

Signal processing preliminaries

ISMIR Graduate School, October 4th-9th, 2004

Contents:

- Digital audio signals
- Fourier transform
- Spectrum estimation
- Filters

1 Digital signals

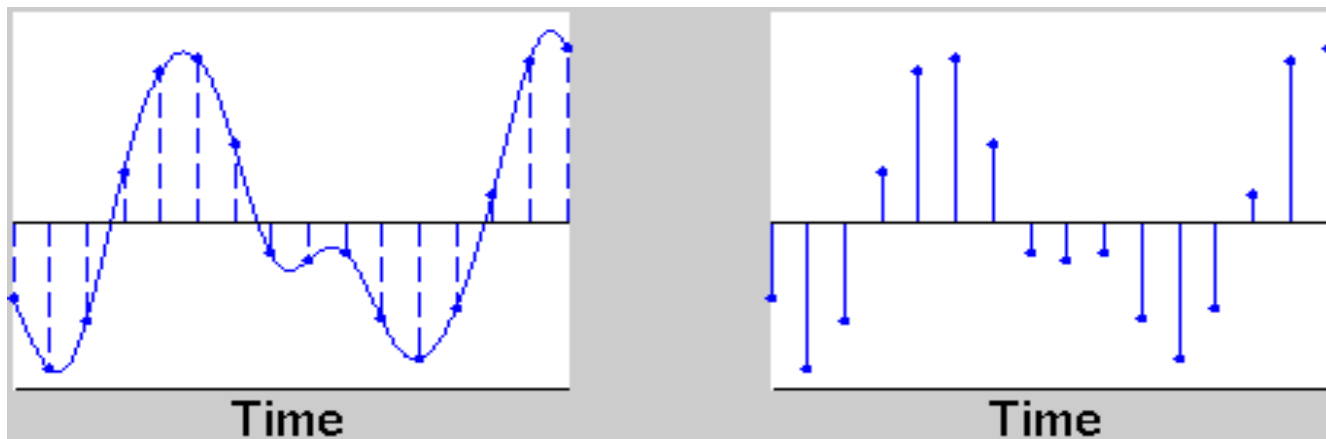
Advantages of digital (versus analog) signals:

- Guaranteed accuracy
 - defined e.g. by sampling rate, number of bits
- Perfect reproducibility (for signal / processing operations)
- Superior performance
 - some operations are practically impossible using analog parts
- Smaller size, lower cost, etc.

Sampling

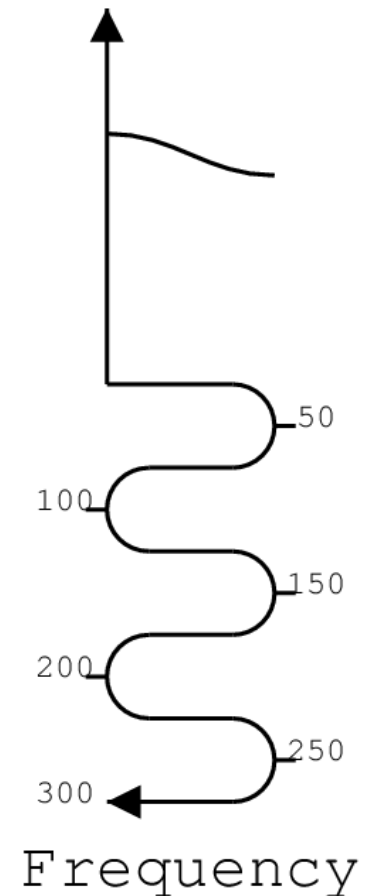
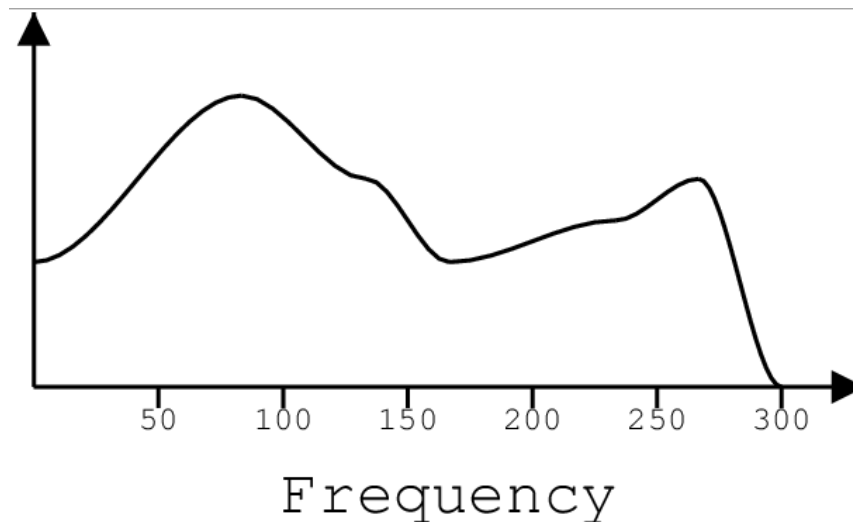
■ Sampling theorem:

- continuous signal can be replaced by a discrete sequence of samples *without losing information*, provided that the sampling frequency f_s is at least twice the highest frequency component in the signal
- original continuous-time signal can be reconstructed from the samples
- the frequency $f_s/2$ is called *Nyquist frequency*



Aliasing

- **Aliasing** occurs if the sampling rate is not sufficient
 - **Figure**: spectrum folds over itself if the sampling rate is 100Hz and the signal contains frequencies up to 300Hz
- Aliasing can be prevented by *lowpass filtering* the analog signal with cutoff $f_s/2$ before sampling

