Digital filters

Honza Černocký, ÚPGM

Aliases

- Numerical filters
- Discrete systems
- Discrete-time systems
- etc.

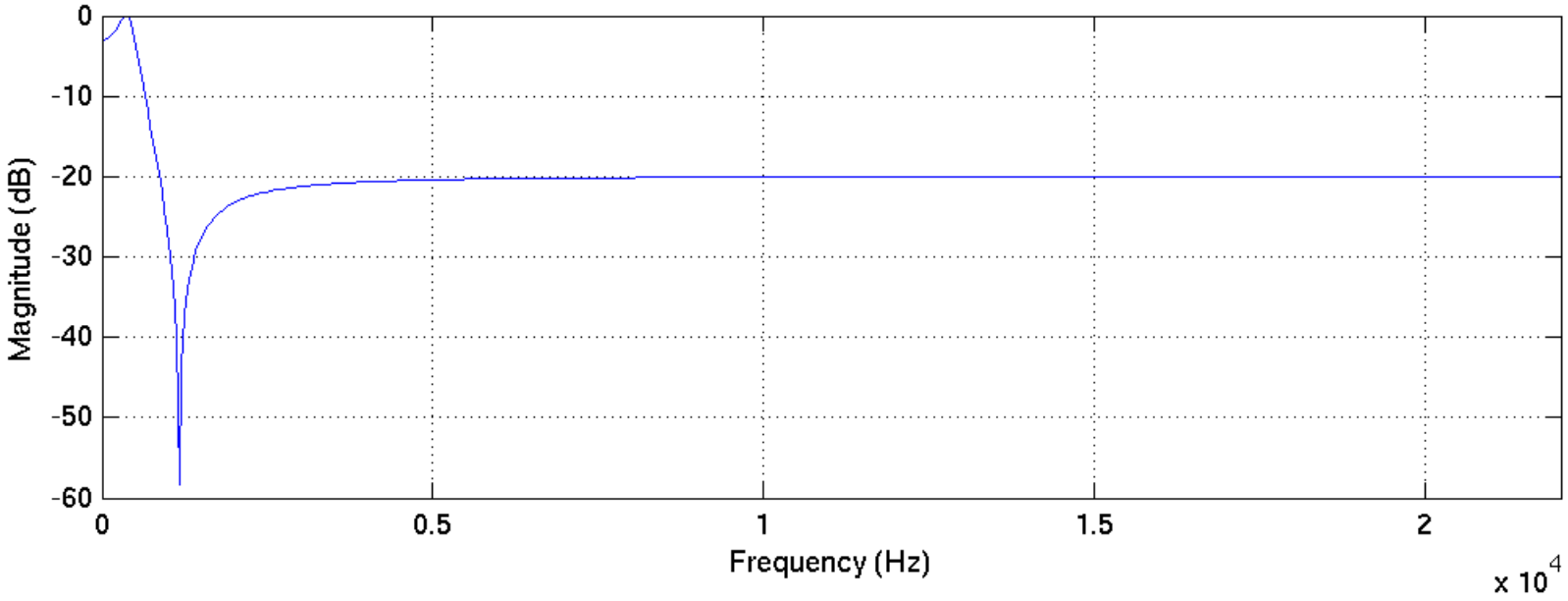
What for ?

