Signaler och system - Projekt

Alfred Axelsson, Elin Matsson & Lo Bergstad

Signaler och system 1TE661
Uppsala universitet

15 oktober 2025

UPPSALA
UNIVERSITET

1 Del 1 - Analysis of analog filters

Här redovisas koden för del 1 av projektet.

1.1 Upg 1.3-1.6

```python
import numpy as np
import matplotlib.pyplot as plt
import control as ct

# Constants
G = R = C = R2 = R3 = 1
GRC = G / (R * C)
R2R3 = R2 / R3

GRC = 1930  # 2000
R2R3 = 0.32 #0.3

# Numerators and denominator
num3 = np.array([-1, 0, 0])      # -s**2
num2 = np.array([-GRC, 0])       # -G/(RC)*s
num1 = np.array([-GRC**2])       # -(G/(RC))**2
den  = np.array([1, R2R3*GRC, GRC**2])

# Transfer functions
H3 = ct.tf(num3, den)
H2 = ct.tf(num2, den)
H1 = ct.tf(num1, den)

systems = [H1, H2, H3]
labels = ["H1", "H2", "H3"]

# Create big figure with 3x3 subplots
fig, axes = plt.subplots(3, 3, figsize=(15, 12))

for row, (sys, label) in enumerate(zip(systems, labels)):
    # --- Pole-Zero Plot ---
    poles = ct.poles(sys)
    zeros = ct.zeros(sys)
    ax = axes[row, 0]
    ax.scatter(np.real(poles), np.imag(poles), marker='x', color='r', label="Poles")
    ax.scatter(np.real(zeros), np.imag(zeros), marker='o', facecolors = None, edgecolors='b', label="Zeros")
    ax.set_xlabel("Real")
    ax.set_ylabel("Imag")
    ax.set_title(f"{label} Pole-Zero")
    ax.set_xlim(np.min(np.real(poles))*1.1, -0.1*np.min(np.real(poles)))
    ax.grid(True, which="both")
    ax.legend()

    # --- Impulse Response ---
```

```
    t, y = ct.impulse_response(sys)
    ax = axes[row, 1]
    ax.plot(t*1000, y, 'b')
    ax.set_xlabel("Time [ms]")
    ax.set_ylabel("Output")
    ax.set_title(f"{label} Impulse Response")
    ax.grid(True, which="both")

    # --- Bode Plot ---
    mag, phase, omega = ct.frequency_response(sys)
    ax = axes[row, 2]
    ax2 = ax.twinx()
    ax.semilogx(omega, 20*np.log10(mag), 'b')
    ax.set_xlabel(" [rad/s]")
    ax.set_ylabel("Mag [dB]", color='b')
    ax.tick_params(axis='y', labelcolor='b')
    if row == 0:
        ax.axvline(x=1885, color='k', linestyle='--', linewidth=1) #För uppgift 6b
        ax.axhline(y=10, color='k', linestyle='--', linewidth=1)   # --//--
    ax2.set_ylim(-190, 190)
    ax.grid(True, which="both", ls="--")
    ax2.semilogx(omega, phase*180/np.pi, 'r')
    ax2.set_ylabel("Phase [deg]", color='r')
    ax2.tick_params(axis='y', labelcolor='r')
    ax.set_title(f"{label} Bode")

fig.suptitle(f'R2/R3 = {R2R3}, G/RC = {GRC}', color = 'r', fontsize = 14)
plt.tight_layout(rect=[0, 0, 1, 0.99])
plt.show()
```

1.2 Upg 1.7

```
import numpy as np
import matplotlib.pyplot as plt
import control as ct
from scipy import signal

# Constants
GRC = 1930  # 2000
R2R3 = 0.32 #0.3

# Numerator and denominator
num1 = np.array([-GRC**2])       # -(G/(RC))**2
den  = np.array([1, R2R3*GRC, GRC**2])

# Transfer functions
H1 = ct.tf(num1, den)

# --- Time vector ---
f = 100               # frequency in Hz
T_end = 0.04          # simulate 40 ms (4 period of 100 Hz)
```

```python
dt = 1e-5              # timestep 10 µs
t = np.arange(0, T_end, dt)

# --- Square wave input ---
u = signal.square(2 * np.pi * f * t)    # values: -1 or +1

# --- Simulate response ---
t_out, y = ct.forced_response(H1, T=t, U=u)

# --- Plot ---
plt.figure(figsize=(10,4))
plt.plot(t*1000, u, 'black', label="Input signal x(t)")
plt.plot(t_out*1000, y, 'b', label="Output signal y1(t)")
plt.xlabel("Time [ms]")
plt.ylabel("Amplitude")
plt.legend()
plt.grid(True)
plt.show()

"""Förklaring av resultaet:
Insignalen är en fyrkantsvåg. Våråt lågpassfilter filtrera bort dom höga frekvenserna dvs de svaga termerna i Fourierserien.
"Första toppen" blir högre pga (resonans) toppen i vårt LP-filter"""
```

2 Del 2 - Sampling

Här redovisas koden för del 2 av projektet.

