# Question 2

This script compares the approaches of upsampling then filtering versus applying a polyphase interpolator.

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
```

## Upsample (Zero-Pack)

In [ ]:
```
# Import polyphase downsampled signal from Question 1
t_signal, x_signal = np.load(Path(A3_ROOT, "output", "q1_signal_out.npy"))

# Import Kaiser LPF from Question 1
x_kaiser_lpf = np.load(Path(A3_ROOT, "output", "q1_kaiser_lpf.npy"))
```

In [ ]:
```
L = 80      # upsampling rate, equal to M from Question 1
FS = 0.5    # sampling frequency, kHz

x_zpack = np.concatenate([[x]+[0]*(L-1) for x in x_signal])
h_zpack = fft.fft(x_zpack, 8192)[:4096]

t_zpack = np.arange(0, 50, 1 / (FS * L))
f_zpack = fft.fftfreq(8192, 1 / (FS * L))[:4096]

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

sns.lineplot(x=t_zpack, y=x_zpack.real, ax=axs[0], lw=1)
sns.lineplot(x=f_zpack, y=np.abs(h_zpack), 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", "q2_zpack.png"), **SAVEFIG_CONFIG)
```

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

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

sns.lineplot(x=t_zpack, y=x_filt.real, ax=axs[0], lw=1)
sns.lineplot(x=f_zpack, 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", "q2_usamp.png"), **SAVEFIG_CONFIG)
```

## Polyphase Upsample

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

In [ ]:
```
# Concatenate results into output array, which becomes the filtered signal
x_polyfilt = []
for i in range(L):
    x_polyfilt.append(signal.convolve(polyfilt[i], x_signal))
x_polyfilt = np.array(x_polyfilt).flatten("F")

# Remove transient edge effects
x_polyfilt = x_polyfilt[(N+k-L)//2:-(N+k-L)//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]

# 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 (kHz)")
axs[1].set_xlim([-0.13, 2.63])

fig.tight_layout()
fig.savefig(Path(A3_ROOT, "output", "q2_polyinterpolate.png"), **SAVEFIG_CONFIG)
```

## 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_zpack = np.concatenate([[x]+[0]*(L-1) for x in x_signal])
    x_filt = signal.convolve(x_kaiser_lpf, x_zpack)
```

```
time_elapsed = time.time() - time_start
print(f"Upsample then filter ({N_TRIALS} trials): {msfmt(time_elapsed)} ms")
```

In [ ]:
```
time_start = time.time()
for _ in trange(N_TRIALS):
    x_polyfilt = []
    for i in range(L):
        x_polyfilt.append(signal.convolve(polyfilt[i], x_signal))
    x_polyfilt = np.array(x_polyfilt).flatten("F")
time_elapsed = time.time() - time_start
print(f"Polyphase interpolator ({N_TRIALS} trials): {msfmt(time_elapsed)} ms")
```