Processing of signals

- Emphasizing
- Attenuating
- Detecting

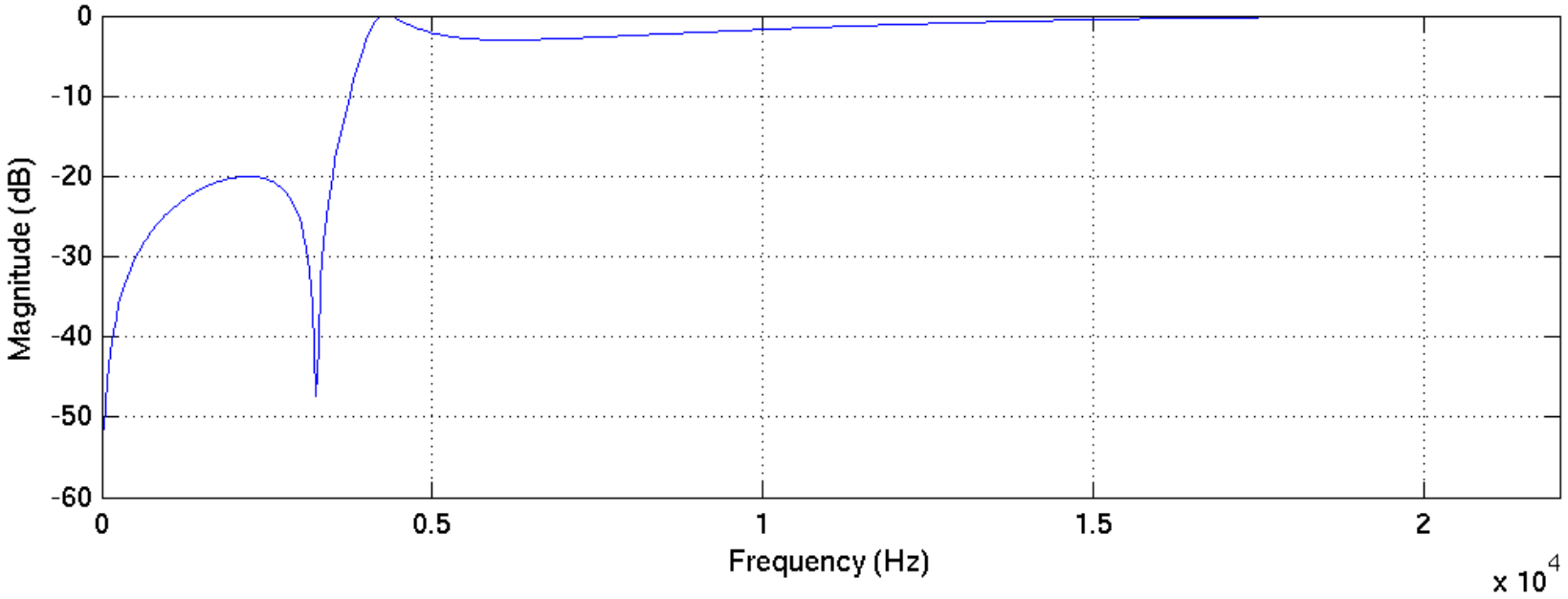
Emphasis

- Basses – low frequencies



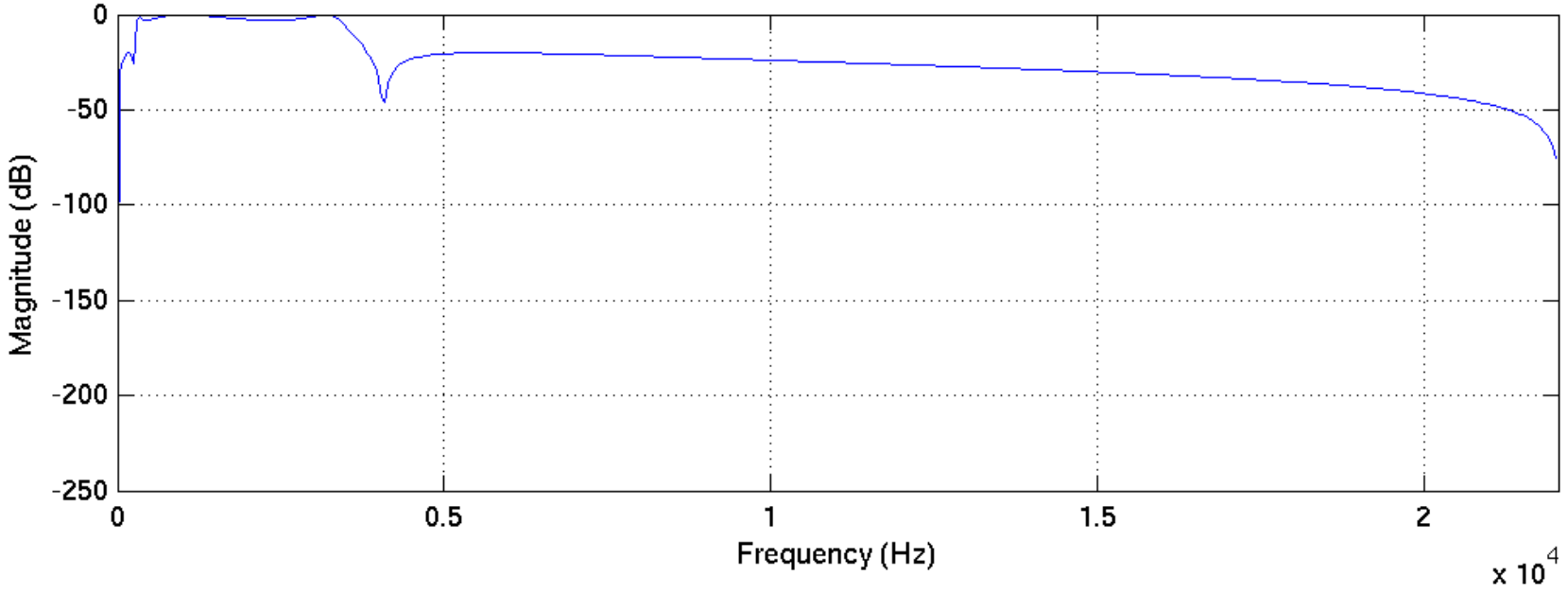
Emphasis

- Trebles – high frequencies



