Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Signal and Information Processing Laboratory
Institut für Signal- und Informationsverarbeitung

Fachpraktikum Signalverarbeitung

# SV2: Digital Filters

## 1 Introduction

A *discrete-time* ("digital") *signal* $x[k]$ is defined only at time points $k \in \mathbb{Z}$. For instance, an audio file on the computer is a discrete-time signal. Often, such a signal is used to describe a continuous-time signal $x(t)$ $(z \in \mathbb{R})$. In this case, a *sampling rate* $f_s$ or a *sampling interval* $T_s = 1/f_s$ is defined.

The *z-transform* of a discrete-time signal $x[k]$ is defined for $z \in \mathbb{C}$ as

$$X(z) = \sum_{k=-\infty}^{\infty} x[k] z^{-k}. \tag{1}$$

The *spectrum* (the frequency response) of $x[k]$ is $X(e^{i\Omega})$, i.e., the $z$-transform evaluated on the unit circle, where the *digital frequency* $\Omega$ corresponds to the argument of $z$ and is usually considered from $-\pi$ to $\pi$, being $2\pi$-periodic. The corresponding continuous-time frequency is $f = \Omega f_s / 2\pi$.

In these experiments, discrete-time signals are manipulated with linear filters and the sampling rate is converted.

## 2 Linear Filters

A *linear filter* processes an input signal $x[k]$ to an output signal $y[k]$, by forming a linear combination of past input values and output values:

$$\begin{aligned} y[k] = {} & a_0 x[k] + a_1 x[k-1] + a_2 x[k-2] + \cdots + a_N x[k-N] \\ & + b_1 y[k-1] + b_2 y[k-2] + \cdots + b_M y[k-M]. \end{aligned} \tag{2}$$

An *infinite impulse response* (IIR) *filter* has at least one coefficient $b_m \neq 0$, whereas a *finite impulse response* (FIR) *filter* has all coefficients $b_m = 0$. The *order* of a filter is defined as the largest of the numbers $N$ and $M$. The FIR filter can be defined by the *impulse response*[1] $h[k]$, since its impulse response has just the coefficient $a_n = h[n]$. Note: this impulse response has duration $L = N + 1$.

For IIR and FIR filters, the $z$-transformed impulse response $H(z)$ is given by *the transfer function* and has the general form

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}{1 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_M z^{-M}}. \tag{3}$$

In these experiments, we often consider the *amplitude response* $\left|H(e^{i\Omega})\right|$ of a filter.

[1]The impulse response is the output signal for the input signal $x[k] = (1, 0, 0, \ldots)$.

The coefficients $a_n$ and $b_n$ can thus be used directly in the filtering and at the same time describe through $H(z)$ the transfer function and thus the *frequency response* $H(e^{i\Omega})$ of the filter.

## 3 Filter Design

### 3.1 FIR Filters

We consider the design of frequency selective FIR filters. We present the method for lowpass filters. The design of other frequency selective FIR filters is similar.

An ideal lowpass filter is defined by its frequency response

$$H(e^{i\Omega}) = \begin{cases} 1, & |\Omega| < \Omega_c, \\ 0, & \Omega_c \leq |\Omega| \leq \pi. \end{cases} \tag{4}$$

With the inversion formation (cf. (1.183) in the DSSP lecture notes), we obtain

$$h[k] = \frac{\sin(\Omega_c k)}{\pi k} = \frac{\Omega_c}{\pi} \operatorname{sinc}\left(\frac{\Omega_c}{\pi} k\right). \tag{5}$$

The resulting filter is neither stable nor causal. An obvious attempt to turn it into a practical filter is to truncate it, by multiplying (5) with a *window function*

$$w[k] = \begin{cases} 1 & |k| \leq L/2 \\ 0 & |k| > L/2 \end{cases} \tag{6}$$

This results in a filter that no longer follows the ideal frequency response of (4) but rather presents a smooth transition between stopband and passband as well as unevenness in the frequency response in these ranges.

Many other window functions have been investigated and used: the Hann window, the Hamming window, the Kaiser window, the Blackman window, the Lanczos window, etc., all of which are better than (6).

To make the so modified impulse response causal, it is shifted by $L/2$ time steps: $a_k = h[k - L/2]w[k - L/2]$. The resulting delay of $L/2$ time steps is accepted.

