Module IN3031 / INM378 Digital Signal Processing and Audio Programming

Johan Pauwels
johan.pauwels@city.ac.uk

based on slides by Tillman Weyde

Digital Filtering (moving from theory to applications)

RECAP: Convolution

- **Convolution** combines two signals, **similar** to **cross-correlation**
  - it's the correlation with a reversed signal

$$conv(s1, s2)[t1] = \sum_{t=0}^{N2-1} s1[t1-t]\, s2[t]$$

N2 is the length of s2, s1[i] = 0 assumed where i<0 or i>=N

- Often **written** as **s1 * s2**

RECAP: Convolution Theorem

- The **most important property** of the convolution is given by the convolution theorem:
  A convolution in the time domain
  is equivalent to
  a multiplication in the frequency domain*:*

$$x * y \circ\!\!-\!\!\bullet X \cdot Y$$

meaning: $FT(conv(x, y)) = FT(x) \cdot FT(y)$

where '$\cdot$' is pointwise multiplication and

x, y are assumed to have equal length.

Digital Filters

- Sound **spectra** are **changed** by **filters**
- **STFT** manipulation and re-synthesis - a form of **filtering in the frequency domain**
- **Most filtering** happens in the **time domain** by **convolution** (FIR) plus **recursion** (IIR)

Linear Filters

- **2 types**, depending on where they get the signal from
  - **Finite Impulse Response** (**FIR**) filters
    (use **only input** signal)
  - **Infinite Impulse Response** (**IIR**) filters
    (use **input & output** signal)
- We **typically** keep the filter itself fixed (**time-invariant**) and **describe** it by its **coefficients**

The Order of Filters

- An **FIR** filter ***f*** of **order *k*** has this **structure**
  $f(x[n]) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$

- An **IIR** filter *g* of **order *k*** has this **recursive** structure
  $g(x[n]) = + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$
  $- a_1 g(x[n-1]) - a_2 g(x[n-2]) - \ldots - a_k g(x[n-k])$

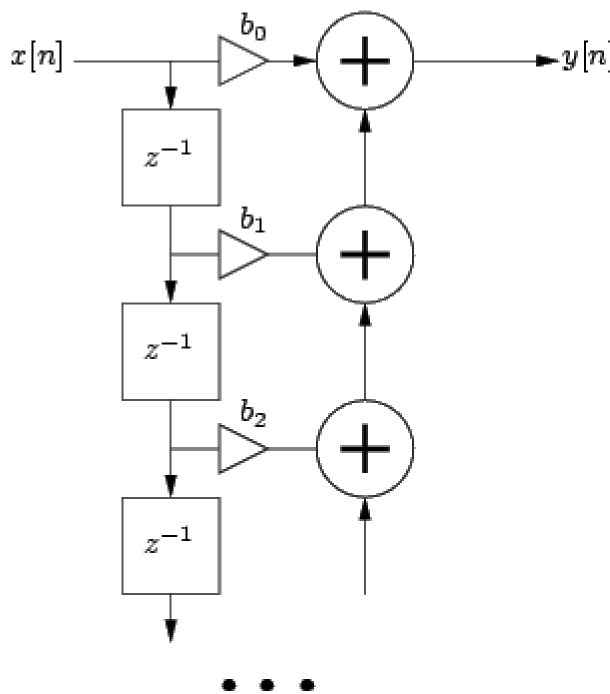
- or as a **difference equation**

$$y[n] = -\sum_{i=1}^{k} a_i y[n-i] + \sum_{i=0}^{k} b_i x[n-i]$$

- where $\mathbf{a_n}$ and $\mathbf{b_n}$ are the **filter coefficients**

An FIR Filter

- An **FIR** filter ***f*** of order ***k*** has this **structure**
  $f(x[n]) = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$
  with **coefficients** $b = [b_0, b_1, b_2, \ldots, b_k]$
- **Graphically**:



▷ : multiplication with a scalar

z^{-1}: delay by 1 sample

Types of Digital Filters

- There are four **common types** of filters:
  - **low pass** (anti-aliasing, synthesis, HF noise removal)
  - **high pass** (remove rumbling, protect speakers)
  - **band pass** (sound analysis)
  - **band stop** (removing unwanted signal, e.g. from power supply)
- **Other** types of filters:
  - **comb** filters (usually the result of short delays)
  - **all pass** filters (modify only the phase)

Uses of Digital Filters

- Digital filters are used in **audio** for
  - **equalisation** (removing frequency imbalances of microphones, room acoustics etc)
  - user **sound modification** (adjust to personal taste)
  - sound **analysis** (select the frequency range to analyse)
  - sound **synthesis** (shape timbre of a synthetic sound)
- … and more **generally for**
  - **anti-aliasing (**before **downsampling)**
  - **noise reduction** (remove unneeded frequencies)
  - many uses in image and video processing ...