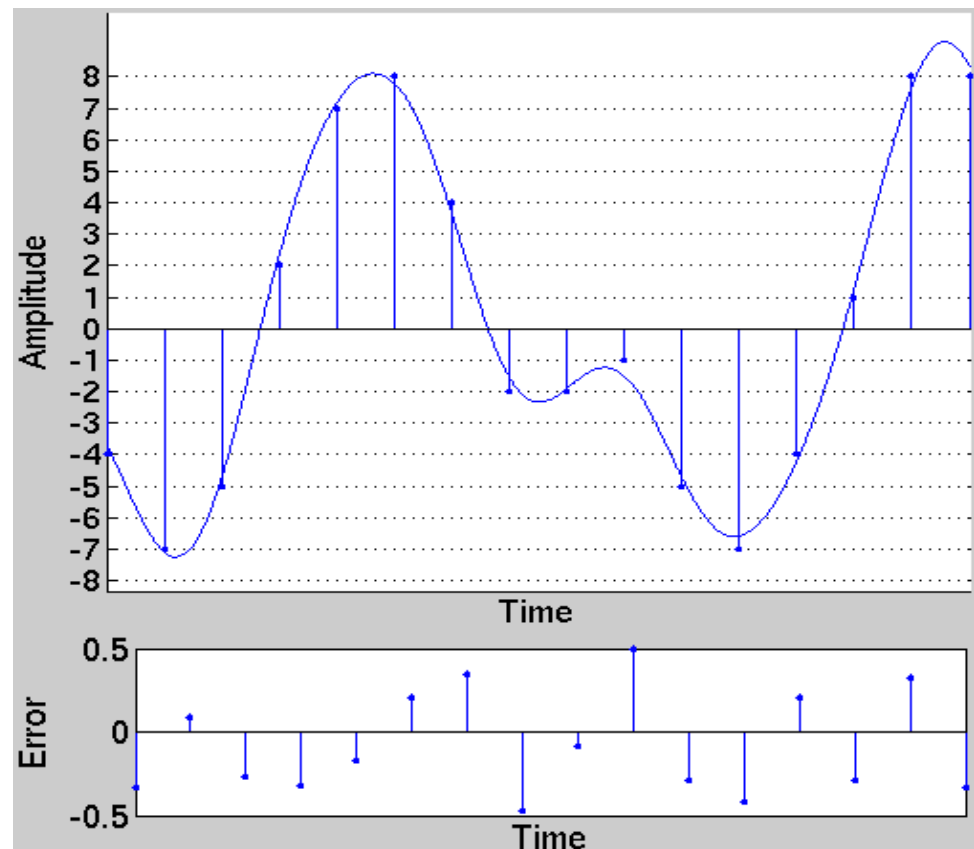


Quantization

- *Quantization* is another essential part of analog-to-digital conversion (along with sampling)
 - analog sample values (signal levels) are converted to (binary) numbers
 - both the sampling rate and sample values have a limited resolution
- Uniform quantization
 - analog values are mapped to a finite number of levels that are uniformly (linearly) distributed on the range of values used
 - 16 bits $\rightarrow 2^{16} = 65\,536$ levels (imagine a 5m pile of paper)
- Note: *sampling* of a bandlimited signal is theoretically lossless, but *quantization* is always lossy

Quantization error

- For binary numbers, *word-length* determines the quantization step size
 - for audio, 16 bits is usually enough
 - in mastering often 24 bits are used in order to avoid error cumulation
 - n bits $\rightarrow 2^n$ quantization levels
- Figure: *quantization error* is the difference between the original and the quantized value
 - the error is between $+Q/2 \dots -Q/2$, where Q is quantization step size
 - \rightarrow "quantization noise"

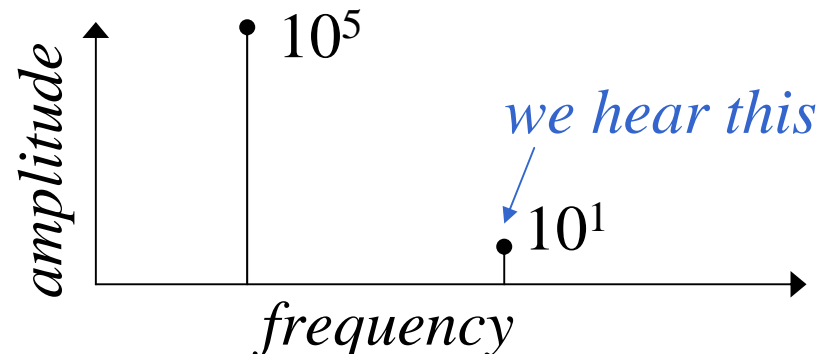


Precision requirements for audio signals

- **Dynamic range** of hearing is wide
 - the sound pressure level ratio of a barely audible vs. hardly tolerable sound is $1:10^5$ (\rightarrow ratio of powers $1:10^{10} \rightarrow 100\text{dB}$)
 - 16bits gives 98dB dynamic range (each additional bit 6dB more)
 - logarithmic level-scale (decibels, dB) is convenient

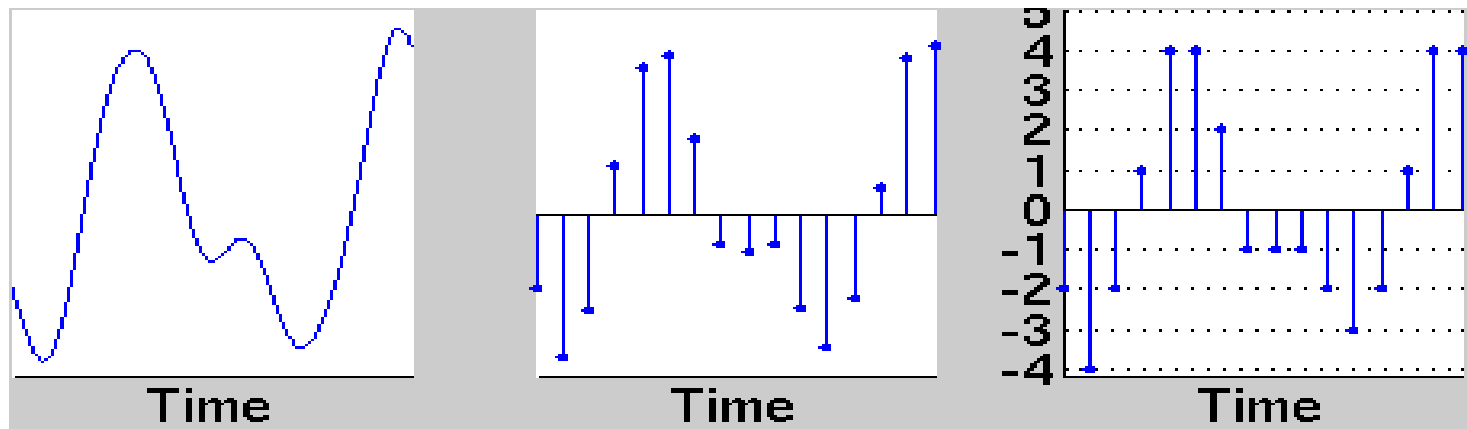
$$level_{dB} = 10 \log_{10}(\text{power})$$

- **Frequency range** of hearing differs between individuals
 - theoretically 20Hz – 20kHz
 - sensitivity below 100Hz is not very good
 - sensitivity above 12kHz degrades with age
- **Frequency selectivity** of hearing



Analog signal \rightarrow sampling \rightarrow quantization

- Figure: (a) analog signal, (b) discrete-time signal, (c) digital signal



- According to the common practice, we consider only discrete-time signals in the following
- Quantization is not specifically addressed
 - we assume a "sufficient" numerical resolution, for example "float" or "double" precision in C++
 - in embedded systems, such as mobile phones, typically fixed-point implementations have to be done (\rightarrow quantization is a big issue)

2 Short-time Fourier transform

- Fourier's theorem (part of)
 - any continuous waveform can be modeled as a sum of infinite number of sinusoids with different freqs, amplitudes, phases
 - *STFT: a finite segment of a discrete-time signal can be represented with a finite number of sinusoids*
- Short-time Fourier transform (STFT)