### 3.2 IIR Filters

Discrete-time IIR filters are often designed in two steps:

1. Design a suitable continuous-time filter.
2. Transform the continuous-time filter into a discrete-time filter.

The classical continuous-time filters include the Butterworth filter, the Chebyshev filter, the elliptic filter, and the Bessel filter.

## 4 Conversion of the Sampling Rate

A (discrete-time) signal $x[k]$ with sampling rate $f_{s1} = 1/T_{s1}$ is to be converted into a signal $y[k]$ with sampling rate $f_{s2} = 1/T_{s1}$ ($\neq f_1$). This corresponds to a resampling of the (continuous-time) signal $x(t)$ on the sampling grid $kT_{s2}$.

For a rational factor $\frac{f_{s2}}{f_{s1}} = \frac{n}{m}$ ($n, m \in \mathbb{N}$) the system of Figure 1 can be applied. It consists of an interpolation block, a low-pass filter and a decimation block.

$$\begin{tikzcd} {} \arrow[r] & \boxed{\uparrow n} \arrow[r] & \boxed{h[.]} \arrow[r] & \boxed{\downarrow m} \arrow[r] & {} \end{tikzcd}$$

Figure 1: Sampling-rate conversion system

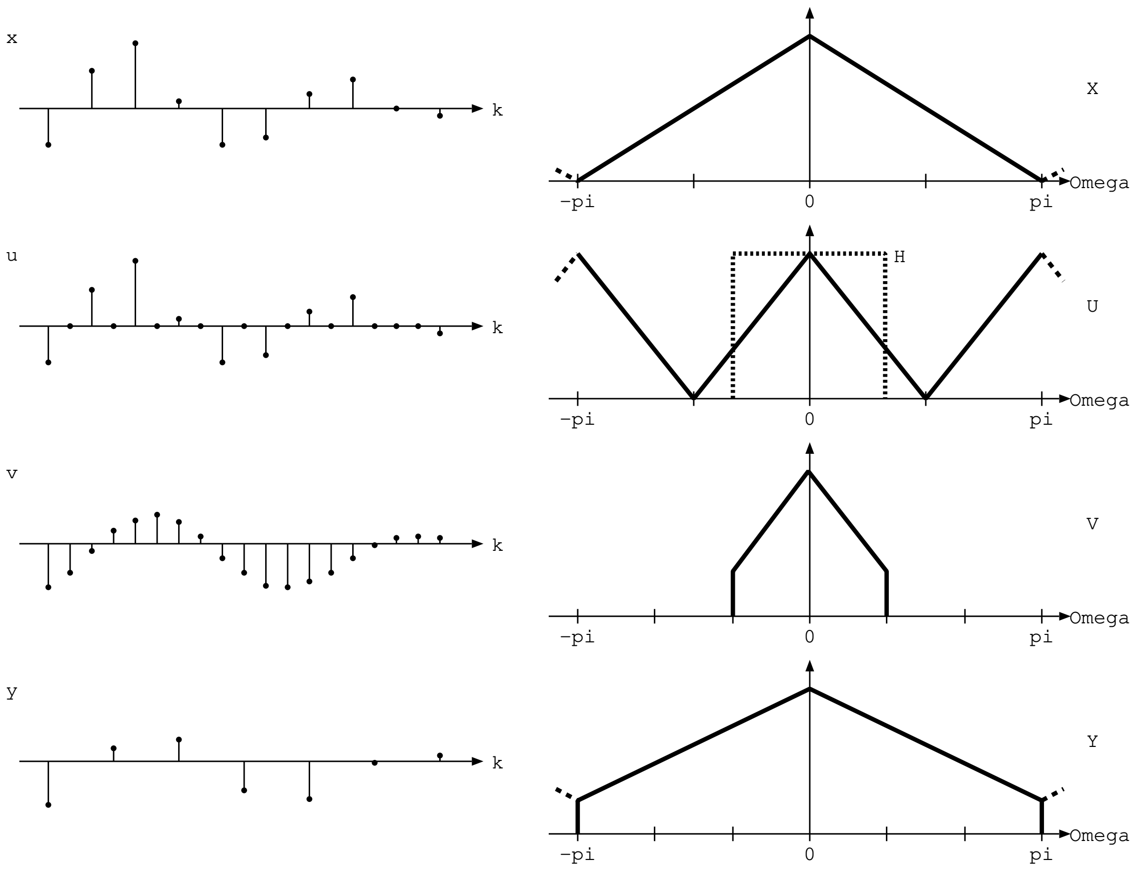

Figure 2: Signal and amplitude response during a sampling rate conversion with $n = 2$ and $m = 3$