Properties of Digital Filters

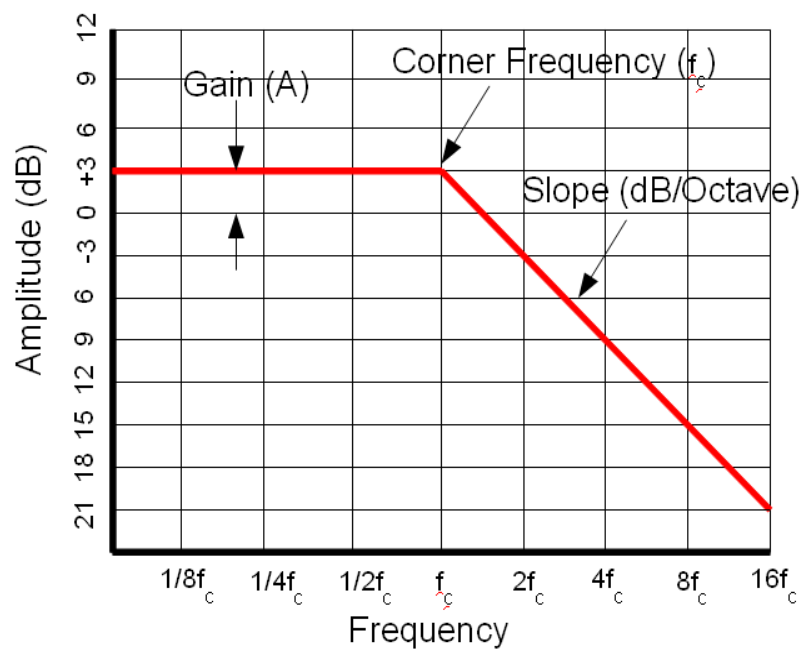
- Filter **architecture** (FIR or IIR)
- Filter **order** (#sample delays = #coefficients-1)
- Filter **coefficients** (the values defining the filter operation)

and resulting from those

- **Frequency response** (mainly magnitude)
- **Impulse response** (sometimes step response)
- **Time behaviour** (phase response, group delay)

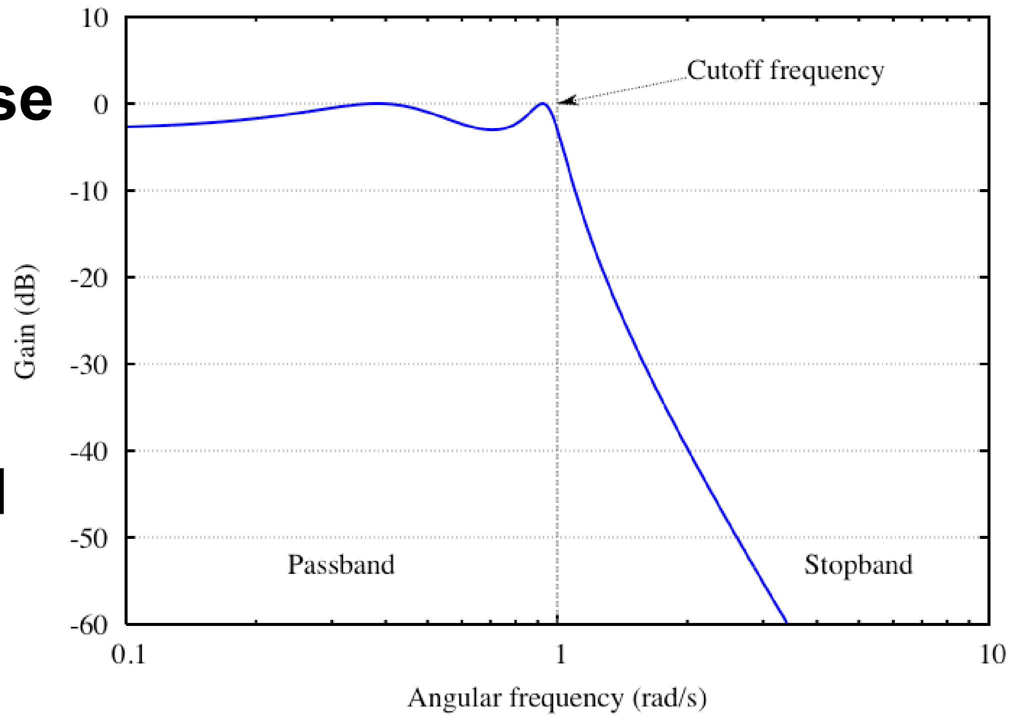
Filter Parameters

- The ***pass band*** is a range of frequencies, that should pass the filter unattenuated.
- The ***stop band*** is a range of frequencies, that should not pass the filter.
- The pass band ends at the ***corner*** *or* ***cut-off frequency*** (usually at -3dB).
- The **slope** of the freq. resp. is measured in dB/octave (sometimes decade)