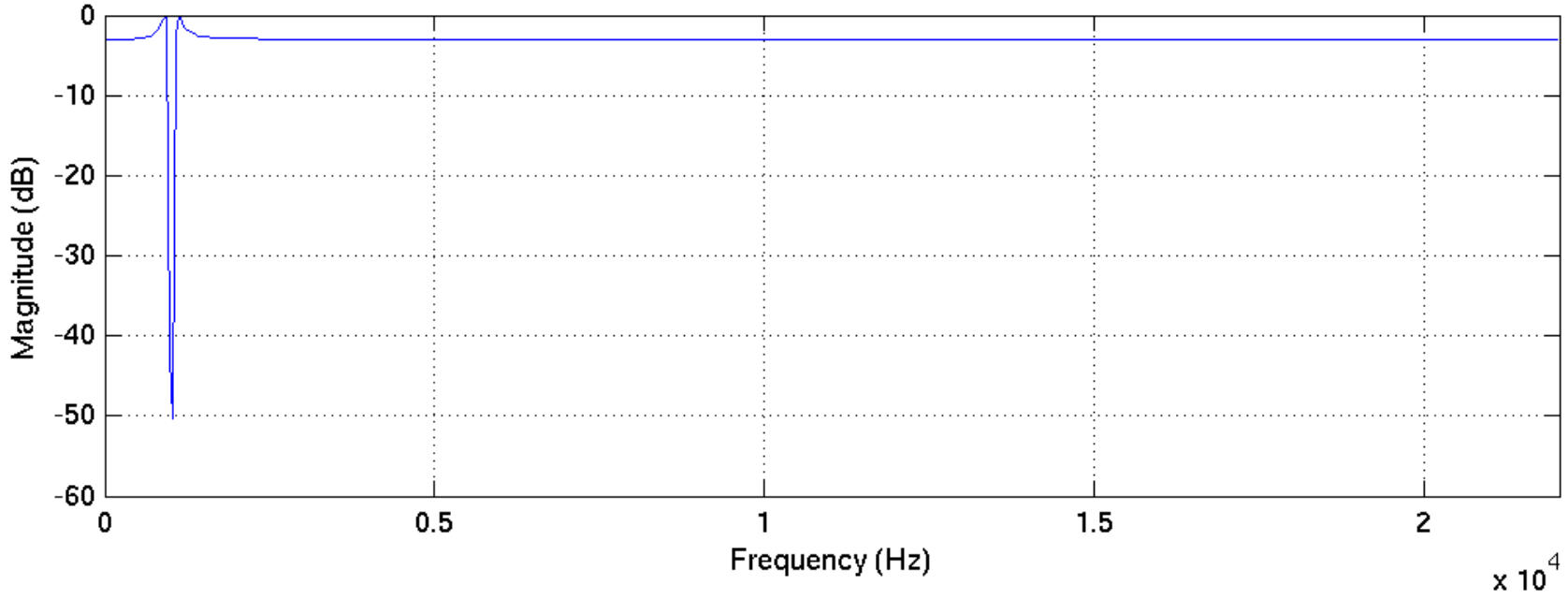
Emphasis

- Telephone band 300-3400 Hz



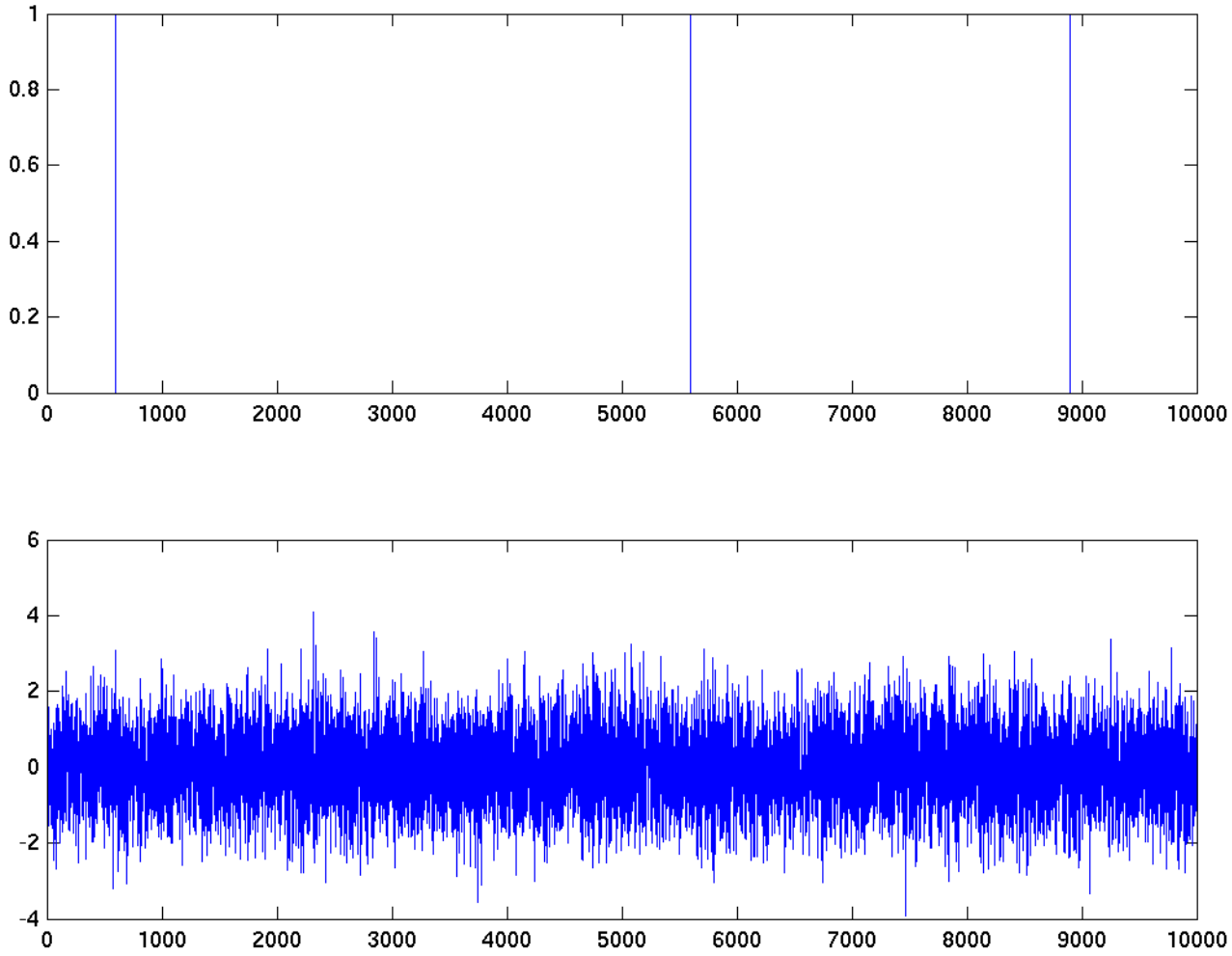
Attenuating

- A signal contamined by 1kHz and its cleaning by a sharp band-stop