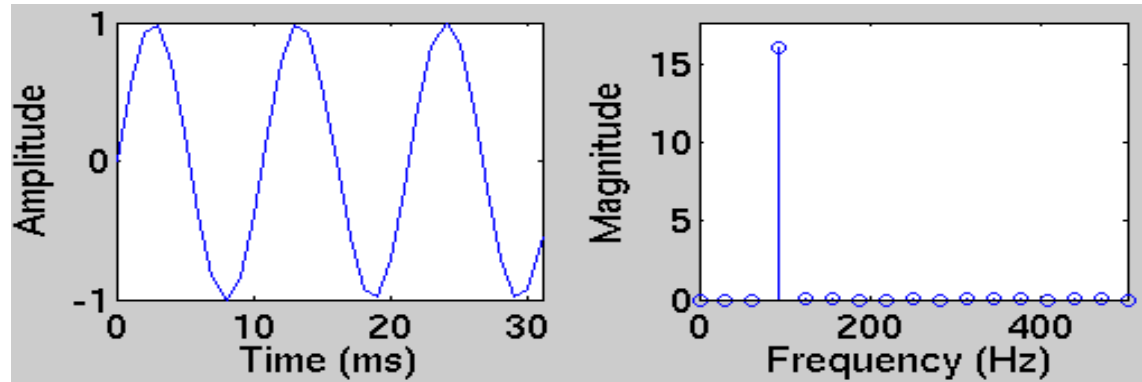
$$X(k) = \sum_{n=0}^{N-1} x(n)W^{nk}$$

where the constant

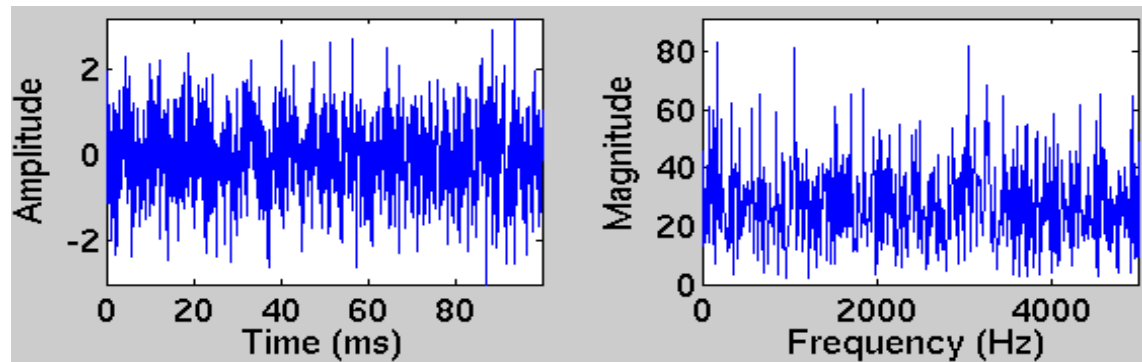
$$W = \exp(-i2\pi / N) = \cos(2\pi / N) + i \sin(2\pi / N)$$

Some Fourier transform pairs

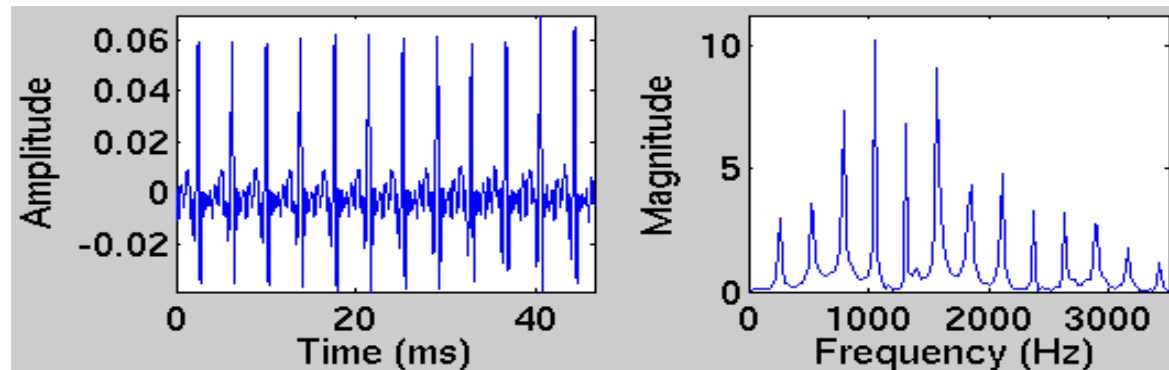
- Sinusoid:
peak in spectrum



- White noise:
flat spectrum



- Periodic sound
(trumpet):
"comb-like"
spectrum



Some properties of the Fourier transform

	Time-domain signal	Discrete Fourier transform
<i>(Notation)</i>	$x(n)$	$X(k)$
<i>Linearity</i>	$ax(n) + by(n)$	$aX(k) + bY(k)$
<i>Convolution</i>	$x(n) \otimes y(n)$ (convolution)	$X(k)Y(k)$
\leftrightarrow <i>multiplication</i>	$x(n)y(n)$ (e.g. windowing)	$X(k) \otimes Y(k)$
<i>Time shift</i>	$x(n+m)$	$W^{-km} X(k)$ (allows arbitrary time delays)

■ Parseval's theorem:
$$\sum_{n=0}^{N-1} x(n)^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2$$

→ power can be computed in either domain

→ power at a certain frequency band can be easily computed

3 Spectrum estimation

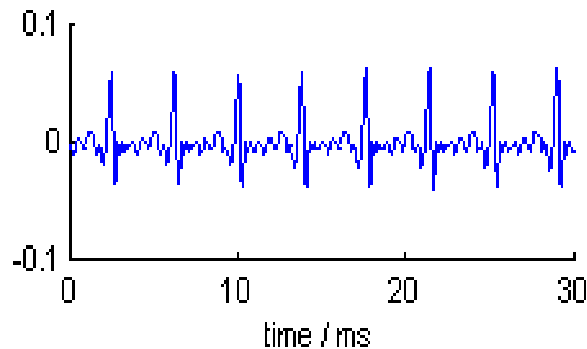
- Spectrum of audio signals is typically estimated in short consecutive segments, *frames*
- Why?
 - the Fourier transform models the signal with *stationary sinusoids* (constant spectrum)
 - real audio signals are not stationary but vary through time
 - framewise processing assumes the signal is time-invariant in short enough time frames
- For audio signals, the frame length typically varies between 20ms – 100ms, depending on the application
 - for speech signals often 25ms
 - multipitch analysis in polyphonic music: often around 100ms
- Transient-like sounds are difficult to represent and process in the frequency domain → time blurring

Windowing

- **Windowing** is essential in frame-wise processing
 - weight the signal with a window function $w(k)$ prior to transform
 - as a rule of thumb, windowing is always needed: one cannot just take a short part of a signal without windowing

