Question 1

This script compares the approaches of filtering then downsampling versus applying a polyphase decimator.

In [ ]:
```
from pathlib import Path

import numpy as np
import scipy.fft as fft
import scipy.signal as signal

import matplotlib.pyplot as plt
import seaborn as sns

from a3_config import A3_ROOT, SAVEFIG_CONFIG
```

Define filter specifications:

In [ ]:
```
FS     = 40     # sampling frequency, kHz
F_PASS = 0.2    # cutoff frequency, kHz
F_STOP = 0.3    # stop band frequency, kHz
A_PASS = 3      # pass band attenuation, dB
A_STOP = 100    # stop band attenuation, dB
```

Construct Signal

In [ ]:
```
# Create signal with tones at: 50, 150, 950, 1050 Hz sampled at 40 kHz
t_signal = np.arange(0, 50, 1 / FS)
x_signal = np.sin(2 * np.pi * 0.05 * t_signal) + \
    np.sin(2 * np.pi * 0.15 * t_signal) + \
    np.sin(2 * np.pi * 0.95 * t_signal) + \
    np.sin(2 * np.pi * 1.05 * t_signal)

f_signal = fft.fftfreq(8192, 1 / FS)[:4096]
h_signal = fft.fft(x_signal, 8192)[:4096]

# Plot the signal
fig, axs = plt.subplots(1, 2, figsize=(7.5, 1.5))

sns.lineplot(x=t_signal, y=x_signal, ax=axs[0], lw=1)
sns.lineplot(x=f_signal, y=np.abs(h_signal), ax=axs[1], lw=1)

axs[0].set_xlabel("Time (ms)")
axs[1].set_xlabel("Frequency (kHz)")
axs[1].set_xlim([-0.13, 2.63])

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q1_signal.png"), **SAVEFIG_CONFIG)
```

Apply Kaiser LPF

In [ ]:
```python
ripple_p = 1 - np.power(10, -A_PASS / 20)
ripple_s = np.power(10, -A_STOP / 20)
print("Maximum pass band ripple:", ripple_p)
print("Maximum stop band ripple:", ripple_s)

A = -20 * np.log10(min(ripple_p, ripple_s))
print("Required attenuation:", A, "dB")
```

In [ ]:
```python
# Kaiser window filter length estimate
N = int(np.ceil((A - 7.95)/(14.36 * ((F_STOP - F_PASS) / FS))))
N = N + 1 if (N % 2) else N
print("Filter length estimate:", N)

beta = 0.1102 * (A - 8.7)
print("Kaiser window beta:", beta)
```

Construct a vector representing the ideal frequency response.

In [ ]:
```python
# Calculate pass band width, L
L = int(np.round(N * F_PASS / FS))
print("Bins in passband:", L)

# Construct V, with 1's in the pass band and 0's in the stop band
h_ideal = np.zeros(N//2)
h_ideal[:L] = np.ones(L)
h_ideal = np.concatenate([h_ideal, np.flip(h_ideal)])

# Impulse (time) response of ideal filter
x_ideal = fft.fftshift(fft.ifft(h_ideal))
```

In [ ]:
```python
# Construct and apply the Kaiser window
x_kaiser_lpf = x_ideal * signal.windows.kaiser(N, beta)
h_kaiser_lpf = fft.fft(x_kaiser_lpf)[:N//2]

# Time and frequency axes for plotting
t_filter = np.arange(N) / FS
f_filter = fft.fftfreq(N, 1 / FS)[:N//2]

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

# Plot windowed filter
fig, axs = plt.subplots(1, 2, figsize=(7.5, 1.5))

sns.lineplot(x=t_filter, y=x_kaiser_lpf.real, ax=axs[0], lw=1)
sns.lineplot(x=f_filter, y=dB(np.abs(h_kaiser_lpf)), ax=axs[1], lw=1)

axs[0].set_xlabel('Time (ms)')
axs[1].set_xlabel('Frequency (kHz)')
axs[1].set_ylabel('Gain (dB)')
axs[1].set_xlim([-0.13, 2.63])

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q1_filter.png"), **SAVEFIG_CONFIG)
```

In [ ]:
```python
from scipy.io import savemat

# Export the Kaiser LPF for Questions 2 & 3
```

```
fname = Path(A3_ROOT, "output", "q1_kaiser_lpf.npy")
np.save(fname, x_kaiser_lpf)

# Export as .mat file also for Question 9 (importing into MATLAB)
fname = Path(A3_ROOT, "output", "q9_kaiser_lpf.mat")
savemat(fname, dict(filter=x_kaiser_lpf))
```

