιήσουμε ως εξης:

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

Αφού υπολογίσουμε τα N και Ω_o χρησιμοποιούμε τη συνάρτηση **butter**, η οποία μας δίνει τους συντελεστές του αριθμητή (b) και του παρονομαστή (a) της συνάρτησης μεταφοράς του χαμηλοπερατού φίλτρου που θέλουμε να κατασκευάσουμε.

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

Παράδειγμα 1

Να κατασκευάσετε ένα αναλογικό χαμηλοπερατό φίλτρο Butterworth που να επιτρέπει τη διέλευση συχνοτήτων μέχρι 4KHz με μέγιστη εξασθένιση 1db στη ζώνη αυτή, ενώ από 8KHz και πάνω να έχει εξασθένιση τουλάχιστον 40db.

Οι προδιαγραφές του φίλτρου είναι:

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

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

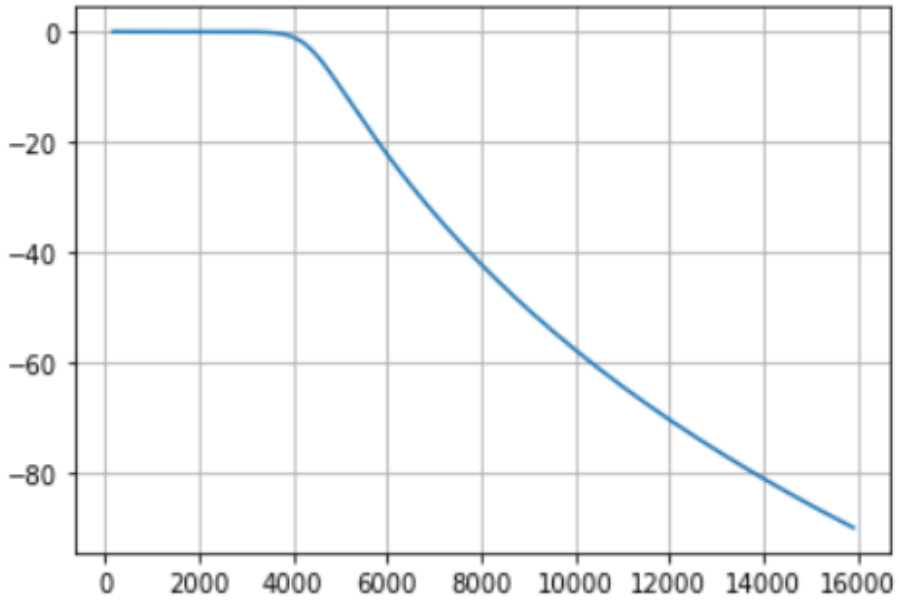
A_{pass} (Μέγιστη εξασθένιση στη ζώνη διέλευσης) = 1db

A_{stop} (Ελάχιστη εξασθένιση στη ζώνη αποκοπής) = 40db

Παράδειγμα 1

Ο κώδικας θα είναι ο εξής:

```
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()
```



Παρατήρηση Α

Η εξασθένηση σε db στη συχνότητα Ω του φίλτρου Butterworth μπορεί να υπολογιστεί με τους παρακάτω τρόπους:

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

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

Επομένως η εντολή για τον σχεδιασμό του φίλτρου μπορεί να γραφεί με τους παρακάτω τρόπους:

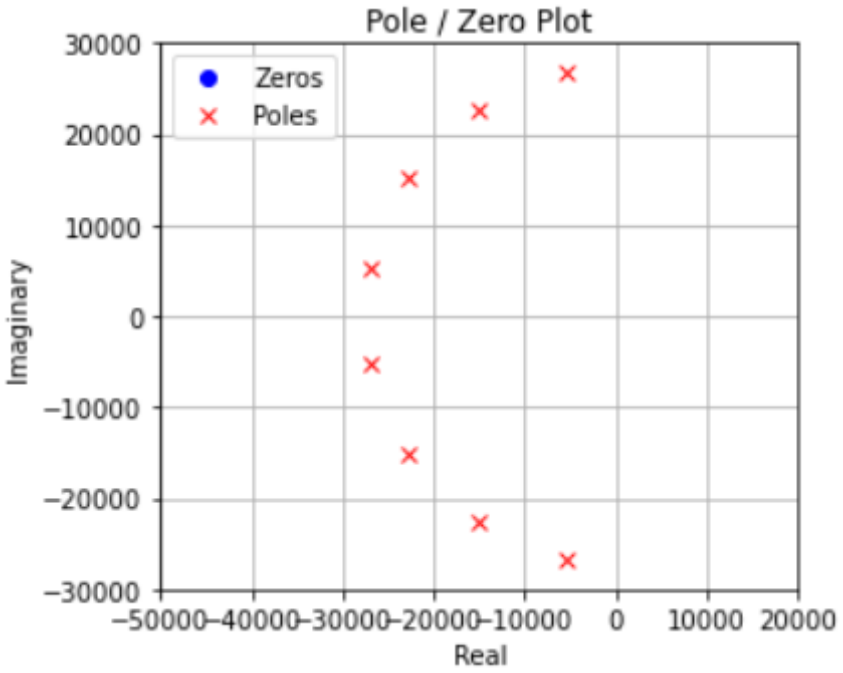
```
plot(W/(2*pi), 10*log10(abs(H)**2))
plot(W/(2*pi), 20*log10(abs(H)));
```