2.1 Upg 2.1-2.3

```python
import numpy as np
from scipy import signal
import control as ct
import matplotlib.pyplot as plt

# --- Värden ---
f_s = 16000 #Samplefrekvens (hitte på över 16000 enligt Nyqvist)
g_stop = 20*np.log10(2**(-12)) # Minsta dB reduktionen i stoppbandet, hittar på 10


deg, stop_frequency = signal.cheb1ord(8000*2*np.pi, 11000*2*np.pi, 3, -g_stop, analog = True) #Beräknar ordnignen som vi behöver på vårat chebichev filter

numerator, denominator = signal.cheby1(deg, 3, 8000*2*np.pi, analog = True)

sys = signal.lti(numerator, denominator)

n = 10**5    #Antal omega för boden
w_faster = np.linspace(1e+1, 1e+5, n)   #Flera w punkter
w, mag, phase = signal.bode(sys, w_faster)
freq_hz = w/(2*np.pi)
```

```python

w, mag, phase = signal.bode(sys)
freq_hz = w/(2*np.pi) #Gör om till vinkelfrekvens

# Rita i samma bild
fig, ax1 = plt.subplots(figsize=(10, 5))

color1 = 'tab:red'
ax1.set_xlabel("Frekvens (Hz)")
ax1.set_ylabel("Magnitud (dB)", color=color1)
ax1.semilogx(freq_hz, mag, color=color1, label="Magnitud")
ax1.tick_params(axis='y', labelcolor=color1)
ax1.grid(True, which="both", ls="--")

ax2 = ax1.twinx()  # dela x-axeln men ha egen y-axel
color2 = 'tab:blue'
ax2.set_ylabel("Fas (grader)", color=color2)
ax2.semilogx(freq_hz, phase, color=color2, label="Fas")
ax2.tick_params(axis='y', labelcolor=color2)

fig.tight_layout()
plt.title("Bode-diagram (Chebyshev I)")
plt.show()
```

2.2 Upg 2.4-2.5

```python
import numpy as np
from scipy import signal, fft
import control as ct
import matplotlib.pyplot as plt

# --- Värden ---
f_s = 16000 #Samplefrekvens (hitte på över 16000 enligt Nyqvist)
g_stop = 20*np.log10(2**(-12)) # Minsta dB reduktionen i stoppbandet, får fluktuera max 2**(-12) från noll, för om till dB

W_pass = 8000 * 2 * np.pi  # Passband (8 kHz)
W_stop = 11000 * 2 * np.pi # Stoppband (11 kHz)

deg, stop_frequency = signal.cheb1ord(W_pass, W_stop, 3, -g_stop, analog = True) #Beräknar ordnignen som vi behöver på vårat chebychev filter

numerator, denominator = signal.cheby1(deg, 3, stop_frequency, analog = True)   # Täljare och nämnare för överföringsfunktionen

sys = signal.lti(numerator, denominator)    # Skapar ett systemobjekt för filtret

# Input signals
frequency_slow = 4*1e+3 # Frekvens för riktiga signalen
frequency_fast = 11*1e+3    #Frekvens för brussignalen
f_analog = f_s*100 # Upplösning på 100ggr sample rate, modell av analog signal

```

```python
    #Tidsvektor och grejer
    periods = 10 #Antal perioder
    T_total = periods / frequency_slow
    time_vector = np.linspace(0, T_total, int(T_total * f_analog))  # Tidsvektor för den långsamma signalen
    sin_slow = np.sin(frequency_slow*2*np.pi*time_vector)   # Riktiga signalen
    sin_fast = np.sin(frequency_fast*2*np.pi*time_vector)  # Pålagt brus
    x = sin_slow + sin_fast     # Total insignal

    t_out, y, _ = signal.lsim(sys, x, time_vector) # Tidsvektor och analog utsignal

    time_vector_sample = t_out[0::int(f_analog/f_s)]    # Samplad tidsvektor, tar var 100e värde från den "analoga" tidsvektorn
    y_sample = y[0::int(f_analog/f_s)]                  # Samplad utsignal, -//-



    # Plotta insignal(er) och utsignal
    plt.plot(time_vector, x, label = 'x(t)')    # "Analog" insignal
    #plt.plot(time_vector, sin_slow, label = 'Slow Sine')    # "Analog", riktiga signalen
    #plt.plot(time_vector, sin_fast, label = 'Fast Sine')    # "Analog", brussignalen
    plt.plot(time_vector, y, label = 'y(t) (ej samplad)')   # Ej samplad utsignal
    plt.scatter(time_vector_sample, y_sample, label = 'y(t)', color = 'red')   # Samplad utsignal
    plt.legend()
    #plt.show()




    Y = fft.fft(y_sample)                   # Filtrerad signal
    #Y = fft.fft(x[0::int(f_analog/f_s)])    # Ej filtrerad signal
    N = len(Y)
    print(N)
    freqs = fft.fftfreq(N, d=1 / f_s)
    # Endast positiva frekvenser
    pos_boolean = freqs >= 0
    freqs_pos = freqs[pos_boolean]
    Y_pos = Y[pos_boolean]


    # Amplitudspektrum (enkel-sidig) och normalisering
    y_sample_max = np.max(y_sample) # Signalens max utvärde
    Y_norm = (np.abs(Y_pos)/np.max(Y_pos))*y_sample_max # Största amplituden = största amplituden för sample

    # --- Plotta amplitudspektrum ---
    plt.figure(figsize=(8, 4))
    plt.stem(freqs_pos / 1000, Y_norm) # /1000 för att få i kHz
    plt.title("Amplitude Spectrum of Filtered Output")
    plt.xlabel("Frequency [kHz]")
    plt.ylabel("Amplitude")
```

```
    plt.grid(True, which="both", ls="--")
    plt.xlim(0, f_s/2/1000)  # upp till Nyquist
    plt.show()
```