More Filter Parameters

- ***Ripple*** is the **variation** of the **frequency response** within a band
- ***Resonance*** is a **peak** in frequency response **near cut-off** frequency
- ***Stability***: The filter should (usually) **not oscillate** by itself



FIR Filter Design

FIR:

- **Approach:** coefficients determined as **iFFT** of desired frequency response
- **Pro**:
  - **FIR** filters are **always stable**
  - Good phase behaviour
- **Cons**:
  - for a **steep slope** in the transition band, we need **high number of coefficients** (and thus compute time)

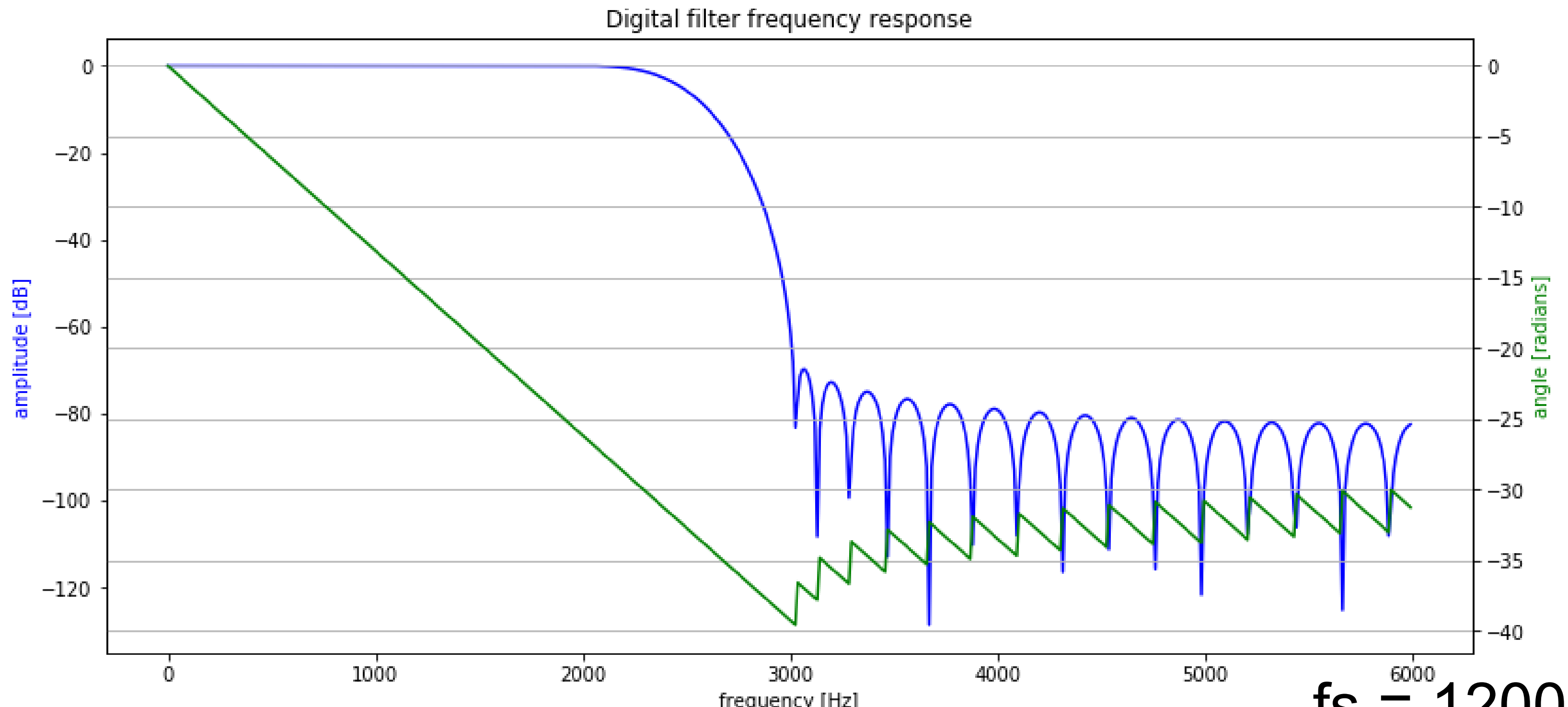
FIR Filter Design in Python

```
firls(num_taps, critical_bands,
desired_gains, fs=None)
```

- Using `scipy.signal.firls`
- `num_taps`: order + 1 (must be odd → even order)
- `critical_bands`: band freq's in increasing order (including 0 and Nyquist)
- `desired_gains`: gains at critical freq's (defines filter type)
- `fs`: samplerate, determines unit of `critical_bands` (None implies normalised freq's)