Παρατήρηση Β

Πόσοι περιμένουμε ότι θα είναι οι πόλοι του φίλτρου που σχεδιάσαμε;

Για να αναπαραστήσουμε γραφικά τους πόλους του φίλτρου που σχεδιάσαμε θα πρέπει να συμπληρώσουμε τον παρακάτω κώδικα:

```
z, p, c = scipy.signal.tf2zpk(b,a)
plot_z_plane(z,p,show_roc=0)
```



```
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()
```

Παράδειγμα 2

Να κατασκευάσετε ένα αναλογικό χαμηλοπερατό φίλτρο Butterworth που να επιτρέπει τη διέλευση συχνοτήτων μέχρι 200KHz με μέγιστη εξασθένιση 3db στη ζώνη αυτή, ενώ από 400KHz και πάνω να έχει εξασθένιση τουλάχιστον 60db.

Λύση

Οι προδιαγραφές του φίλτρου είναι:

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

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

A_{pass} (Μέγιστη εξασθένιση στη ζώνη διέλευσης) = 3db

A_{stop} (Ελάχιστη εξασθένιση στη ζώνη αποκοπής) = 60db

Παράδειγμα 2

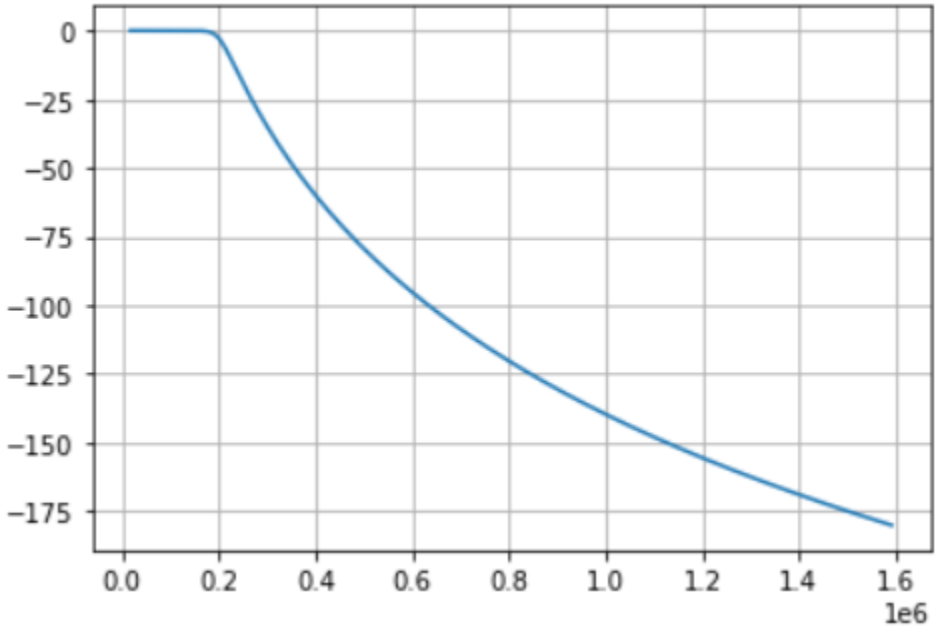
Ap= 3
As = 60

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()



Φίλτρα Chebyshev

Το φίλτρο Chebyshev είναι ένας τύπος αναλογικού ή ψηφιακού φίλτρου που χρησιμοποιείται για το διαχωρισμό μιας ζώνης συχνοτήτων από μια άλλη. Είναι γνωστό για την πιο απότομη απόκλιση σε σύγκριση με άλλους τύπους φίλτρων όπως τα φίλτρα Butterworth. Τα φίλτρα Chebyshev κατηγοριοποιούνται σε δύο τύπους: Τύπος I και Τύπος II. Τα φίλτρα Chebyshev τύπου I παρουσιάζουν κυμάτωση στη ζώνη διέλευσης, ενώ τα φίλτρα τύπου II, γνωστά και ως αντίστροφα φίλτρα Chebyshev, έχουν κυμάτωση στη ζώνη αποκοπής.