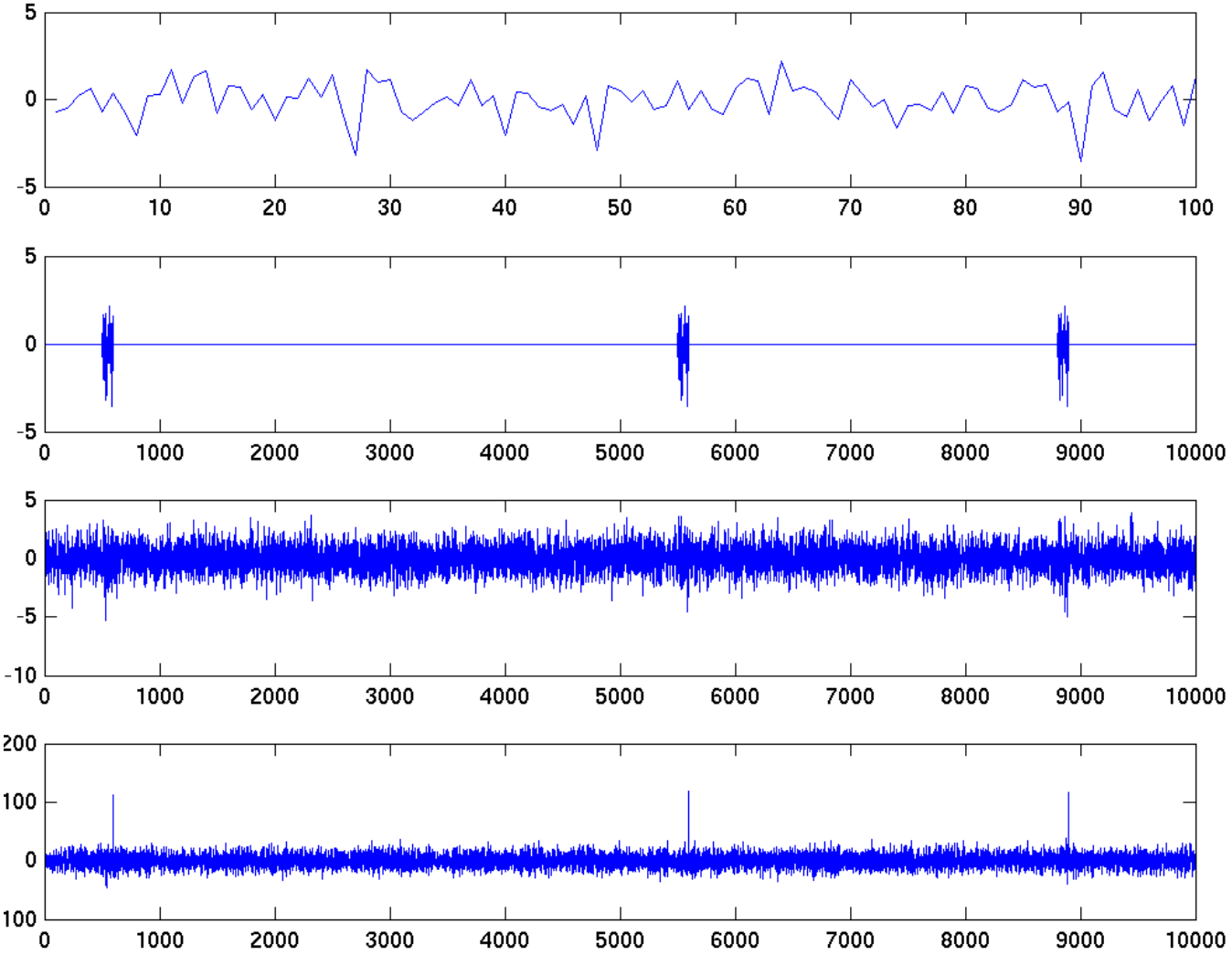
Detection

- Couple of peaks in noise



Detection

- Matched filter



- A nice app: RICHTER Jiří. Echo-Based Distance Measurement on Mobile Phone, BP FIT, 2014/2015

What will be needed ?