FIR Filter Design in Python

```
signal.firls(51, [0, 2000, 3000, 6000], [1, 1, 0, 0], fs=12000)
```



fs = 12000 Hz

bands:	0	2000	3000	6000
gains:	1	1	0	0

Frequency Units

- **analogue** frequencies
  - standard f : **cycles/sec** $(0 \ldots f_s/2)$
  - angular $\Omega = 2\pi f$: **radians/s** $(0 \ldots \pi f_s)$
- **digital** frequencies, independent of f_s
  - f/f_s : normalised frequency (fraction of sample rate), as **cycles/sample** $(0 \ldots 1/2)$
  - angular: $\omega = 2\pi f/f_s$: **radians/sample** $(0 \ldots \pi)$

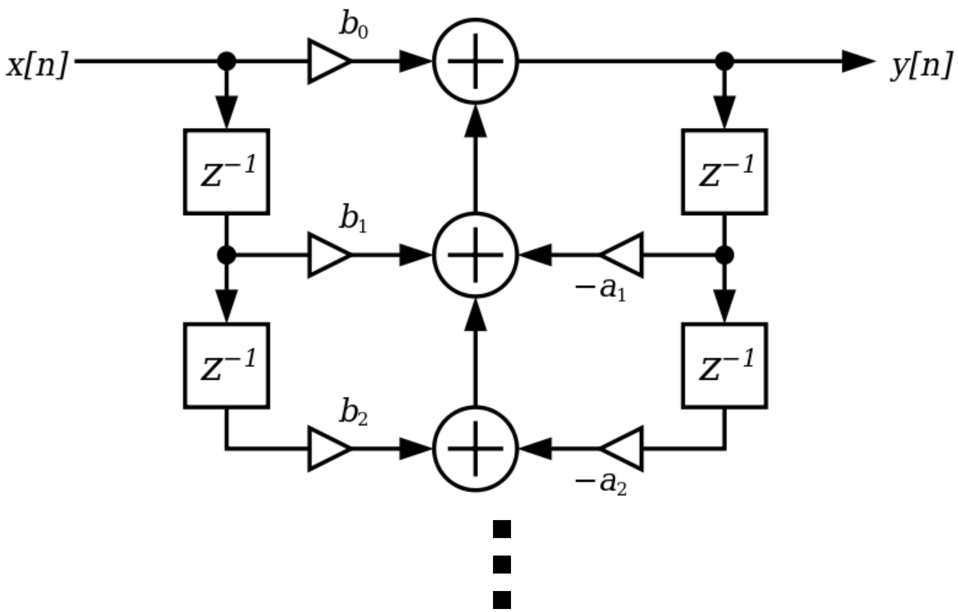
IIR Filter Design

IIR:

- **Approach:** Define **numerical methods** for finding appropriate filter **coefficients** (mathematically demanding).
- **Pro:**
  - IIR filters can be **very efficient**.
- **Cons:**
  - Uncontrolled **phase** behaviour
  - **May be unstable**
  - Quantisation noise may multiply through recursion

IIR Filter Diagram

$$y[n] = -\sum_{i=1}^{k} a_i\, y[n-i] + \sum_{i=0}^{k} b_i\, x[n-i]$$



z^{-1} represents a delay by one sample

This structure is called *Direct Form 1*

The Impulse Response of an IIR Filter

- An **IIR** filter g

$$g(x[n]) = -a_1 g(x[n-1]) - a_2 g(x[n-2]) - \ldots - a_k g(x[n-k]) + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] + \ldots + b_k x[n-k]$$

- The **impulse response** of **g** has to be **computed recursively** and **may** be **infinitely long**
- IIR filters
  - allow very **effective** filtering with **few coefficients**
  - **may oscillate** by themselves
  - **frequency response is hard** to compute

IIR Filter Design in Python

```
iirdesign(pass_freq, stop_freq, pass_gain,
stop_gain, ftype='butter', fs=None,
output='sos')
```

- Using `scipy.signal.iirdesign`
- minimum order automatically determined
- `pass_freq, stop_freq, pass_gain, stop_gain`: band freq's and gains (excluding 0 and Nyquist)

