Digital Filter Implementation 1

442.003 Digital Signal Processing, Laboratory
Winter Term 2023/24
Signal Processing and Speech Communication Laboratory
www.spsc.tugraz.at

Last updated: November 3, 2023

Abstract

The main topics of this part of the laboratory are structures for digital filters (FIR/IIR filters in direct form and cascade form) and effects of finite-precision arithmetic (generation of round-off noise and effects of coefficient quantization).

1 Theoretical Overview

This short overview only presents some of the most important points. For more complete descriptions see the references mentioned at the end.

1.1 FIR Filter

A causal filter with a finite impulse response (FIR) computes the output signal $y[n]$ as a weighted sum of the actual and M previous input samples $x[n]$, where M is the order of the filter. Consequently, the relation between input and output can be described according to the following difference equation:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

The result $y[n]$ is the discrete convolution of $x[n]$ with the (finite) impulse response

$$h[n] = \begin{cases} b_k & k = 0,\, 1,\, \ldots,\, M \\ 0 & \text{otherwise} \end{cases} .$$

Note that if the filter order is M, the impulse response has $M + 1$ coefficients.

The z-transform of the impulse response $h[n]$ yields the transfer function $H(z)$, which has, in the case of a causal FIR filter, M zeros whose positions are determined by the coefficients b_k and an M-fold pole at the origin (i.e., $z = 0$). Thus, an FIR filter is always stable[1]. Due to the fact that the transfer function of the filter is exclusivly determined by the position of the zeros, FIR filters are called 'all-zero filters'. Be aware that this can be misleading.

[1] Another condition for stability equivalent to the position of the poles is the condition that the impulse response has to be absolutely summable, i.e., $\sum_{k=0}^{M} |b_k| < \infty$. This condition is automatically fulfilled for FIR filters.

Linear-Phase Filter Another property unique to FIR filters is the linear phase property. It can be shown that if the impulse response $h[n]$ of the filter is symmetric, i.e.

$$h[M-n] = h[n] \quad n = 0,\, 1,\, \ldots,\, M \;\text{ (even symmetry, cosine terms only)}$$

or

$$h[M-n] = -h[n] \quad n = 0,\, 1,\, \ldots,\, M \;\text{ (odd symmetry, sine terms only),}$$

the frequency response $H(e^{j\theta}) = |H(e^{j\theta})|e^{j\angle H(e^{j\theta})}$ of the system has a linear phase function $\varphi(\theta)$. In other words, if the symmetry condition is satisfied, we have

$$\varphi(\theta) = \angle H(e^{j\theta}) = \pm\frac{k\pi}{2} - \frac{M}{2}\theta.$$

In these cases the filter has a constant group delay of $\tau_g(\theta) = -\frac{d\varphi(\theta)}{d\theta} = M/2$ samples or $\tau_g(\omega) = -\frac{d\varphi(\omega)}{d\omega} = M/(2f_s)$ seconds.

1.2 IIR Filter

A causal filter with an infinite impulse response (IIR) computes the output samples $y[n]$ as a weighted sum of the actual and M previous input samples $x[n]$, *and* as a weighted sum of N previous output samples. Therefore it can be described by the difference equation

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k].$$

By setting the coefficient $a_0 = 1$ and rewriting the difference equation we get[2]

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k].$$

The difference equation above implements a cascade of a feed-forward (FIR) system and a feed-back system. The impulse response of the overall system is thus the convolution of the impulse responses of the feed-forward and the feed-back system. Accordingly, in terms of z-transforms, the overall transfer function is the product of the transfer functions of the two cascade elements.

Feed-Back System The impulse response of the feed-back structure is a weighted sum of N (complex) exponential sequences. Thus, this cascade element causes the infinite impulse response of the IIR filter. The transfer function $H(z)$ of the feed-back part has N poles whose positions are determined by the coefficients a_k and an N-fold zero, which is always at $z = 0$. For a stable causal IIR filter, all poles must be inside the unit circle. Since the poles determine the transfer function of the feed-back system, such filters are sometimes called 'all-pole filters' (misleading!).

[2] In the literature, you can often find this equation with a different sign—we use the form above to be able to exchange coefficients with MATLAB without changing the sign of a_k for $k = 1, \ldots, N$.