$\boxed{\uparrow n}$ In the *interpolation block*, $n-1$ zeros are inserted between two values $x[k]$ and $x[k+1]$ (for each $k$). The sampling rate is thus increased to $f_{s(up)} = nf_{s1}$.

$$u[k] = \begin{cases} x[k/n] & k/n \in \mathbb{Z} \\ 0 & k/n \notin \mathbb{Z} \end{cases} \tag{7}$$

In the frequency domain, this compresses the spectrum along the frequency axis by a factor $n$ (from $[-\pi, \pi]$ to $[-\pi/n, \pi/n]$) with $n-1$ "copies". As an example for $n = 2$ compare the amplitude responses $|X(e^{i\Omega})|$ and $|U(e^{i\Omega})|$ in Figure 2.

$\boxed{\downarrow m}$ The *decimation block* picks every $m$-th value from $v[k]$ and ignores the values in between. The sampling rate is thus lowered to $f_{s2} = f_{s(up)}/m$.

$$y[k] = v[mk] \tag{8}$$

This results in a stretching by a factor $m$ in the frequency domain (from $[-\pi/m, \pi/m]$ to $[-\pi, \pi]$). Compare $|V(e^{i\Omega})|$ and $|Y(e^{i\Omega})|$ in Figure 2 for $m = 3$.

$\boxed{h[k]}$ The *low pass filter* performs two tasks simultaneously:

- As *interpolation filter* it suppresses in $U(e^{i\Omega})$ the unwanted copies. Thus, the original spectrum is theoretically completely preserved.
- In $U(e^{i\Omega})$ the signal must be limited to the frequency range $-\pi/m \ldots \pi/m$. Otherwise, during decimation, signal components outside this interval will be stretched beyond $\pi$ (below $-\pi$). Because every digital spectrum is periodic, these parts are "folded back" into the spectrum, so to speak. This effect is called *aliasing* and the filter $h[k]$ therefore also acts as an *antialiasing filter*. As shown in Figure 2 it is possible that the original shape of the spectrum is not preserved.

For the combined interpolation and antialiasing filter $h[k]$ to meet both requirements, the cutoff frequency is set to

$$\Omega_c = \min\{\pi/n, \pi/m\}. \tag{9}$$

## References

H.-A. Loeliger, *Discrete-time and Statistical Signal Processing*, Chapters 1 and 2, 2021.
J. G. Proakis und D. G. Manolakis, *Digital Signal Processing*, Prentice Hall, 1996.

## 5 Experiments

Inputs on the system command line (shell) are marked with ">" whereas input on the Matlab command line is marked with ">>".

Copy `/home/isistaff/glf/fachprak_isi/SV2` in your home directory:

```
> cp -irL /home/isistaff/glf/fachprak_isi/SV2 ./
> cd SV2/matlab
> matlab &
```

The experiments consist of completing Matlab script files in the folder `SV3/matlab` and playing with the "Filter Analyse- und Designtool" (`fdatool`) in Matlab. For illustration, we use audio signals[2] which are all stored as .wav files in the folder `SV2/matlab/signals`.

### 5.1 Linear Filters

1. Listen to the noise signal `sig_original.wav`. It lasts one second and is sampled at 44.1 kHz.
   ```
   > play signals/sig_original.wav
   ```
2. In the file `fir_window_lp.m`, (5) is used to obtain the impulse response of a FIR lowpass filter. Generate the impulse response `h` and the impulse response `h_w` multiplied by a Blackman window function for a freely selected filter length and cutoff frequency
   ```
   >> f_s = 44100; L = 43; f_c = 8000; window = blackman(L);
   >> h = fir_window_lp(f_s, L, f_c);
   >> h_w = fir_window_lp(f_s, L, f_c, window);
   ```

[2]The best way to output the audio files is to write a short Matlab function (`play.m`):

```
function play(audiofile):
  [y, fs] = audioread(audiofile);
  sound(y, fs)
  end
```

Because these are FIR filters, in equation (2) all coefficients $b_m = 0$ and the Matlab vector `h` and `h_w` contain coefficients $a_n$.[3]

3. Use Matlab's own "Filter Visualization Tool" `fvtool` to display the amplitude response of both filters:[4].
   ```
   >> fvtool(h, 1, h_w, 1);
   ```
   $|H_w(e^{i\Omega})|$ (red curve) differs from $|H(e^{i\Omega})|$ (blue curve) in the following features: the transition between passband and stopband is less sharp, but the stopband attenuation is larger and the ripple in the passband is smaller.