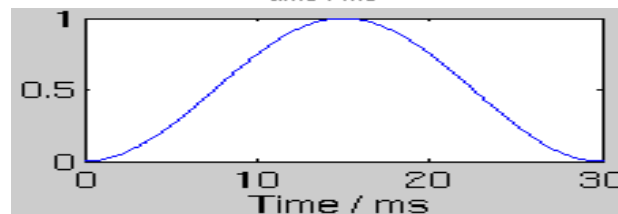
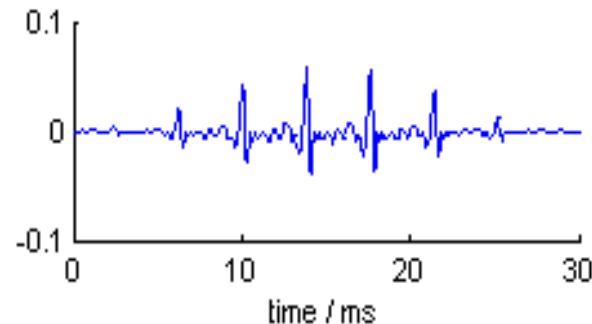
signal in frame m :

$$x_m(n), n = 0, \dots, N-1$$



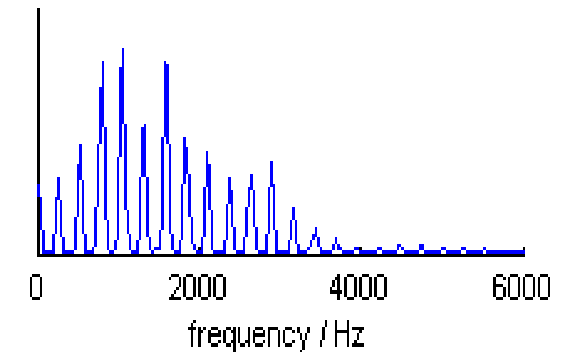
windowed signal:

$$x_m(n)w(n)$$



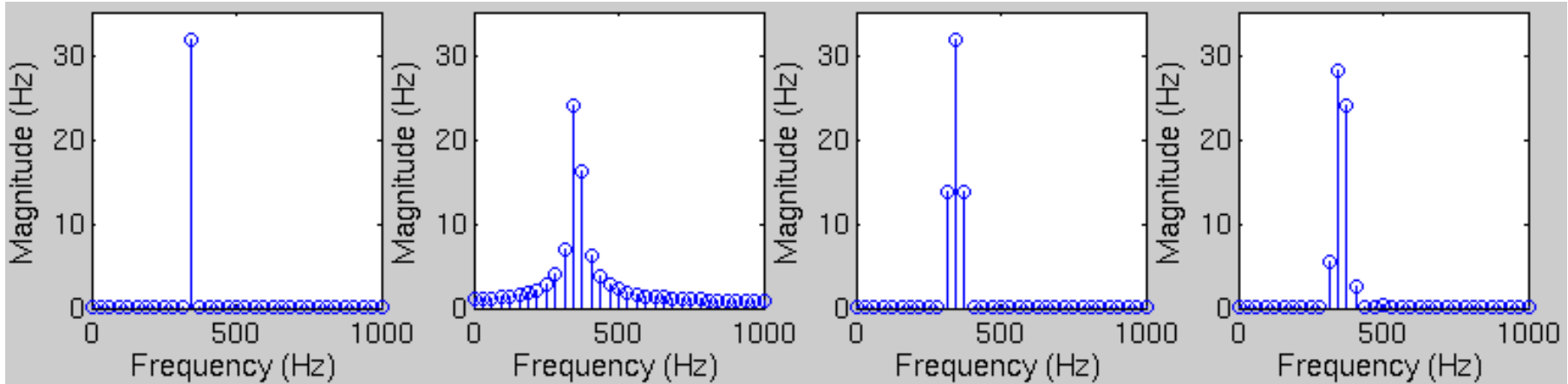
short-time spectrum:

$$X_m(k) = \sum_{n=0}^{N-1} x_m(n)w(n)W^{nk}$$



Windowing

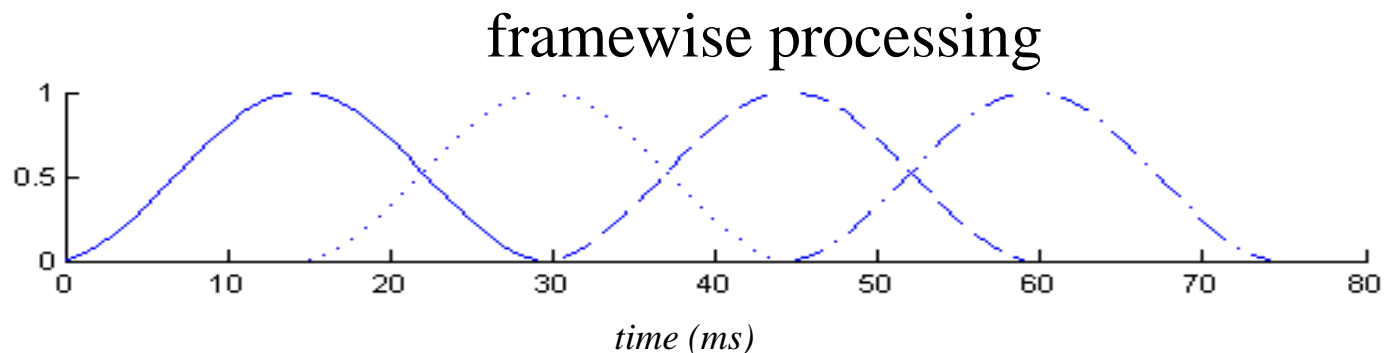
- Example: *spectrum of a sinusoid* with/without windowing
 1. No windowing (=rectangular window), sinusoid at a spectral bin
 2. No windowing, random off-bin frequency → *spectral blurring!*
 3. Hanning window, sinusoid at a spectral bin
 4. Hanning window, random off-bin frequency → *ok*
- There are different types of windows, but most important is not to forget windowing altogether



Windowing in framewise processing

■ Figure: Hanning windows

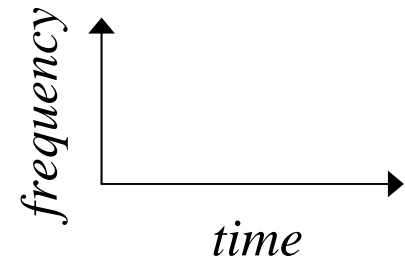
- adjacent windows sum to unity when frames overlap 50%
- all parts of the signal get an equal weight



- In each frame, the signal is weighted with the window function and short-time discrete Fourier transform is calculated