1.2.1 Direct Form

A filter in direct form is the straightforward implementation of the difference equation above. The transfer function has the following form:

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}.$$

1.2.2 Cascade Form

If we factor the polynomials by finding their roots (i.e., the zeros c_k and the poles d_k) we can rewrite the transfer function:

$$H(z) = b_0 \frac{\prod_{k=1}^{M}(1 - c_k z^{-1})}{\prod_{k=1}^{N}(1 - d_k z^{-1})} = b_0 \frac{z^{-M} \prod_{k=1}^{M}(z - c_k)}{z^{-N} \prod_{k=1}^{N}(z - d_k)}.$$

By combining pairs of real factors and complex conjugate pairs into second-order stages (so called *biquads*) we get:

$$H(z) = b_0 \prod_{k=1}^{N_{biqu}} \frac{b_{0k} + b_{1k} z^{-1} + b_{2k} z^{-2}}{1 + a_{1k} z^{-1} + a_{2k} z^{-2}}.$$

One advantage of the cascade form over the direct form is that a small change of a coefficient (e.g., due to quantization; see below) influences only the pair of poles (or zeros) of the corresponding stage. Furthermore, the amount of displacement is less than that for the overall higher-order direct form filter. Another advantage is that we can check directly if the filter is stable by looking at the a_{2k} coefficients only, at least if the system is real-valued. For a complex conjugate pair of poles d_k, d_k^* we get

$$1 + a_{1k} z^{-1} + a_{2k} z^{-2} = 1 - 2\Re\{d_k\} z^{-1} + |d_k|^2 z^{-2}.$$

1.3 Finite Word-length Effects

On a DSP or CPU we have registers with a finite length. Therefore, we have to consider the effects of a finite-precision arithmetic. This is particularly important when algorithms are implemented on a fixed-point architecture. We will consider three different effects: coefficient quantization, round-off noise, and limit cycles. The latter will be considered in the Lab 'Digital Filter Implementation 2'.

1.3.1 Quantization of Filter Coefficients

The coefficients of a specific filter have to be quantized before they can be used on a DSP, which results in a deviation from the desired frequency response. The quantization leads to a modification of the transfer function polynomials and therefore, a movement of the poles and zeros. In the worst case, a stable filter becomes unstable. A demonstrative example for this issue is the difference between a direct-form and a cascade-form implementation (see above).

1.3.2 Generation of Round-off Noise

If we like to calculate for example

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]$$

using integer arithmetic where b_k should have a fractional value (e.g., $b_k = 0.125$), we must scale it up to get a useful integer before the multiplication can be performed. We also have to scale it down again (= division) either directly after the multiplication or, if possible, after the accumulation. The result of this division is truncated to the number of expressable bits, and this leads to a loss of accuracy when it is performed on a DSP/CPU with finite register lengths.

Let Q denote this scaling factor and for simplicity, let's choose a power of 2 (e.g., $Q = 2^{15}$), which can be applied using a bitwise shift. The finest fractional resolution (i.e., the quantization step or the value of the LSB) is Q^{-1}. Introducing this scaling factor in our expression, we get

$$y[n] = \sum_{k=0}^{M} Q^{-1} \left(b_k Q x[n-k]\right)$$

or

$$y[n] = Q^{-1} \left(\sum_{k=0}^{M} b_k Q x[n-k] \right).$$

The latter option produces less round-off noise (σ_n^2 instead of $(M+1)\sigma_n^2$) but needs a wider accumulator (double-length or more).

2 Practical Part

The practical part extensively uses FIR and IIR filter implementations. Download the netbeans project Unit2 from /courses/dsplab/Unit2.zip and open it in netbeans. You will find C++ implementations for FIR (FIRfilter.cpp), IIR (IIRfilter.cpp) and Biquadfilters (Biquadfilter.cpp). Function Biquadchain.cpp generates a chain of consecutive biquad objects. Parameters for the RPi are stored in blockprocessing.h and the filter coefficients are stored in taps.cpp with data type *const float*.

A FIR object with name fir can be created using FIRfilter fir;. Its coefficients are renewed using fir.setCoefficients(const float coefficients [], const int size) and a filter operation is performed using fir.doFiltering(const float input_stream *, const float output_stream *, unsigned long framelength). The IIR object works equivalently.