4. Look at the impulse response and the step response. Click on the corresponding toolbar buttons in the `fvtool` window. Here you can see that $h_w[k]$ (red) has a smoother transient response than $h[k]$ (blue).

The method of Section 3 can also be used to design high-pass, band-pass and band-stop filters. The corresponding functions are implemented in `fir_window_hp.m`, `fir_window_bp.m` and `fir_window_bs.m`.

5. In `fir_window_bs.m`, complete the implementation with the following equation. This can be derived analogously to equation (5).

$$h[k] = \mathrm{sinc}(k) - \frac{\Omega_{c1}}{\pi}\,\mathrm{sinc}\left(\frac{\Omega_{c1}}{\pi}k\right) - \frac{\Omega_{c2}}{\pi}\,\mathrm{sinc}\left(\frac{\Omega_{c2}}{\pi}k\right) \tag{10}$$

6. Calculate the impulse response of a bandstop filter (e.g.: `>> f_c = [5000 15000];`) analog to the lowpass filter example in point 2 and plot the amplitude response with the `fvtool`.

7. The Matlab function `filter` computes equation (2), so it filters a signal. Look at `filter_signal.m` and run it with
   ```
   >> filter_signal()
   ```
   to get four filtered versions of `sig_original.wav`.

8. Listen to the generated wave files. If you want, you can modify `filter_signal.m` so that all filters use a rectangular window (delete the argument `window` in the function calls `fir_window_...`) and the signals are stored under a different name. Can you hear a difference?

In the file `noisy.wav` a useful signal is perturbed by a narrowband noise signal (2.2 kHz to 2.8 kHz). We try to remove the latter with a bandstop filter, but of course, the useful signal is changed as well.

9. Listen to the useful signal `sound.wav`, the noise signal `noise.wav` and the addition of both `noisy.wav`.
   ```
   > play signals/sound.wav signals/noise.wav signals/noisy.wav
   ```

10. Complete the corresponding marked section in `denoise_signal.m` so that a bandstop filter is applied using `fir_window_bs`.

[3] In the Matlab documentation, the coefficients in the numerator of equation (3) are called $b_n$ and those in the denominator $a_n$.

[4] In the `fvtool` window, the sampling rate can be set in the menu `Analysis -> Sampling Frequency`. This will cause correct labeling of the frequency axis

| Parameters | Values |
|---|---|
| Sampling frequency | 44.1 kHz |
| Lower passband | 0 kHz to 1.5 kHz |
| Stop band | 2.2 kHz to 2.8 kHz |
| Upper passband | 3.5 kHz to 22.05 kHz |
| Maximum ripple in passband | 0.8 dB |
| Stop band attenuation | 80 dB |

Table 1: Bandstop filters specifications

11. Now execute the script:
    `> denoise_signal();`
    (possibly for different filter lengths `L`) and listen to the result:
    `> play signals/denoised_fir_window.wav`

Matlab supports many other filter design methods, including IIR filters. Most of them can be done with the `fdatool`. `fvtool` is integrated with `fdatool`.

We now use the exact specification in Table 1 to design an elliptic IIR filter and an equiripple FIR filter. In particular, we let Matlab compute the required filter order.

12. Start the tool with `fdatool`.

13. Select "Response Type" > "Bandstop" and "Design Method" > "FIR Equiripple"

14. Enter the parameters from Table 1 into the "Frequency Specifications" and "Magnitude Specifications" fields.[5]

15. Click on the "Design Filter" button. The calculated filter order is displayed in the upper left corner.

16. Change the "Design Method" to "IIR Elliptic" and click again on "Design Filter". Note the filter order.

At this point, you could export a filter.[6]. Likewise, it is possible to generate a .m file to view the Matlab functions called. The design of these two filters is also implemented in `denoise_signal.m`.

17. Close the `fdatool` window. The three filters are designed in `denoise_signal.m` and loaded into the `fvtool` with: `>> denoise_signal(1);` The `fvtool` window shows three plots.

    - Blue: FIR filter by window method with self-selected filter order.
    - Yellow: FIR equiripple filter.
    - Red: IIR elliptic filter.