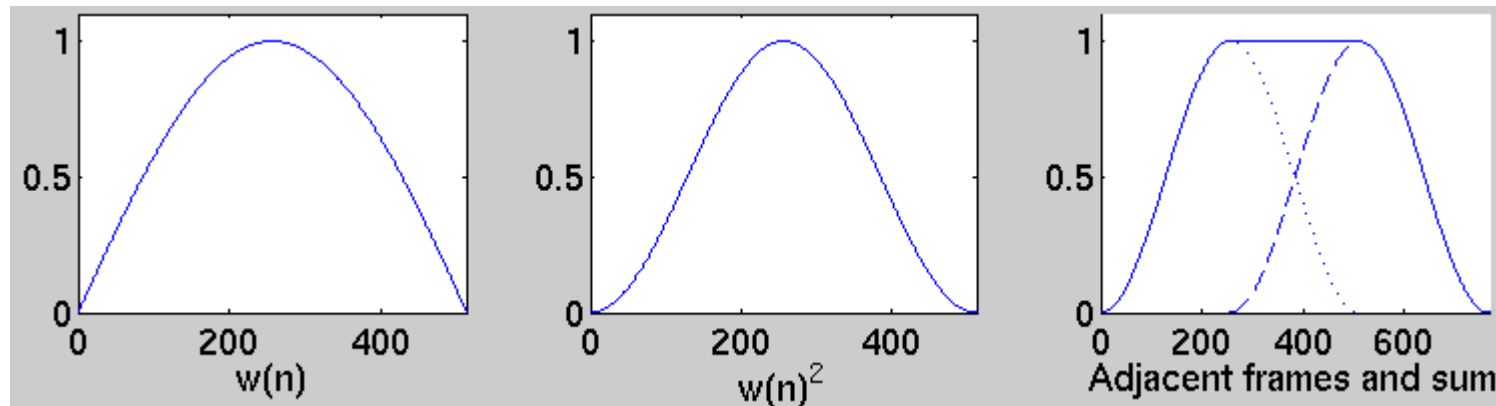
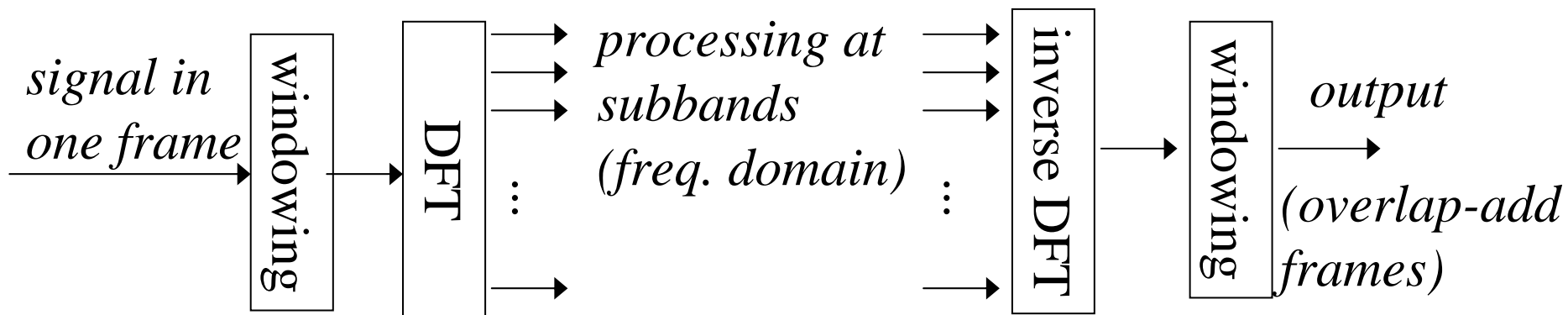
■ This yields a *spectrogram*

- time-frequency representation:
complex spectrum in each frame over time



Windowing in analysis-synthesis systems

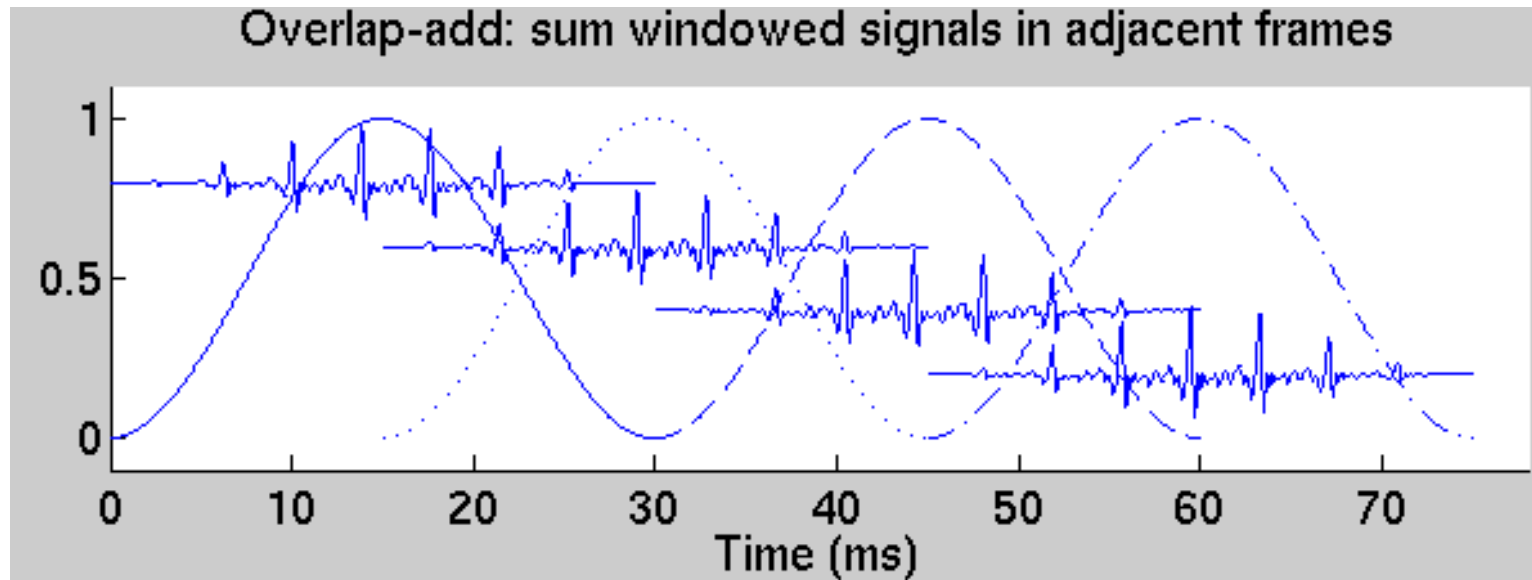
- *Sine window* is useful in analysis-synthesis systems (see [Figure](#))
- Windowing is done again in resynthesis to avoid artefacts at frame boundaries in the case that the signal is manipulated in the f -domain
 - [Figure](#) below: 50% frame overlap leads to perfect reconstruction if nothing is done at subbands



Reconstructing the time domain signal from its spectrogram

■ Overlap-add technique:

1. inverse Fourier transform the spectrum of each frame back to time domain
2. apply windowing in each frame (e.g. sine / Hanning window)
3. successive frames are positioned to overlap 50% or more, and summed sample-by-sample

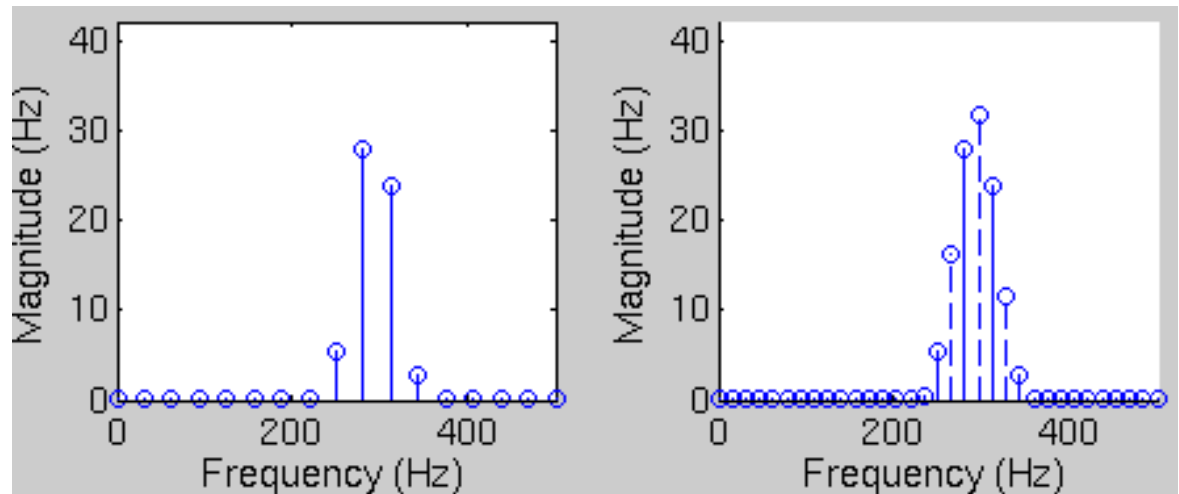


Zero padding

- Zero padding is used
 - to improve the frequency resolution in short time frames
 - to constraint the transform length to 2^n (n integer) for FFT (fast F. transf.)
- Zero padding = sequence of zeros, $\mathbf{0}$, is concatenated with a windowed time-domain signal before the transform takes place

