Questions 7 & 8

This script designs low pass Butterworth filters, of order varying from 4th to 8th, to meet a given specification. The script examines the stability of the filters and also the effects of 16-bit quantisation.

In [ ]:
```
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from a2_config import A2_ROOT, SAVEFIG_CONFIG
```

Filter Design

In [ ]:
```
# Define filter specifications

F_S = 30    # sampling frequency, kHz
F_C =  3    # cutoff frequency, kHz
```

In [ ]:
```
# Define utility functions for displaying frequency response and pole-zero plots
import scipy.signal as signal
from matplotlib.patches import Circle

def plot_freqz(w, h, axs=None, fname=None, color="C0", ls="-", label=None):
    """Plot frequency response and overlay filter requirements."""
    if axs is None:
        fig, axs = plt.subplots(2, sharex=True, figsize=(6, 4))
        fig.tight_layout()
    else:
        fig = None
    sns.lineplot(x=w, y=np.abs(h), ax=axs[0], c=color, ls=ls, label=label)
    sns.lineplot(x=w, y=np.angle(h), ax=axs[1], c=color, ls=ls)
    # Axis labels
    axs[0].set_ylabel("Gain")
    if label:
        axs[0].legend(loc="upper right", framealpha=1)
    axs[1].set_xlabel("Frequency (kHz)")
    axs[1].set_ylabel("Phase (rad)")
    axs[1].set_yticks([-np.pi, 0, np.pi])
    axs[1].set_yticklabels(["$-\pi$", "0", "$\pi$"])
    # Save or just show
    if fig and fname:
        fig.savefig(Path(A2_ROOT, "output", fname), **SAVEFIG_CONFIG)
    if fig:
        plt.show()

def zplane(b, a, ax=None, fname=None, color="C0", label=None):
    """
```

```python
    Plot poles and zeros from numerator and denominator of transfer function.
    """
    z, p, _ = signal.tf2zpk(b, a)
    if ax is None:
        fig, ax = plt.subplots(figsize=(5, 4))
        fig.tight_layout()
    else:
        fig = None
    ax.set_aspect("equal")
    # Axes and unit circle
    ax.add_patch(Circle((0, 0), 1, fill=False, color="k", ls=":", lw=0.5))
    ax.autoscale()
    ax.axhline(0, c="k", ls=":", lw=0.5)
    ax.axvline(0, c="k", ls=":", lw=0.5)
    # Poles and zeros
    sns.scatterplot(
        x=z.real, y=z.imag, ax=ax, marker="o", edgecolor=color, facecolor="none")
    sns.scatterplot(
        x=p.real, y=p.imag, ax=ax, marker="x", lw=2, color=color, label=label)
    # Axis labels
    ax.set_xlabel("Real")
    ax.set_ylabel("Imaginary")
    if label:
        ax.legend(loc="upper left", framealpha=1)
    if fig and fname:
        fig.savefig(Path(A2_ROOT, "output", fname), **SAVEFIG_CONFIG)
    if fig:
        plt.show()
```

In [ ]:
```python
# Design 4th order Butterworth filter
b, a = signal.butter(4, F_C, btype="low", fs=F_S)
w, h = signal.freqz(b, a, fs=F_S)
plot_freqz(w, h, fname="q7_4th_freqz.png")
zplane(b, a, fname="q7_4th_zp.png")
```

In [ ]:
```python
# Quantize the transfer function coefficients to 16 bits (15-bit mantissa)

def quantize(x, m):
    """Returns the given array quantised to m bits ([m-1]-bit mantissa)."""
    norm_factor = 2 * max(np.abs(x))
    xq = np.abs(x) / norm_factor + 0.5
    xq = np.round(xq * (1 << (m - 1))) / (1 << (m - 1))
    xq = np.sign(x) * (xq - 0.5) * norm_factor
    return xq

def tf_quantize(b, a, m=16):
    """Returns the given transfer function coefficients quantised to 16 bits."""
    return quantize(b, m), quantize(a, m)
```

In [ ]:
```python
# Quantize coefficients of 4th-order filter and observe differences
bq, aq = tf_quantize(b, a)
wq, hq = signal.freqz(bq, aq, fs=F_S)
plot_freqz(wq, hq, fname="q7_q4th_freqz.png")
zplane(bq, aq, fname="q7_q4th_zp.png")
```