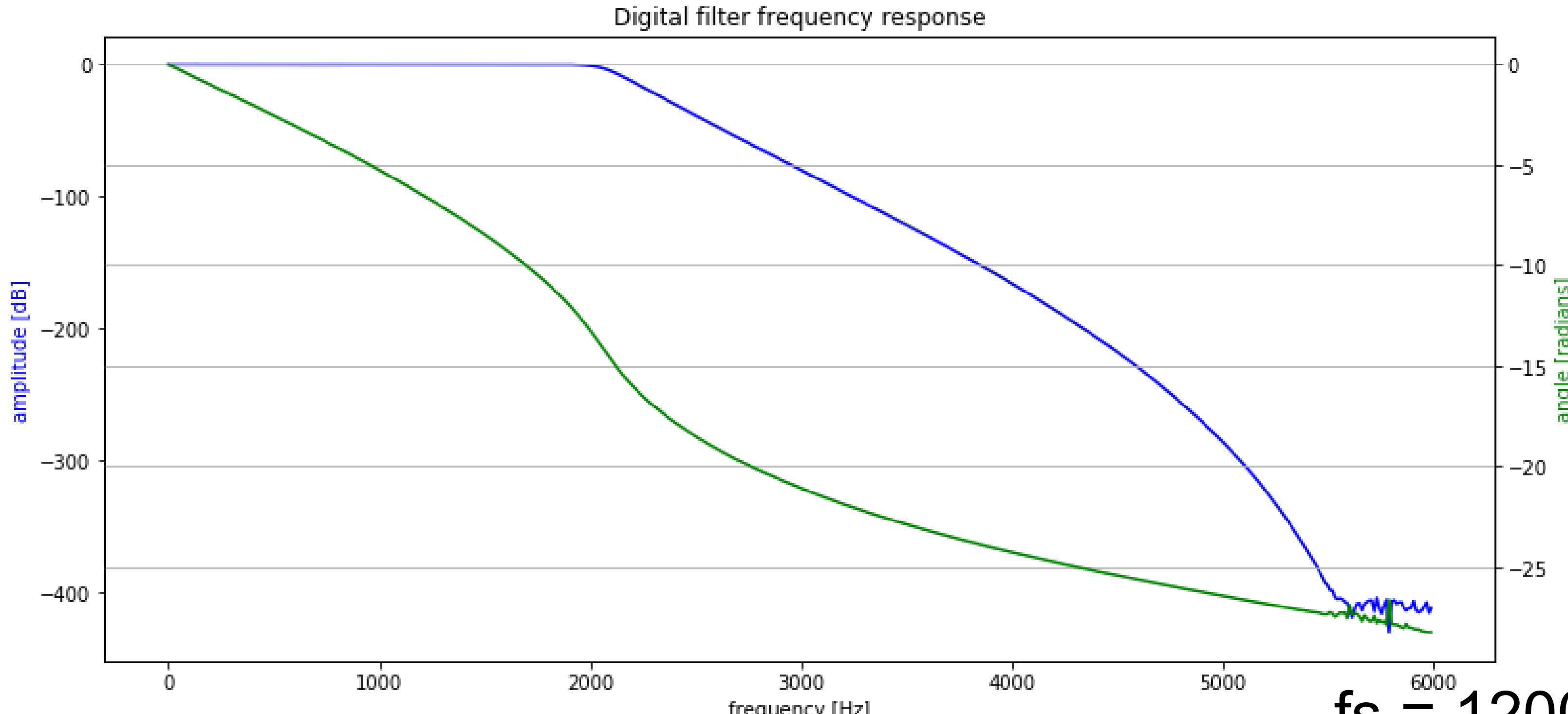
IIR Filter Design in Python

```
iirdesign(pass_freq, stop_freq, pass_gain,
stop_gain, ftype='butter', fs=None,
output='sos')
```

- `ftype`: multiple types available, trading off stopband and passband ripple, attenuation slope, resonance
- `fs`: samplerate, determines unit of `*_freqs` (None implies normalised freq's)
- `output='sos'`: instead of `'ba'` for improved numerical stability

IIR Filter Design in Python

```
signal.iirdesign(2000, 3000, 1, 80, ftype='butter', fs=1200, output='sos')
```