- Sampled signal
  - At some sampling frequency (most often 8kHz, 16kHz, 44.1kHz (CD), 48kHz, 96kHz)
- For visualization, will show it as continuous (stem vs. plot)
- For ISS no special formats (MP3, OGG, WAV), but a plain RAW
  - No header, sequence of 16-bit shorts

… wire.c

What next ?

- Only **3** simple operations
  - Multiplication with a constant *****
  - Summation (addition) **+**
  - Shift

Shift ? Only to the past !

x[n]

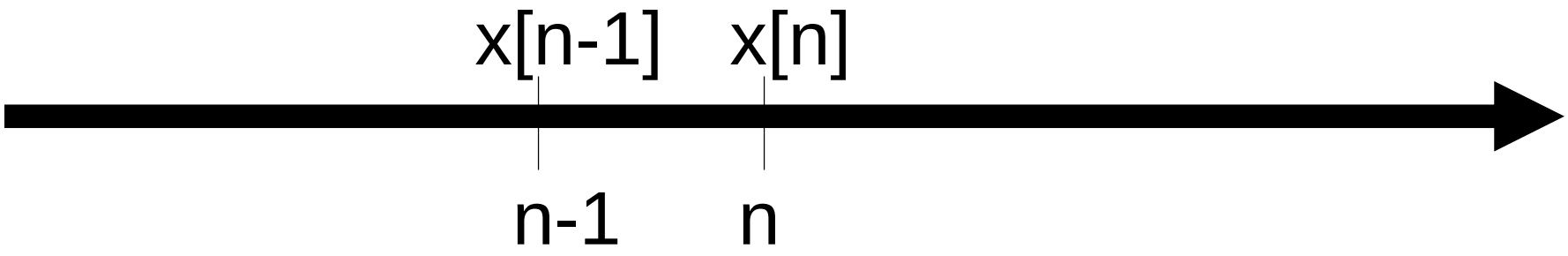
n

„now“

„pres

past

Shift off-line



… just changing the value of the pointer.

Shift on-line

- We have only *x[n]*
- Function with a **memory**

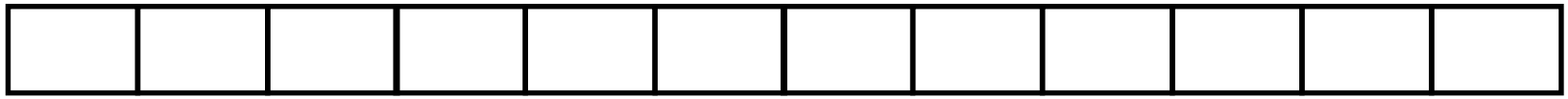
```
float filter (float xn) {
   static float xn1, xn2;
   … some processing …
   xn2 = xn1;
   xn1 = xn;
   return something;
}
```

A simple filter

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + b_3x[n-3]$$

… **difference equation**

How does it work ?



Implementation off-line

- Allocation of space for the output signal.
- Then absolutely verbatim re-writing of the difference equation

fir_offline.c

Implementation on-line

- A function for filtering is called for every sample
- It must memorize the past samples

fir_online.c

What will it do for …

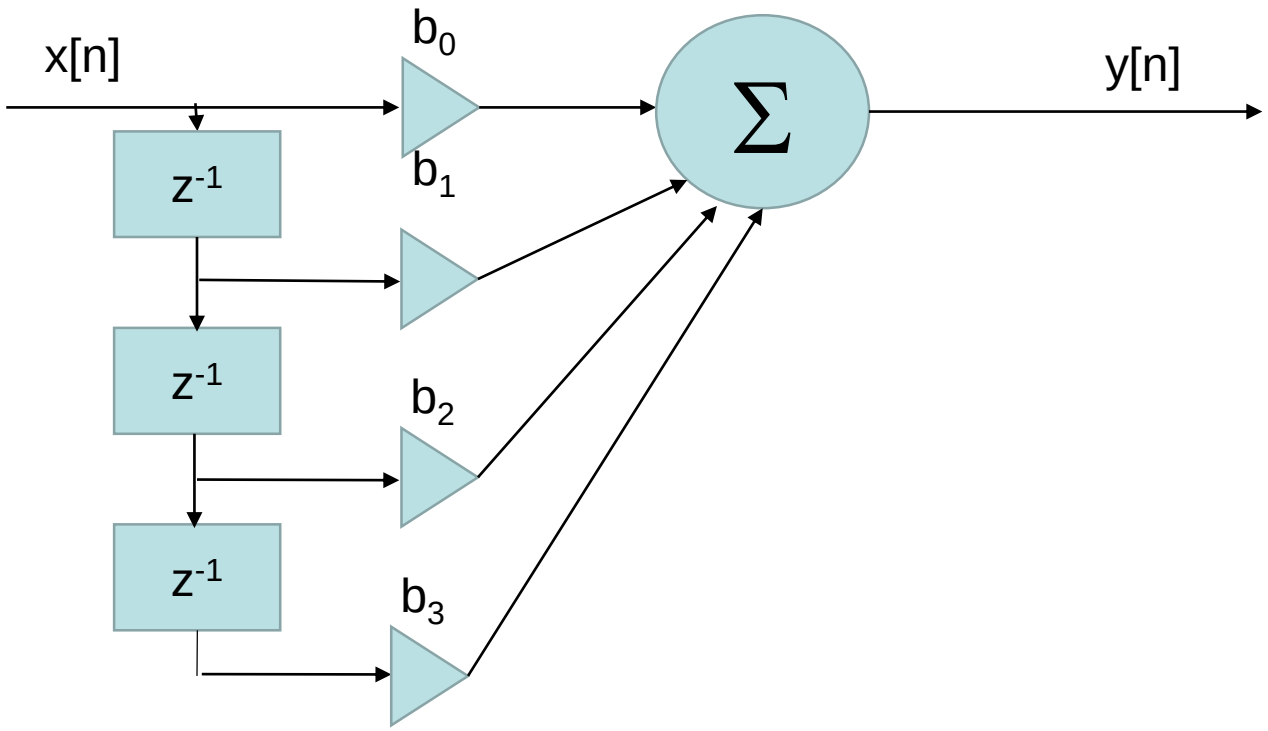
b0	b1	b2	b3
0.25	0.25	0.25	0.25
0.25	-0.25	0.25	-0.25
1	0	0	0
2	0	0	0
0	0	0	1