Filter Testing

In [ ]:

```
import pandas as pd
import scipy.fft as fft

# Helper function for converting frequency response to dB scale
dB = lambda x: 20 * np.log10(x)

def test_filter(b, a, n_trials=10, seed=42, fname=None, figsize=(6, 4)):
    """
    Applies the filter to noise vectors and plots before and after time and
    frequency plots. Provides experimental insight into filter stability.
    """
    t = np.linspace(0, 100/3, 100, endpoint=False)
    f = fft.fftfreq(200, 1/F_S)[:100]
    ones = np.ones(100)
    agg = []
    np.random.seed(seed)
    for i in range(n_trials):
        x = np.random.rand(100) - 0.5
        z = signal.lfilter(b, a, x)
        xfft = dB(np.abs(fft.fft(x, n=200)[:100]))
        zfft = dB(np.abs(fft.fft(z, n=200)[:100]))
        agg.append(np.array([t, f, ones * i, xfft, zfft, x, z]))
    columns = ["Time", "Freq", "Trial", "FFT_In", "FFT_Out", "In", "Out"]
    agg = pd.DataFrame(np.hstack(agg).T, columns=columns)
    fig, axs = plt.subplots(2, figsize=figsize)
    fig.tight_layout()
    # Frequency plot
    sns.lineplot(
        data=agg, x="Freq", y="FFT_In", ax=axs[0], label="Input noise")
    sns.lineplot(
        data=agg, x="Freq", y="FFT_Out", ax=axs[0], label="Filtered signal")
    # Time plot
    sns.lineplot(data=agg, x="Time", y="In", ax=axs[1], label="Input noise")
    sns.lineplot(data=agg, x="Time", y="Out", ax=axs[1], label="Filtered signal")
    # Axis labels
    axs[0].set_xlabel("Frequency (kHz)")
    axs[0].set_ylabel("Gain (dB)")
    axs[0].legend(loc="upper right", framealpha=1)
    axs[1].set_xlabel("Time (ms)")
    axs[1].set_ylabel("Response")
    axs[1].legend(loc="upper right", framealpha=1)
    if fname:
        fig.savefig(Path(A2_ROOT, "output", fname), **SAVEFIG_CONFIG)
    plt.show()

# Test the filter on several zero-mean random sequences and plot the outputs
test_filter(b, a, n_trials=25, fname="q7_4th_stability.png")
test_filter(bq, aq, n_trials=25, fname="q7_q4th_stability.png")
```

Repeat with Higher Order Filters

In [ ]:
```python
def repeat_everything(filter_order, save=False):
    # Pre-define some figures and axes for compactness
    freqz_fig, freqz_axs = plt.subplots(2, figsize=(6, 4))
    freqz_fig.tight_layout()
    zp_fig, zp_ax = plt.subplots(figsize=(5, 4))
    zp_fig.tight_layout()
    # Filter design
    b, a = signal.butter(filter_order, F_C, btype="low", fs=F_S)
    w, h = signal.freqz(b, a, fs=F_S)
    plot_freqz(w, h, axs=freqz_axs, label="Full Precision")
    zplane(b, a, ax=zp_ax, label="Full Precision")
    # Quantization
    bq, aq = tf_quantize(b, a)
    wq, hq = signal.freqz(bq, aq, fs=F_S)
    plot_freqz(wq, hq, axs=freqz_axs, color="C1", ls="--", label="Quantized")
    zplane(bq, aq, ax=zp_ax, color="C1", label="Quantized")
    # Save figures on pre-defined axes
    if save:
        freqz_fname = Path(A2_ROOT, "output", f"q8_{filter_order}th_freqz.png")
        freqz_fig.savefig(freqz_fname, **SAVEFIG_CONFIG)
        zp_fname = Path(A2_ROOT, "output", f"q8_{filter_order}th_zp.png")
        zp_fig.savefig(zp_fname, **SAVEFIG_CONFIG)
    # Random testing
    test_filter(b, a, n_trials=25, fname=f"q8_{filter_order}th_stability.png",
        figsize=(9.6, 4))
    test_filter(bq, aq, n_trials=25, fname=f"q8_q{filter_order}th_stability.png",
        figsize=(9.6, 4))

repeat_everything(filter_order=5, save=True)
```

In [ ]:
```python
repeat_everything(filter_order=6, save=True)
```

In [ ]:
```python
repeat_everything(filter_order=7, save=True)
```

In [ ]:
```python
repeat_everything(filter_order=8, save=True)
```

How High Can We Go?