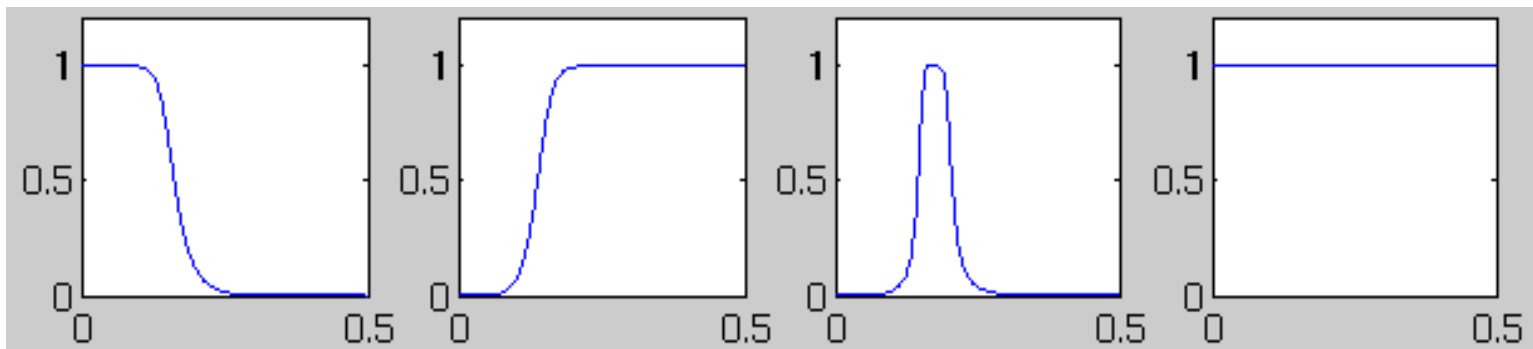
$$\tilde{x}(n) = \begin{bmatrix} x(n)w(n) \\ \mathbf{0} \end{bmatrix} \Rightarrow \tilde{X}(k) = \sum_{n=0}^{N-1} \tilde{x}(n)W^{nk}$$

- **Figure:** spectrum of a sinusoid (left) and the same using zero-padding with factor 2 (right)



4 Filters

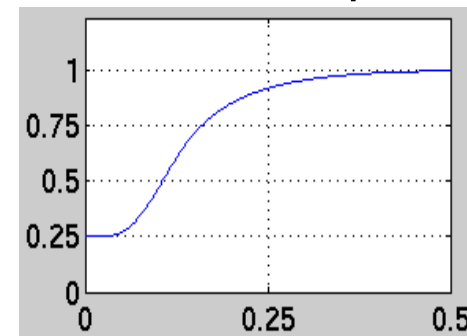
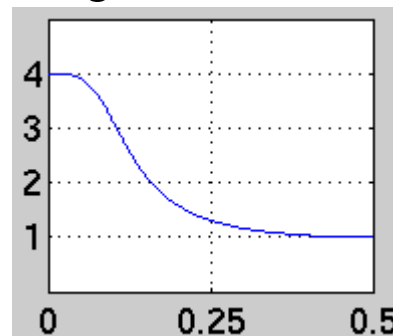
- In a typical filtering task, the aim is to *suppress* certain frequencies and to *retain* other frequencies
- Figure: magnitude responses as a function of normalized frequency
 - (a) lowpass, (b) highpass, (c) bandpass, and (d) allpass responses



- Another typical filtering task is to *boost* or *cut* certain frequencies



bass



Implementing filters in practice

- Typical filtering problem:
 - "weight each frequency k with a desired frequency response $H(k)$ "
- Property of the Fourier transform: weighting (multiplication) in the frequency domain, $H(k)X(k)$, is equivalent to a *convolution in the time domain*, $h(n) \otimes x(n)$
- This leads directly to so-called finite impulse response (FIR) filters
 - $h(n)$ is called the *impulse response* of the filter

4.1 Finite impulse response (FIR) filters

- Output = convolution of a signal with the *impulse response* of the filter

$$y(n) = h_0 x(n) + h_1 x(n-1) + \dots + h_K x(n-K)$$

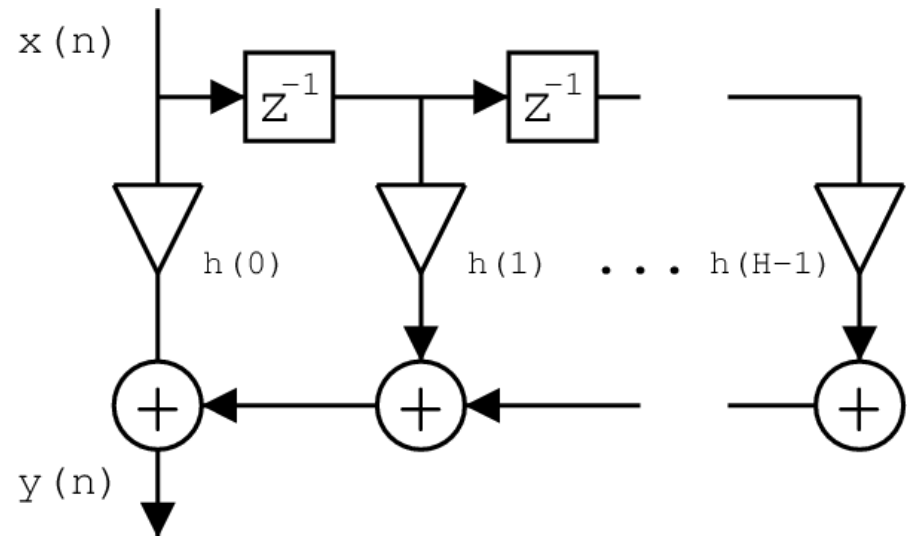
$$= \sum_{k=0}^K h_k x(n-k)$$

- Figure: Output is a *sum of delayed and scaled inputs*
- Example impulse response:

$$\mathbf{h} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\Rightarrow y(n) = 0.5x(n) + 0.5x(n-1)$$

which performs averaging
(= simple lowpass filtering)