Υπολογισμός σε Python

Αντίστοιχα με την buttord στα φίλτρα chebyshev έχουμε την cheb1ord την οποία μπορούμε να την χρησιμοποιήσουμε ως εξής:

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

και μας δίνει την χαμηλότερη τάξη για ένα φίλτρο Chebyshev τύπου I που πληροί τις προδιαγραφές.

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.

Αφού υπολογίσουμε την τάξη του φίλτρου μπορούμε να χρησιμοποιήσουμε την cheby1 ως εξής:

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: float. 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, k ndarray, ndarray, float

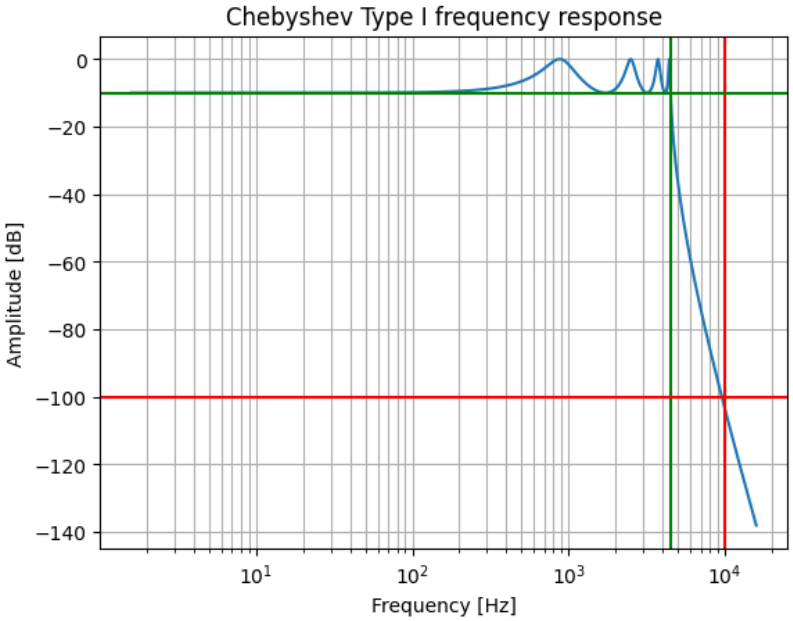
Zeros, poles, and system gain of the IIR filter transfer function. Only returned if `output='zpk'`.

Παράδειγμα 1

Κατασκευάστε ένα αναλογικό φίλτρο Chebysev (τύπου 1) το οποίο έχει μέγιστη διακύμανση 10 dB στη ζώνη διέλευσης έως τα 4500 Hz και ελάχιστη εξασθένηση 100 dB στη ζώνη αποκοπής από τα 10000 Hz και άνω.

```
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()
```



Ελλειπτικά Φίλτρα

Ένα ελλειπτικό φίλτρο είναι ένα φίλτρο επεξεργασίας σήματος με εξισορροπημένη συμπεριφορά κυμάτωσης (equiripple) τόσο στη ζώνη διέλευσης όσο και στη ζώνη αποκοπής.

Υπολογισμός σε Python

Αντίστοιχα με την buttord στα elliptic φίλτρα έχουμε την ellipord την οποία μπορούμε να την χρησιμοποιήσουμε ως εξής:

scipy.signal.ellipord(*wp*, *ws*, *gpass*, *gstop*, *analog=False*, *fs=None*)

Επιστρέφει την χαμηλότερης τάξης ψηφιακό ή αναλογικό ελλειπτικό φίλτρο που δεν χάνει περισσότερο από gpass σε dB στη ζώνη διέλευσης και έχει τουλάχιστον gstop dB εξασθένηση στη ζώνη αποκοπής.

Αφού υπολογίσουμε την τάξη του φίλτρου μπορούμε να χρησιμοποιήσουμε την ellip ως εξής:

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'.

Παράδειγμα 1

Σχεδιάστε ένα αναλογικό ελλειπτικό φίλτρο με μέγιστη απόκλιση 1 dB στη ζώνη διέλευσης, ελάχιστη εξασθένηση 30 dB στη ζώνη αποκοπής, συχνότητα ζώνης διέλευσης 4000 Hz και συχνότητα ζώνης αποκοπής 1 8000 Hz, δεδομένης συχνότητας δειγματοληψίας 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')


# Add lines for passband edge, stopband edge, passband ripple, and stopband attenuation
plt.axvline(4500, color='green', linestyle='--', label='Passband Edge (4500 Hz)')
plt.axvline(10000, color='red', linestyle='--', label='Stopband Edge (10000 Hz)')
plt.axhline(-100, color='red', linestyle='--', label='Stopband Attenuation (-100 dB)')
plt.axhline(-10, color='green', linestyle='--', label='Passband Ripple (-10 dB)')

# Add legend
plt.legend()
```

```
# Show plot
plt.show()

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