fs = 12000 Hz

passband freq: 2000 Hz

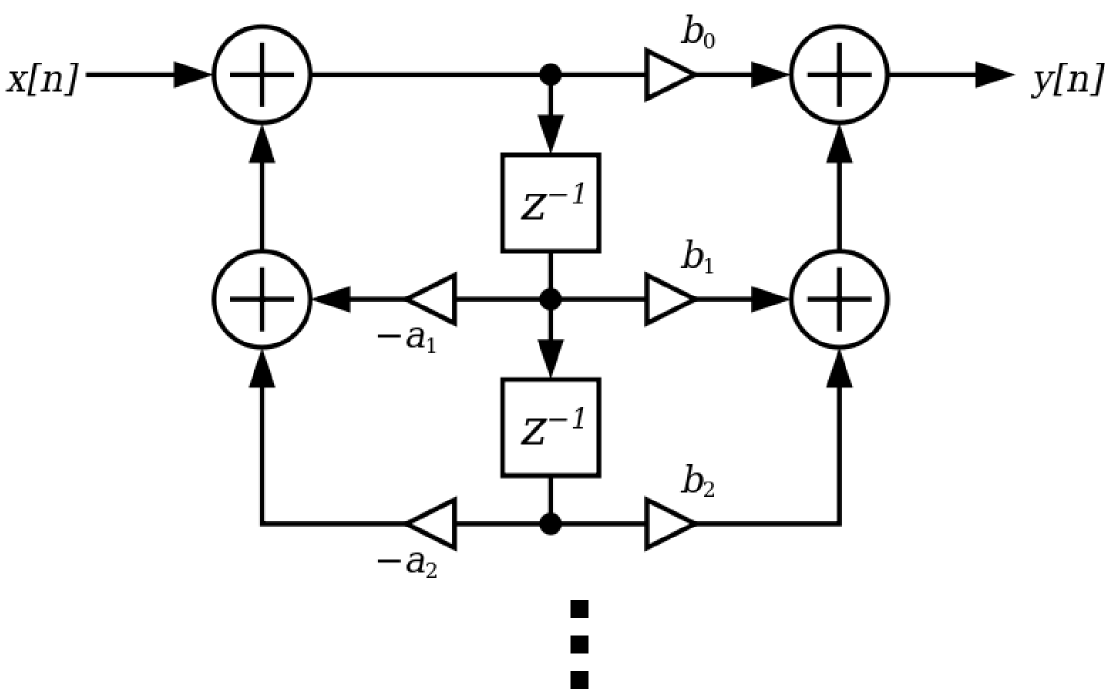
stopband freq: 3000 Hz

passband loss: 1 dB

stopband attenuattion: 80 dB

IRR Filters in Practice

- IRR filters are used in *Direct Form 2*
- Equivalent to Direct Form 1, but more efficient



IRR Filters in Practice (2)

- **Many representations** exist
  - **equivalent** possibilities (**linear systems** …)
  - **numerical** and **computational trade-offs**
- In practice the **SOS (second order sections)** is a good way to represent a filter
- Can be transformed to A and B coefficients as

```
[b, a] = signal.sos2tf(sos)
```

- Apply SOS coefficients with

```
signal.sosfilt(sos, samples)
```

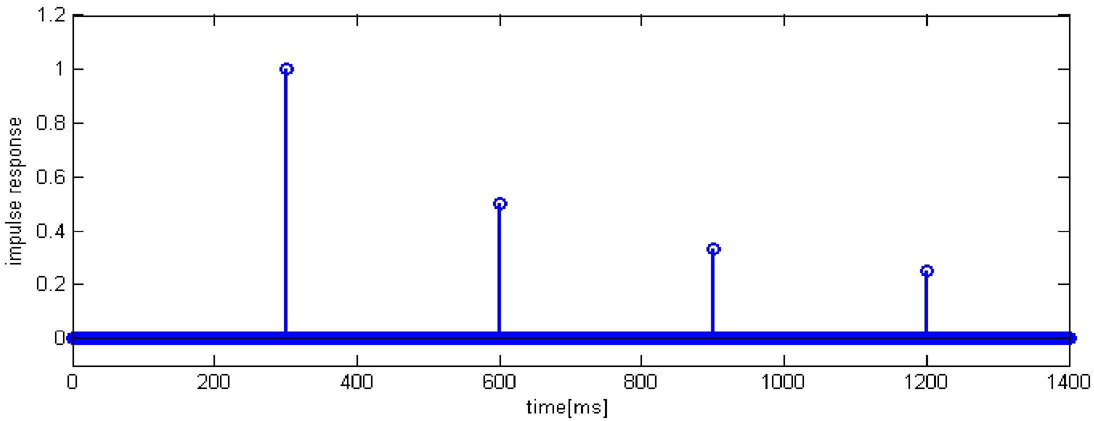
Impulse Responses and Audio Effects

Impulse Response (recap)

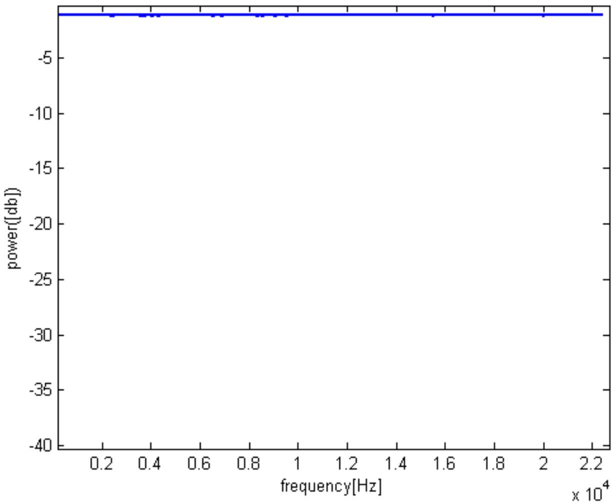
- Impulse Response is **system output** in response to a **unit impulse** [1,0,0,...].
- It **completely describes** the **behaviour** of a **linear filter** (or linear **system**) because
  - **every signal** can be described as a **sum of differently scaled unit impulses** (one per sample)
  - the **system's** signal **response** is then a **sum** of the differently **scaled impulse responses**
- Remember: a **linear system** satisfies the **superposition principle** f(ax[n]+by[y]) = a f(x[n]) + b f(y[n])