18. Listen to the resulting files.

    - `denoised_fir_window.wav`
    - `denoised_fir_equirip.wav`

[5]The lower end of the lower passband and the upper end of the upper passband do not need to be specified, as these values are automatically 0 and the Nyquist frequency $f_s/2$.

[6]If you want to do this, first select the menu "Edit > Convert to Single Section" for the elliptic filter to get the filter coefficients according to equation (3)

  – `denoised_iir_elliptic.wav`

The following observations and remarks can be made:

- The useful signal has undergone audible changes due to filtering. This signal separation is not optimal but easy to perform.
- The IIR filter requires a much smaller filter order than the FIR filters to meet the specification in Table 1.
- The phase response of both FIR filters is linear. Thus, the group delay is constant for these two filters. However, this property is hardly audible.

## 5.2 Conversion of Sampling Rate with Rational Factor

19. Listen to `convert.wav`. It contains a speech signal sampled at $f_{s1} = 16\,\mathrm{kHz}$. We now want to convert the sampling rate to $f_{s2} = 6\,\mathrm{kHz}$.

20. Calculate the corresponding truncated ratio $n/m$ and perform the conversion using the script `resample_signal.m`:
```
>> resample_signal('./signals/convert.wav', n, m);
```
Listen to the result `convert6000_wrong.wav`.

The noise you hear is due to interpolation and decimation artifacts. Since the conversion works without interpolation and antialiasing filters, the spectrum is distorted as described in Section 4. This is of course audible in the time signal.

21. Calculate the cutoff frequency ($f_c = \Omega_c f_{s(up)}/2\pi$) of the required lowpass filter for the example of point 20. (Note: the filter operates at a sampling rate of $f_{s(up)} = n f_{s1}$.)

22. Implement the lowpass filter in `resample_signal.m` at the given position in the file. Use either `fir_window_lp.m` with self-selected order or implement an equiripple FIR filter (or an elliptic IIR filter) as in `denoise_signal.m`.

23. Now perform the conversion again. The additional fourth argument (`1`) turns filtering on:
```
>> resample_signal('./signals/convert.wav', n, m, 1);
```
Listen to the result `convert6000_right.wav`.[7]

The speech signal has changed due to the lowpass filtering, but the aliasing noise is hardly audible anymore. The lower the order of the filter is selected, the more audible this lowpass filtering is. It is not completely inaudible even with a high filter order, since the signal has frequency components above 6 kHz.

If the sampling frequency is increased, e.g., to 24 kHz, then – with a sufficiently high filter order – the speech signal will hardly experience a perceptible change. Try it out if you like.

Finally, we look at the amplitude responses for the sample conversion of the signal `triang.wav`. Since this is a stationary signal, we can estimate the spectrum with the discrete Fourier transform.

[7]Remaining noise can be explained by the non-ideal character of the filter, quantization effects, or re-sampling rate conversion in the operating system audio output.

24. Load the signal and plot the amplitude response:[8]
```
>> [sig, f_s] = audioread('./signals/triang.wav');
>> spec = fftshift(fft(sig));
>> mag_spec = abs(spec); f = linspace(-f_s/2, f_s/2, length(sig));
>> plot(f, mag_spec);
```

25. In `resample_signal.m` the following amplitude responses are plotted for positive frequencies $f$ if an additional fifth argument (`1`) is passed:
    - Amplitude of the original signal $|X(f)|$
    - Amplitudes of the interpolated signal $|U(f)|$ and filter $|H(f)|$
    - Amplitude of the decimated signal $|Y(f)|$

26. Now execute `resample_signal.m` with the values for $n$ and $m$ calculated in point 20, first without filter:
```
>> resample_signal('./signals/triang.wav', n, m, 0, 1);
```
    Look at the plots and now switch the filter to them:
```
>> resample_signal('./signals/triang.wav', n, m, 1, 1);
```
    Listen to the corresponding signals.

27. Now select $n = 3$ and $m = 2$ to increase the sampling rate to 24 kHz. Run `resample_signal` again (you may have to adjust the cutoff frequency $f_c$ of the filter), with and without a filter. Look at the spectra and listen to the result.

Congratulations! You made it to the end. If there is still time left, play with the used Matlab functions and the audio signals.

[8]Often the amplitude response is converted to decibels. Because of the "beautiful" triangular shape, we omit this step here.