Transfer function

- An important tool in studying discrete-time systems is the *Z-transform*
- Z-transform of a signal $x(n)$ is the series

$$X(z) = \dots + x(-2)z^{-2} + x(-1)z^{-1} + x(0)z^{-0} + x(1)z^1 + \dots$$

- FIR filter output:

$$y(n) = \sum_{k=0}^K h_k x(n-k)$$

Z-transform:

$$Y(z) = \sum_{k=0}^K h_k X(z)z^{-k} = X(z) \sum_{k=0}^K h_k z^{-k} = X(z)H(z)$$

where

$$H(z) = \sum_{k=0}^K h_k z^{-k}$$

is called the *transfer function* of the filter

Frequency response

- *Frequency response* of a filter is obtained by substituting

$$z = \exp(-i2\pi k / N) = \exp(-i\omega)$$

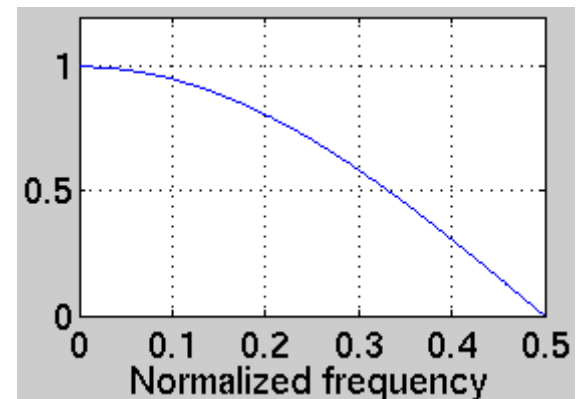
to the transfer function

(Z-transform is a generalization of the Fourier transform)

- Consider the example with impulse response $\mathbf{h} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$
 $\Rightarrow y(n] = 0.5x(n) + 0.5x(n-1)$

- Transfer function of this filter is $H(z) = 0.5 + 0.5z^{-1}$

- Frequency response is $H(z)$, $z = -i\omega$:
 - as predicted, it is a lowpass filter



How to implement a desired frequency response?

- Several software packages exist for filter design
 - pen and paper are seldom needed
- For example the Matlab routine `remez (. . .)`
 - designs an optimal FIR filter given (a) the filter order and (b) an arbitrary piece-wise linear frequency response
- Most important is to understand the *design criteria* and general properties of different filters

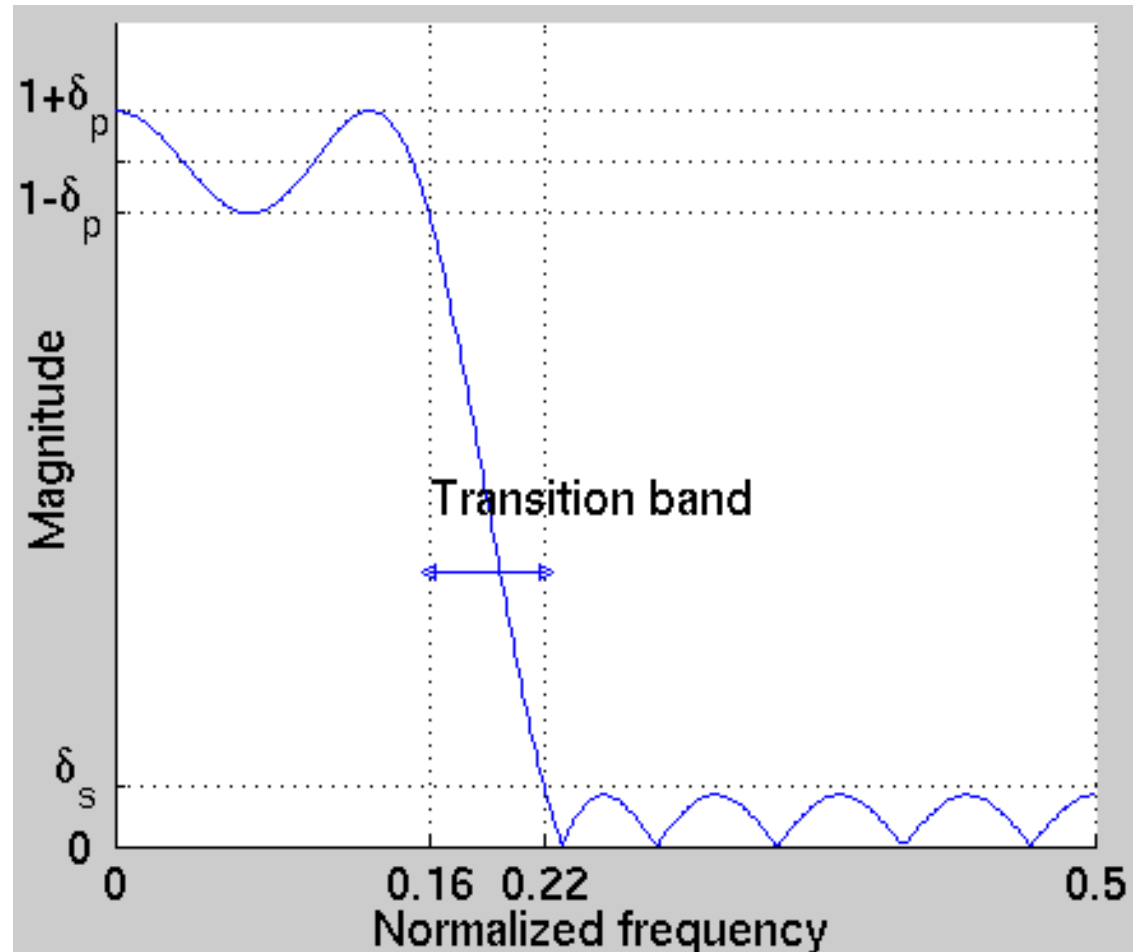
Filter design criteria

■ Basic design requirements for the magnitude response:

- passband ripple δ_p
- stopband ripple δ_s
- transition band width

■ For audio signals

- passband ripple often $<1\text{dB}$ (ripple <0.1)
- stopband attenuation often $>60\text{dB}$ (ripple <0.001)

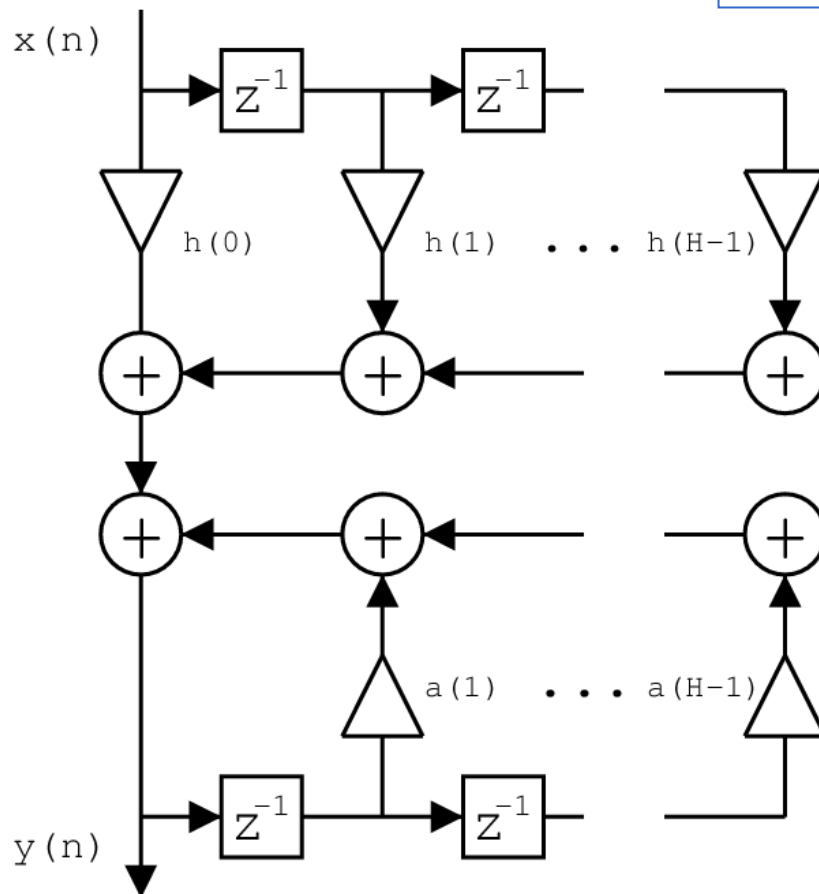


Main drawback of FIR filters

- A large number of filter coefficients is needed to implement good filters
 - small ripple at passband and stopband
 - steep transition bands
- This leads to infinite impulse response (IIR) filters