Echo

- Impulse response: Few filter coefficients **span over seconds**
- Frequency response is **flat**



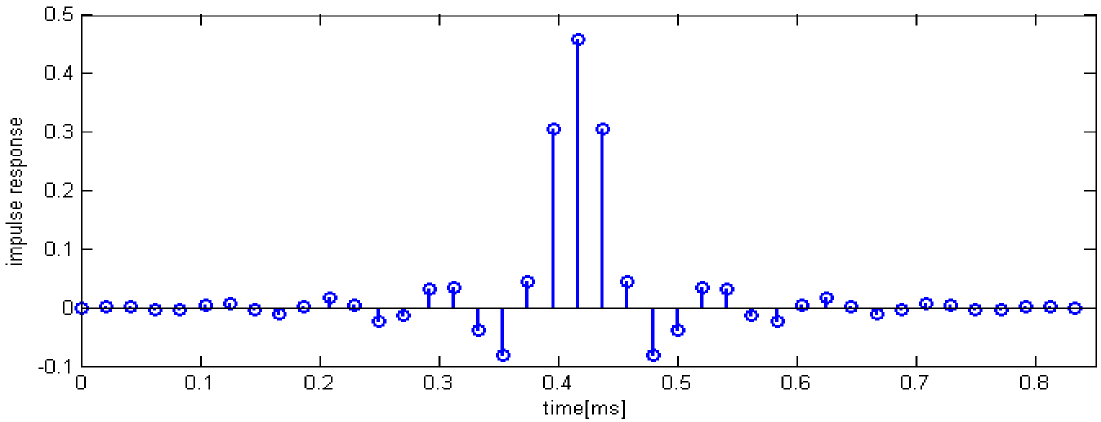
Impulse Response



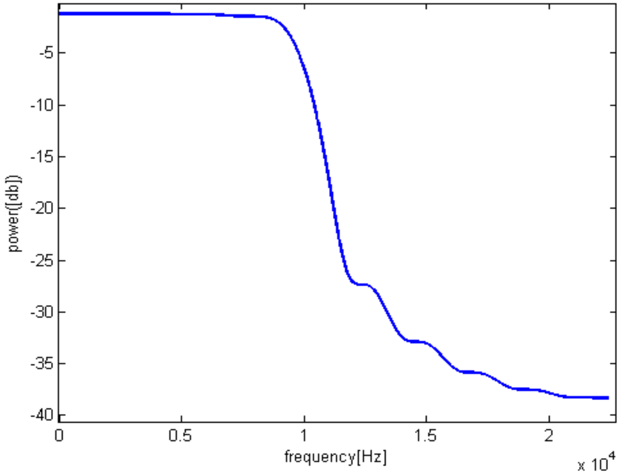
Frequency Domain

Low-Pass

- Impulse response: Many filter coefficients in the **first few milliseconds**
- **Approximates a rectangular window** in frequency domain