In [ ]:
```
# Apply filter to signal, removing transient edge effects
x_filt = signal.convolve(x_kaiser_lpf, x_signal)[N//2:-(N//2-1)]
h_filt = fft.fft(x_filt, 8192)[:4096]

fig, axs = plt.subplots(1, 2, figsize=(7.5, 1.5))

sns.lineplot(x=t_signal, y=x_filt.real, ax=axs[0], lw=1)
sns.lineplot(x=f_signal, y=np.abs(h_filt), ax=axs[1], lw=1)

axs[0].set_xlabel("Time (ms)")
axs[1].set_xlabel("Frequency (kHz)")
axs[1].set_xlim([-0.13, 2.63])

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q1_filtered.png"), **SAVEFIG_CONFIG)
```

Maximally Downsample

In [ ]:
```
M = int(FS // (F_PASS + F_STOP))
print("Downsampling by factor of:", M)

x_dsamp = x_filt[::M]
h_dsamp = fft.fft(x_dsamp, 8192)[:4096]

t_dsamp = np.arange(0, 50, M / FS)
f_dsamp = fft.fftfreq(8192, M / FS)[:4096] * 1000 # show in Hz rather than kHz

fig, axs = plt.subplots(1, 2, figsize=(7.5, 1.5))

sns.lineplot(x=t_dsamp, y=x_dsamp.real, ax=axs[0], lw=1)
sns.lineplot(x=f_dsamp, y=np.abs(h_dsamp), ax=axs[1], lw=1)

axs[0].set_xlabel("Time (ms)")
axs[1].set_xlabel("Frequency (Hz)")

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q1_dsamp.png"), **SAVEFIG_CONFIG)
```

Polyphase Downsample

In [ ]:
```
# Reshape filter coefficients into matrix, zero padded to muliple of M
k = M - (N % M)
polyfilt = np.concatenate([x_kaiser_lpf, np.zeros(k)])
polyfilt = polyfilt.reshape(int((N + k) / M), M).T  # reshape row-major then T
polyfilt = np.flipud(polyfilt)                      # vertical flip

# Reshape signal to equal vertical dimension
x_polysig = x_signal.reshape(int((len(x_signal) + k) / M), M).T
```

In [ ]:
```
# Accumulate results into output array, which becomes the filtered signal
x_polyfilt = np.zeros(int((len(x_signal) + N + k) / M - 1), dtype=np.complex128)
for i in range(M):
    x_polyfilt += signal.convolve(polyfilt[i], x_polysig[i])

# As before, remove transient edge effects
N_polyfilt = polyfilt.shape[1]
x_polyfilt = x_polyfilt[N_polyfilt//2:-(N_polyfilt-1)//2]

# Calculate transform for plotting
h_polyfilt = fft.fft(x_polyfilt, 8192)[:4096]

# Construct time and frequency axes for plotting
t_polyfilt = np.arange(0, 50, 50 / len(x_polyfilt))
f_polyfilt = fft.fftfreq(8192, 50 / len(x_polyfilt))[:4096] * 1000 # kHz -> Hz

# Plot the polyphase downsampled signal
fig, axs = plt.subplots(1, 2, figsize=(7.5, 1.5))

sns.lineplot(x=t_polyfilt, y=x_polyfilt.real, ax=axs[0], lw=1)
sns.lineplot(x=f_polyfilt, y=np.abs(h_polyfilt), ax=axs[1], lw=1)

axs[0].set_xlabel("Time (ms)")
axs[1].set_xlabel("Frequency (Hz)")

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q1_polydecimate.png"), **SAVEFIG_CONFIG)
```

In [ ]:
```
# Export the polyphase downsampled signal for Questions 2 & 3
fname = Path(A3_ROOT, "output", "q1_signal_out.npy")
np.save(fname, np.stack([t_polyfilt, x_polyfilt]))
```

Performance Comparison

In [ ]:
```
import time
from tqdm import trange

N_TRIALS = 10000
msfmt = lambda t: f'{(t * 1000 / N_TRIALS):.5f}'

time_start = time.time()
for _ in trange(N_TRIALS):
    x_filt = signal.convolve(x_kaiser_lpf, x_signal)[N//2:-(N//2-1)]
    x_dsamp = x_filt[::M]
time_elapsed = time.time() - time_start
print(f"Filter then downsample ({N_TRIALS} trials): {msfmt(time_elapsed)} ms")
```

In [ ]:
```
time_start = time.time()
for _ in trange(N_TRIALS):
    x_polyfilt = np.zeros(
        int((len(x_signal) + N + k) / M - 1), dtype=np.complex128)
    for i in range(M):
        x_polyfilt += signal.convolve(polyfilt[i], x_polysig[i])
time_elapsed = time.time() - time_start
print(f"Polyphase decimator ({N_TRIALS} trials): {msfmt(time_elapsed)} ms")
```