Another possible implementation…

How to represent a filter ?

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + b_3x[n-3]$$

… **difference equation**



… **scheme**

Impulse response

- Reaction to unit impulse

0	0	0	0	0	1	0	0	0	0	0	0

Finite / infinite impulse response?

Convolution

$$y[n] = x[n] \star h[n]$$
$$= \sum_{k=-\infty}^{+\infty} x[k]h[n-k]$$

… demo on our filter

It works also the other way round (convolution is commutative…)

$$y[n] = x[n] \star h[n] = h[n] \star x[n]$$

$$= \sum_{k=-\infty}^{+\infty} h[k]x[n-k]$$

… demo on our filter – homework ?

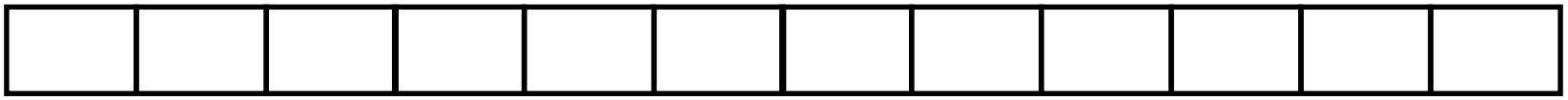
Convolution - summary

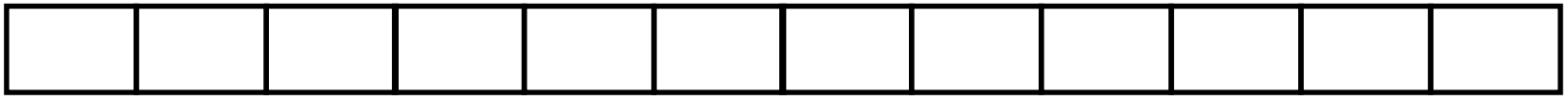
- Flip the impulse response
- Shift it to given „n“
- Mutliply
- Add
- Write the results