Experiment 1:

Group Delay of a Linear-Phase FIR Filter

Equipment: PC + RPi, signal generator and oscilloscope, headphones

Software: Netbeans, MATLAB, download /courses/dsplab/Unit2.zip and unzip it on your workstation in directory /home/Documents

1. Connect the PC soundcard (or CD player) output with the RPi input and the headphones with the RPi output.

2. In netbeans, load the project Exp1. The source file of major interest is main.cpp where digital signal processing functions can be added to the callback loop paCallback() (line 23). The pointers input and output target to the address of the input and output stream.

3. In the callback function a FIR filter is active. Comment out the FIR filter and provide a direct connection from input to output, i.e. implement

```
for(int i=0; i < framelength; i++){
  *out = *in;
  in++;
  out++;
}
```

   Verify that netbeans considers the correct target host.

4. Provide an audio signal to the RPi input and use the headphones to listen to the output signal to check if the direct connection works as expected.

5. Start MATLAB, type » sptool, and design an FIR filter of your choice (order between 100 and 200, sampling rate is set to 16000 Hz for this experiment). Does your design yield a linear-phase filter? How do you verify this?

6. Export (in sptool) the filter variable to the MATLAB workspace (henceforth we assume filt1 for the name of the variable). To be able to use the coefficients filt1.tf.num in the C++ file taps.cpp, we have to put commas between them. This can easily be done by typing » sprintf('%.15f, \n', filt1.tf.num). In taps.cpp some coefficients are provided already. Append your coefficients with type and name const float CoeffFIR. Go to main.cpp and identify the line ud.fir.setCoefficients which loads the coefficients to the FIR filter. Replace the moving average parameters with yours. Exchange the direct connection in the callback function (comment it out) by the FIR filter ud->fir.doFiltering()!

7. Connect the signal generator output both with the RPi input and the oscilloscope. Plug the RPi output to the oscilloscope's second channel to be able to measure delays (don't forget to unplug the 3.5 mm connector!).

8. Build the program, load it to the RPi, and run it.

9. Measure the group delay $\tau_g(\omega)$ using a sine produced by the signal generator (between 1 and 2 Volts peak-to-peak) and exploiting the automatic phase-offset measuring function of the oscilloscope: after you have got a stable graph of input and output sinusoids, press **QuickMeas**, ensure ***Source*** is set to 1, and push ▶ several times until you see ***Phase 1→2***. Press it and the phase offset will be measured automatically[3].

[3]Make sure that you always display approximately two periods of the signal on the oscilloscope's screen; this way you ensure that the phase measurements have a high precision.

When you change the frequency of the sine by a small Δf (e.g. 1–5 Hz), you will get a corresponding $\Delta\varphi$. Measure several (i.e., 4-6) such (Δf, $\Delta\varphi$) pairs along the frequency axis (passband only). Create a table, e.g.:

f_1/Hz	f_2/Hz	Δf/Hz	$\Delta\varphi/^o$	$\Delta\omega$/rad	$\Delta\varphi$/rad	τ_g/ms
100	102	2	...			
400	405	5	...			
⋮						

Don't miss 2π/sign jumps and don't forget to convert to radians before calculating the group delay!

10. Compare your τ_g results with the theory. Is it constant over frequency? Are the values for τ_g comparable with the value from the filter design in MATLAB? What is wrong? Hint: how does block processing work? See [3] for more details.

11. Repeat the group delay measurement (only for one (Δf, $\Delta\varphi$) pair) when the filter is disabled in the paCallback() function in main.cpp (replace the filter by the wire through code from point 3). It suffices to measure the group delay for just a single pair of frequencies. The block size is defined in the headerfile blockprocessing.h by the constant FRAMELENGTH – do you see any connection between the group delay and the block size?

Experiment 2:
IIR Filter in Direct and Cascade Form

IIR Filter in Direct Form with 32-bit Floating-Point Coefficients

1. In netbeans open project Exp2.
2. Connect the signal generator output with the RPi input, the RPi output with the oscilloscope, and the signal generator sync with the oscilloscope's external trigger input to be able to measure frequency responses using sweeps. Set the signal amplitude to approximately 0.5 VPP.
3. Start MATLAB, type » sptool, and design a filter with the following specifications:

Algorithm	Elliptic IIR
Sampling Frequency	16000 Hz
Order	Minimum
Type	Bandpass
Fp1	2400 Hz
Fp2	3200 Hz
Apass	0.2 dB
Fs1	2320 Hz
Fs2	3280 Hz
Astop1, Astop2	40 dB

Press **Apply** and export the filter in sptool to the MATLAB workspace.