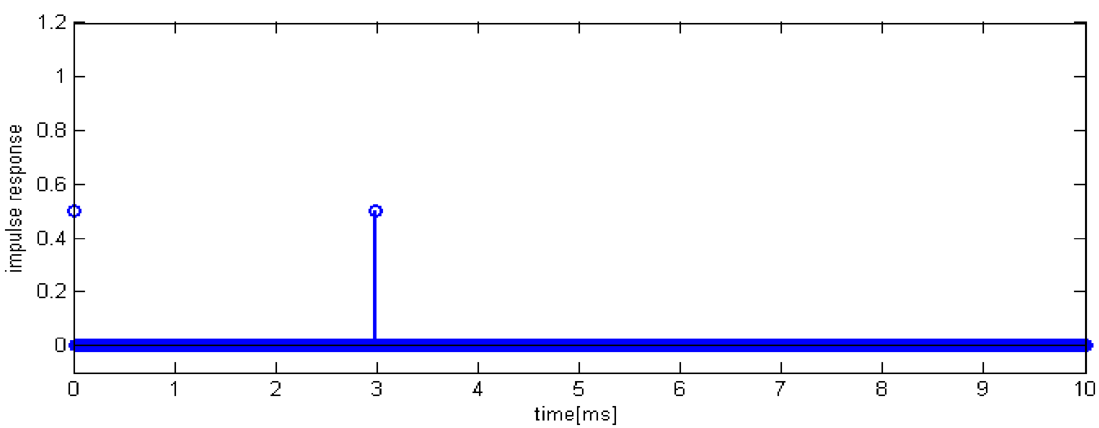
Impulse Response



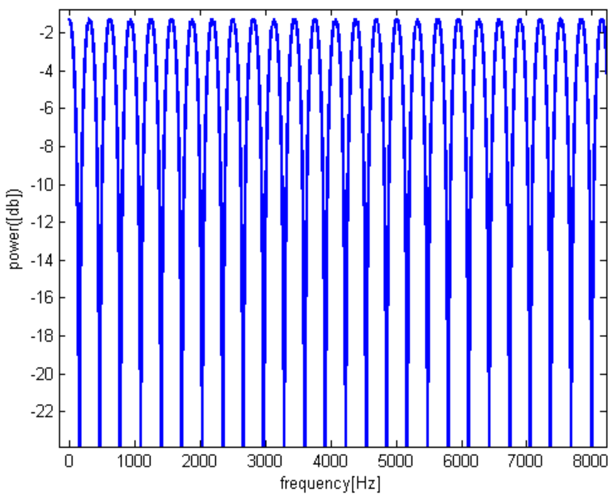
Frequency Domain

Flanger as Dynamic Filter

- Impulse response: 2 filter coefficients in the **first few milliseconds (identity + delay)**
- **Comb filter shape** in frequency domain
- IR changes over time (time-variant)



Impulse Response

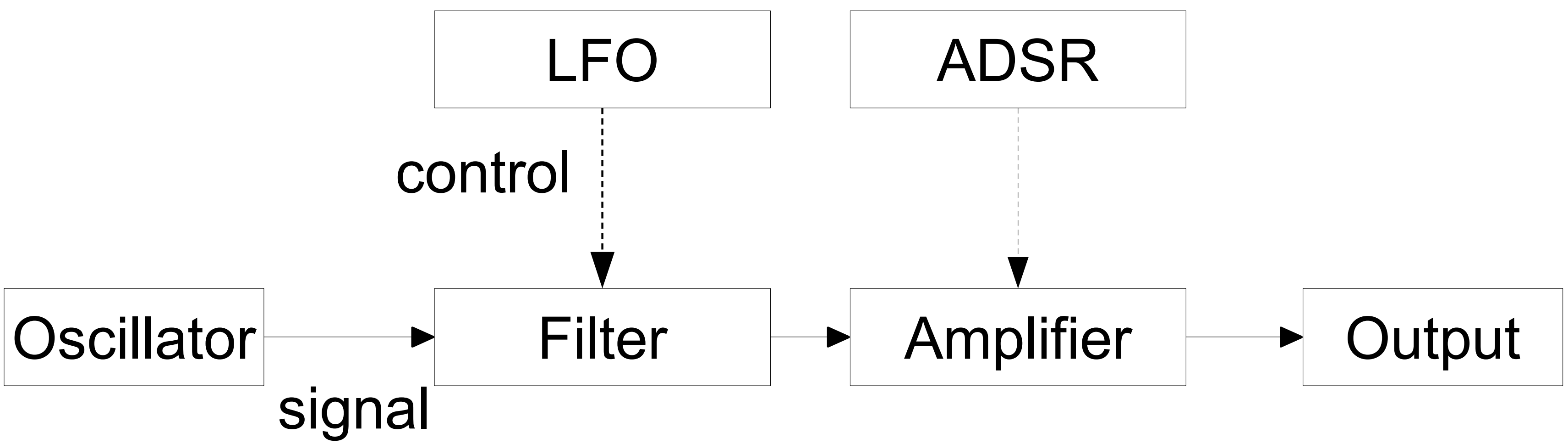


Frequency Domain

Using Filters for Subtractive Sound Synthesis

Subtractive Sound Synthesis

- **Most common** form of **analogue** synthesis
- **Generates** a sound, **filters** and **amplifies** (or attenuates)
- **Example** set-up:



"Virtual" Synthesizer

Take-Home Messages

- **FIR** and **IIR** filters are often used to remove **frequency bands** (**Hi-Pass**, **Low-Pass**, …)
- **Filters** need **delay-lines**, i.e. **memory buffers**
- **Filters** are **mostly time-invariant**, **can be time-variant** (flanger)
- Can be used in **subtractive synthesis**
- Games need **real-time programming**, i.e. time domain processing
- **Complexity** often hidden by **building blocks** (FMOD)

Reading:
FMOD Studio API Docs
Smith, DSP Guide, chpt 15