… „paper strip“ method – demo

Making use of filter **output**

- Feedback – **recursive filters**





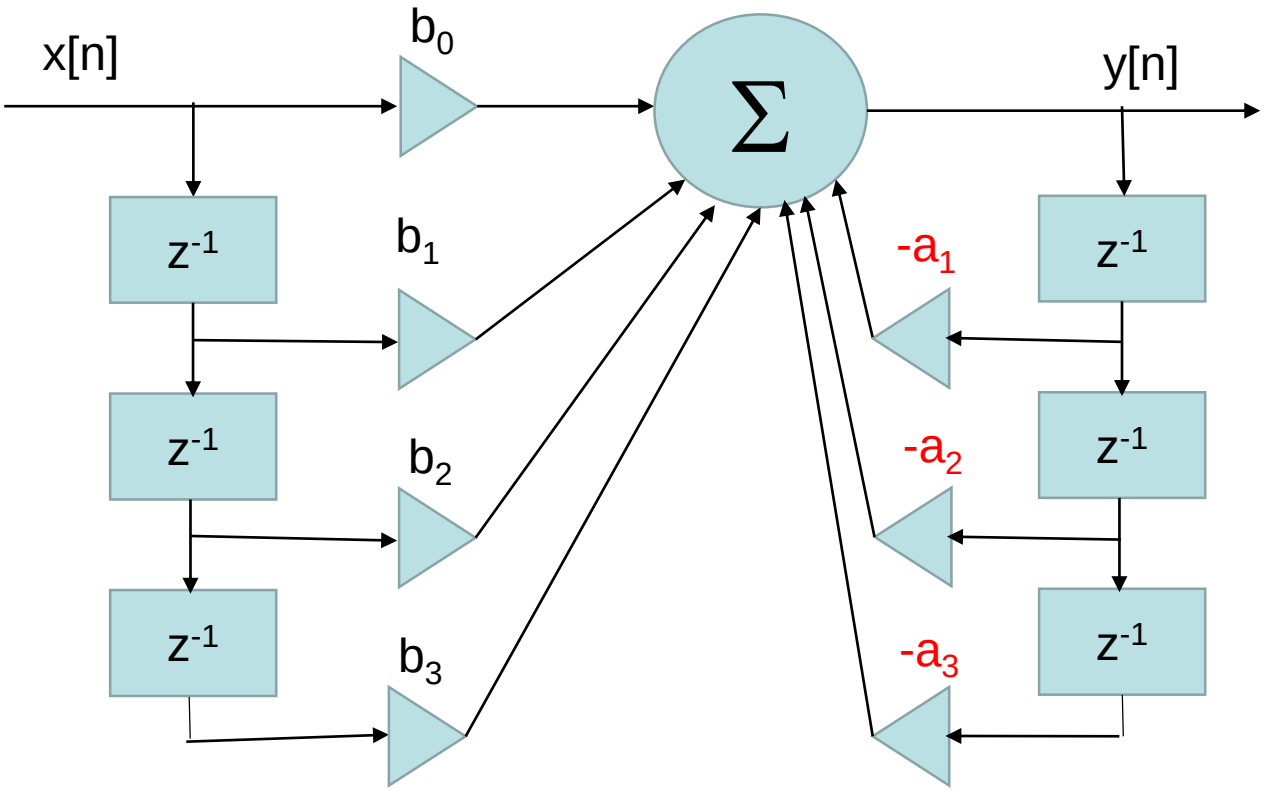
Diference equation

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + b_3x[n-3]$$
$$- a_1y[n-1] - a_2y[n-2] - a_3y[n-3]$$

… why not a_0 *?*

… and why do the feed-back coefficients have negative sign ?

Scheme



Implementation off-line

- Digging also in the old outputs …

… **iir_offline.c**

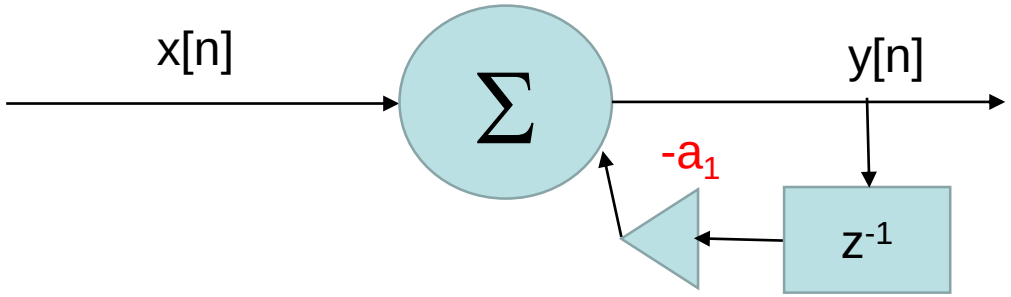
Implementation on-line

- The function has to memorize also the past outputs.

… **iir_online.c**

Impulse response

- For example for a simple, purely recursive filter (nothing done with the input)



- **Infinite impulse response - IIR**

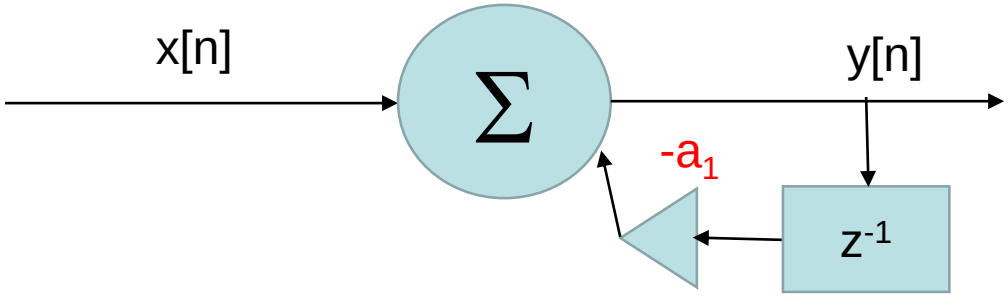
Stability

- Mathematically „Bounded Input Bounded Output“
- Popularly: “if reasonable stuff on the input expecting reasonable stuff on the output”