What is the maximum stable filter order, both before and after quantization?

In [ ]:
```python
def is_stable(b, a, quantize=False):
    """
    Uses the roots of the numerator of a transfer function to determine
    infer stability.
    """
    if quantize:
        b, a = tf_quantize(b, a)
    _, p, _ = signal.tf2zpk(b, a)
    return all(np.abs(p) < 1)

# Perform binary search for maximum stable filter order before quantization
lower = 8
upper = 50
```

```
while (upper - lower) > 1:
    centr = (lower + upper) // 2
    b, a = signal.butter(centr, F_C, btype="low", fs=F_S)
    if is_stable(b, a, quantize=False):
        print(f"Order {centr:>2}: stable")
        lower = centr
    else:
        print(f"Order {centr:>2}: unstable")
        upper = centr
max_stable_bfr = lower
print("\nMax. stable order before quantization:", max_stable_bfr)
```

In [ ]:

```
# Plot freqz, pole-zero and experimental plots
b, a = signal.butter(max_stable_bfr, F_C, btype="low", fs=F_S)
w, h = signal.freqz(b, a, fs=F_S)
plot_freqz(w, h, fname=f"q8_{max_stable_bfr}th_freqz.png")
zplane(b, a, fname=f"q8_{max_stable_bfr}th_zp.png")
test_filter(b, a, fname=f"q8_{max_stable_bfr}th_stability.png",
    figsize=(9.6, 4))
```

In [ ]:

```
b, a = signal.butter(max_stable_bfr+1, F_C, btype="low", fs=F_S)
w, h = signal.freqz(b, a, fs=F_S)
plot_freqz(w, h, fname=f"q8_{max_stable_bfr+1}th_freqz.png")
zplane(b, a, fname=f"q8_{max_stable_bfr+1}th_zp.png")
test_filter(b, a, fname=f"q8_{max_stable_bfr+1}th_stability.png",
    figsize=(9.6, 4))
```

In [ ]:

```
# Repeat search for maximum stable filter order *after* quantization
lower = 8
upper = max_stable_bfr
while (upper - lower) > 1:
    centr = (lower + upper) // 2
    b, a = signal.butter(centr, F_C, btype="low", fs=F_S)
    if is_stable(b, a, quantize=True):
        print(f"Order {centr:>2}: stable")
        lower = centr
    else:
        print(f"Order {centr:>2}: unstable")
        upper = centr
max_stable_aft = lower
print("\nMax. stable order after quantization:", max_stable_aft)
```

In [ ]:

```
# Plot freqz, pole-zero and experimental plots
b, a = signal.butter(max_stable_aft, F_C, btype="low", fs=F_S)
bq, aq = tf_quantize(b, a)
wq, hq = signal.freqz(bq, aq, fs=F_S)
plot_freqz(w, h, fname=f"q8_q{max_stable_aft}th_freqz.png")
zplane(bq, aq, fname=f"q8_q{max_stable_aft}th_zp.png")
test_filter(bq, aq, fname=f"q8_q{max_stable_aft}th_stability.png",
    figsize=(9.6, 4))
```

In [ ]:

```
b, a = signal.butter(max_stable_aft+1, F_C, btype="low", fs=F_S)
bq, aq = tf_quantize(b, a)
wq, hq = signal.freqz(bq, aq, fs=F_S)
plot_freqz(w, h, fname=f"q8_q{max_stable_aft+1}th_freqz.png")
```

```python
zplane(bq, aq, fname=f"q8_q{max_stable_aft+1}th_zp.png")
test_filter(bq, aq, fname=f"q8_q{max_stable_aft+1}th_stability.png",
    figsize=(9.6, 4))
```