4. In MATLAB, again display the filter coefficients using » sprintf('%.15f, \n', filt1.tf.den). Copy the filter coefficients from the MATLAB window to taps.cpp to dir_a[]={…}. Repeat the same for the numerator of the transfer function, i.e., copy » sprintf('%.15f, \n', filt1.tf.num) to dir_b[]={…}. Do not close MATLAB—you will need the current filter variable later.
5. Set the appropriate FIR and IIR filter coefficients. Build the program, load it to the RPi, and run it. Measure the frequency response of this direct form filter. Do you meet the desired parameters? You should save a oscilloscope screenshot to the floppy disk for later comparisons.

IIR Filter in Cascade Form with 32-bit Floating-Point Coefficients

1. In MATLAB, change the working directory to the netbeans project directory and type » dir2cas. This script[4] converts the coefficients of the direct form to the coefficients of the cascade form (you should take a look at dir2cas.m). Answer, how this conversion works.
2. Type » cascaded and copy the coefficients to cas_coeffs[] ={…} in taps.cpp. Also activate the Biquadfilters ud->bchain.doFiltering in the callback loop paCallback() and comment out the direct form FIR filter
ud->fir.doFiltering as well as the direct form IIR filter
ud->iir.doFiltering. Rebuild and reload the program, run it and measure the frequency response again. Save another screenshot to the floppy disk.

Experiment 3:
IIR Filter in Direct and Cascade Form with Quantized Coefficients

IIR Filter in Direct Form

For this experiment, we still use the floating-point functions and floating-point coefficients, but we quantize the coefficients to simulate the effect of using fixed-point registers with a fractional portion of 15 bits. We continue with netbeans project Exp2.

1. The MATLAB script dir2cas.m has already calculated the quantized coefficients—they are stored in the variables a_quant and b_quant.
2. Copy them to a[]={…} and b[]={…} in taps.cpp. Activate the FIR and IIR filter and deactivate the Biquad filter in the callback loop. Load the coefficients to the FIR and IIR objects in main(), rebuild and reload the program, and run it again. What can you observe?
3. You should check if the quantized coefficients still yield a stable filter! The MATLAB script dir2cas.m has also calculated the radii of the poles (variable p_abs).

[4]Make sure that your filter is stored in a struct with the name filt1. If your filter has another name, e.g., filt2, you can easily rename it by typing » filt1=filt2.

4. Repeat this task using a finer quantization (e.g., 17 or 18 bits). You can change the quantization by editing dir2cas.m. Look at the radii of the poles! Rebuild the DSP program again, and, if the filter is stable, measure the frequency response again. Are there any differences to observe?

IIR Filter in Cascade Form with Quantized Coefficients

1. For the case you have performed the last experiment using 16 bits, run the MATLAB script dir2cas.m again with 15 bits.
2. This MATLAB script also calculates the quantized coefficients for the cascade form filter—they are stored in the variable cascaded_quant.
3. Copy them to cas_coeffs[]={…} in taps.cpp. Load the coefficients to the biquad chain object, activate the Biquads and deactivate the FIR and IIR. Rebuild and reload the program, and run it again. Does it work? Measure the frequency response again and compare it with the previously measured ones.
4. How can you easily check whether the quantized cascade coefficients yield a stable filter or not?

A Credits

This document was authored and/or adapted by Christian Feldbauer, Bernhard Geiger and Josef Kulmer.

References

[1] Oppenheim, A.V. and Schafer, R.W.: "Discrete-Time Signal Processing," Second Edition, Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1999.

[2] Doblinger, G.: "Signalprozessoren. Architekturen—Algorithmen—Anwendungen," J. Schlembach Fachverlag, Weil der Stadt, Deutschland, 2000.

[3] Feldbauer, C.: "Real-Time Block Processing Environment," Laboratory Handout, http://www.spsc.tugraz.at/courses/dsplab/intro/blockprocessing.pdf, 2005.