Stability of FIR

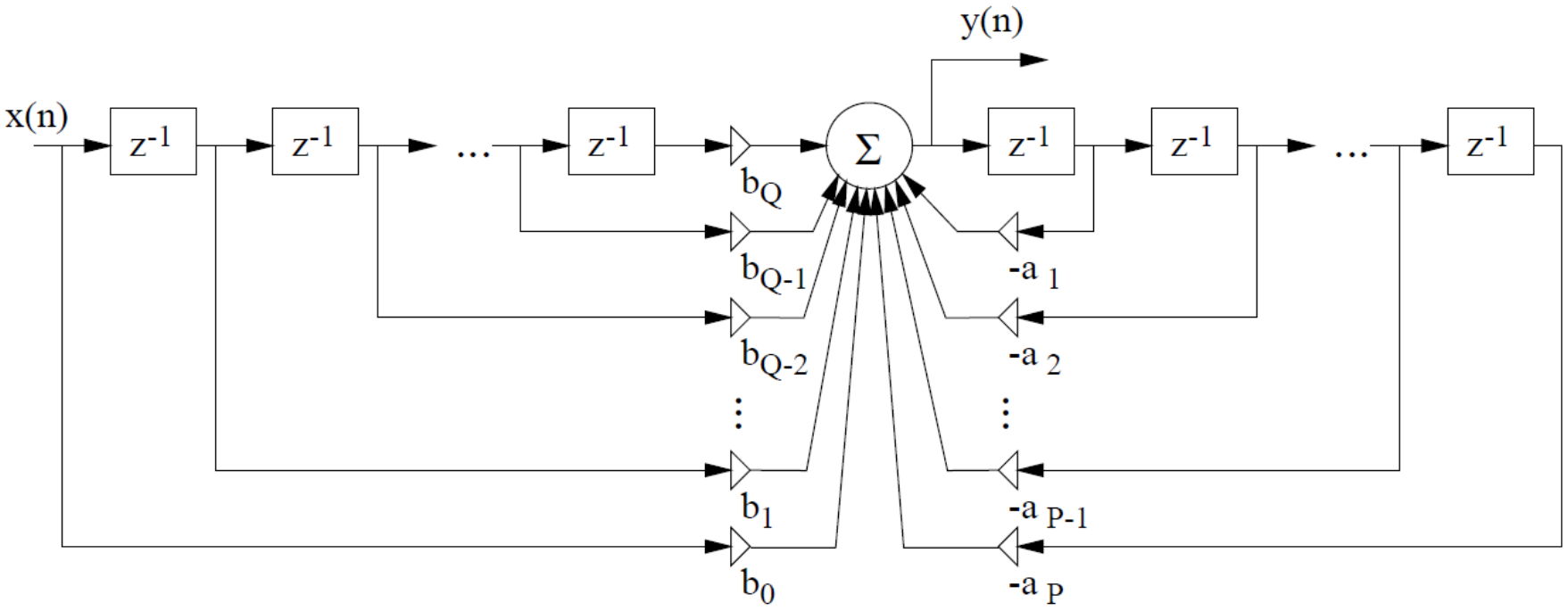
Stability of IIR

First order IIR



- What must be respected for a_1?
- And how about for more complicated IIR filters ?

General filter



General filter

- Order of input *Q*
- **Order of output** *P*

Difference equation

$$y[n] = b_0x[n] + b_1x[n-1] + \ldots + b_Qx[n-Q]$$
$$- a_1y[n-1] + \ldots + a_Py[n-P]$$

$$y[n] = \sum_{k=0}^{Q} b_k x[n-k] - \sum_{k=1}^{P} a_k y[n-k]$$

Implementation off-line

- … still the same, this time with cycles

iirbig_offline.c

Implementation on-line

- The cycles must run backward (otherwise the old values rewrite the new ones)
- The cycle for the output must stop at index 1.
- „Trash“ memory fields, that will never be used – make things easier and save us some „if“s.

iirbig_online.c

Frequency responses

- For FIR filters:
- The coefficients of the filter (resp. of its impulse response) act as a **machining tool** – the result will be similar

Tests

- 2s of chamber „a“ (440 Hz) plus some noise

See the Matlab file **matlab_filtry.m** (section “filcy atd”)

Mason’s float

- 20 equal samples valued 1/20
- Smoothing, less noise

=> Low-pass

Mason’s superfloat

- 181 equal samples valued 1/181
- Smoothing, less noise but almost no signal left ☹

=> Even more low-pass

Cutter with a dented edge

=> High pass

Band-pass ?

- We want to select 440 Hz.
- The output will resemble the impulse response …

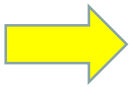
… listen to the result also on a speech file.

Frequency response more precisely

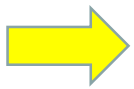
- Currently, interested only in magnitude
- i.e. how individual frequencies are amplified / attenuated
- … not how they are shifted

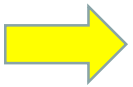
See the last part of **matlab_filtry.m**

Method 1 – measure !



filtr





Method 2 – use impulse response

- Generate impulse response and perform its frequency transform
- But the impulse response can be loooonnnnng.

The ultimate solution
z-transform

Basic tools:

$x[n] \Rightarrow X(z)$

$a\,x[n] \Rightarrow a\,X(z)$

$x[n-k] \Rightarrow X(z)\,z^{-k}$

Difference equation => transfer function

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots + b_Q x[n-Q]$$
$$- a_1 y[n-1] + \ldots + a_P y[n-P]$$

Transfer function => frequency response

$z \Rightarrow e^{j2\pi f}$

- *f* is normalized by the sampling frequency
- Will obtain a complex number, must take only its magnitude.
- Matlab: by hand or with `freqz`

Relation of time and spectra

$$y[n] = x[n] * h[n] \qquad \text{(convolution)}$$

$$Y(f) = X(f)\,H(f) \qquad \text{(mutliplication)}$$

- Attention, must multiply the complex numbers in spectra !

Summary

Describing the filter by

- Scheme
- Difference equation
- Impulse response
- Transfer function
- Frequency response

Summary II.

- Types of filters
  - FIR
  - IIR
- Implementation
  - Mutliplication
  - Summation
  - Shifts
    - Off-line
    - On-line

Summary III.

- Computing the frequency response
  - Ugly:
    - „measurement“
    - Analysis of h[n]
  - Nicely
    - Transfer function: replacing z by $e^{j2\pi f}$
    - f is normalized frequency

TODO's

- Phase – shifts of signals
- Stability for more complex IIR filters
- Design of filters
- Learning filters on data

The END