4.2 Recursive (IIR) filters

- Difference equation: $y(n) = \sum_{k=0}^K h_k x(n-k) + \sum_{m=1}^M a_m y(n-m)$



→ Delayed and scaled *output* is fed back and summed, too

IIR filters – transfer function

- Difference equation: $y(n) = \sum_{k=0}^K h_k x(n-k) + \sum_{m=1}^M a_m y(n-m)$

- Z-transform

$$Y(z) = \sum_{k=0}^K h_k X(z) z^{-k} + \sum_{m=1}^M a_m Y(z) z^{-m}$$

$$Y(z) \left[1 - \sum_{m=1}^M a_m z^{-m} \right] = X(z) \sum_{k=0}^K h_k z^{-k}$$

from which the transfer function

$$H(z) = \frac{X(z)}{Y(z)} = \frac{\sum_{k=0}^K h_k z^{-k}}{1 - \sum_{m=1}^M a_m z^{-m}}$$

Designing IIR filters

- Again, good software packages exist for IIR filter design
- Often some standard "off the shelf" filter type is used
 - e.g. Matlab routines `butter` (Butterworth filter), `ellip` (elliptic filter), `cheby1`, `cheby2` (Chebyshev filters)
 - just determine the filter order and the filter type

Properties of some IIR filters

- **Comparison:** sixth-order bandpass filter
 - solid: Butterworth, - - - dashed: elliptic filter

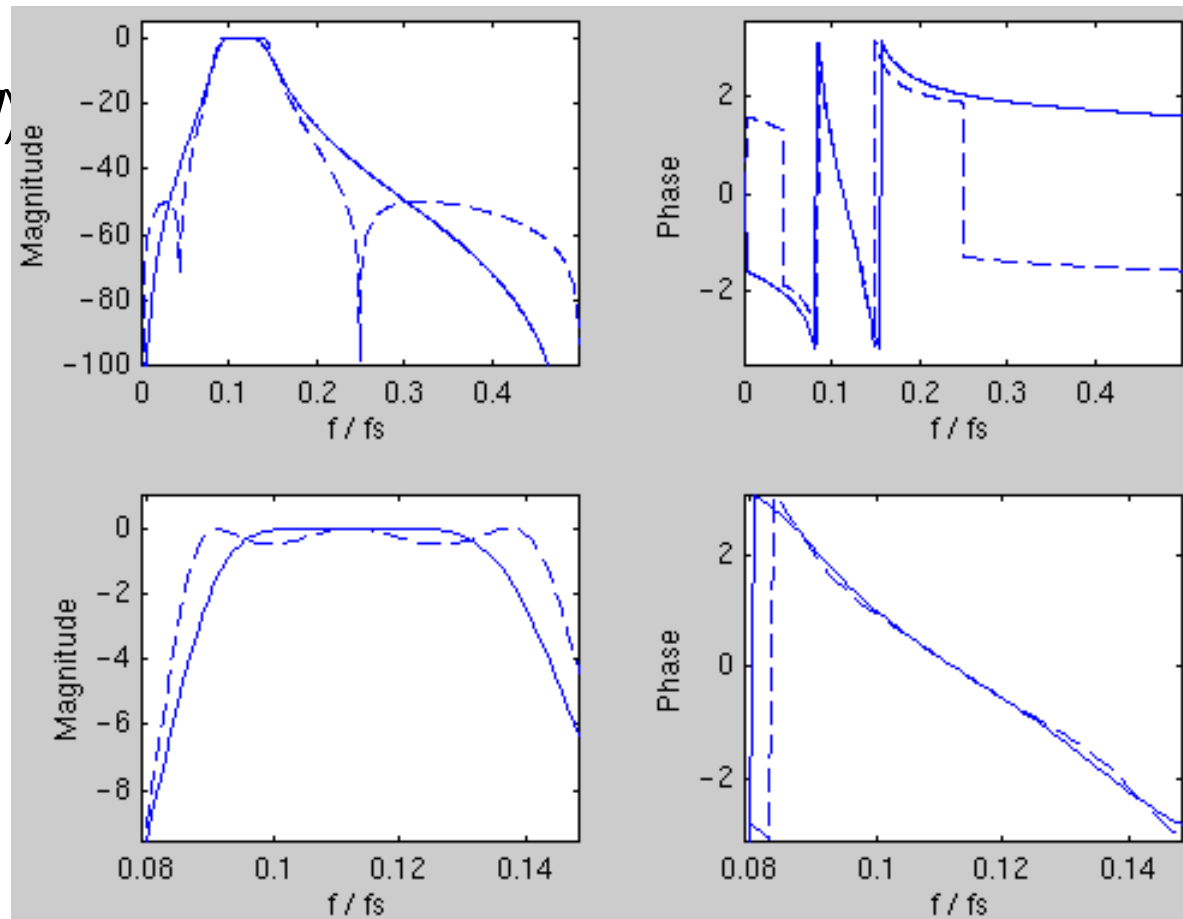
(Magnitudes in dB!)

TOP

- magnitude resp.
- phase response

BOTTOM

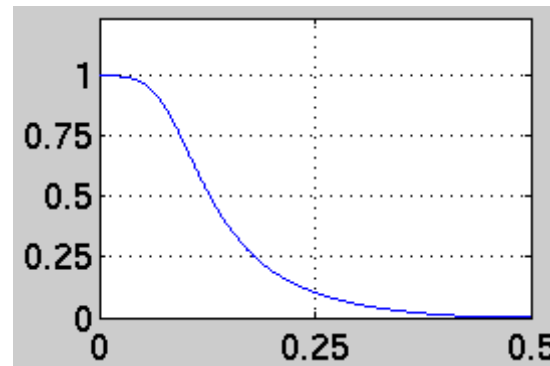
- magnitude resp.
zoomed to the passband
- phase resp.
zoomed to the passband



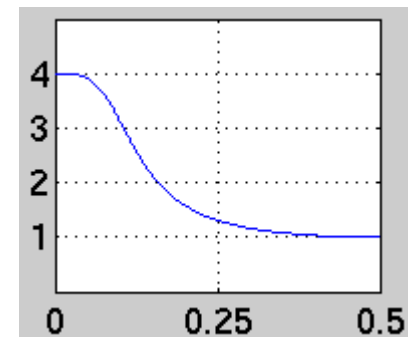
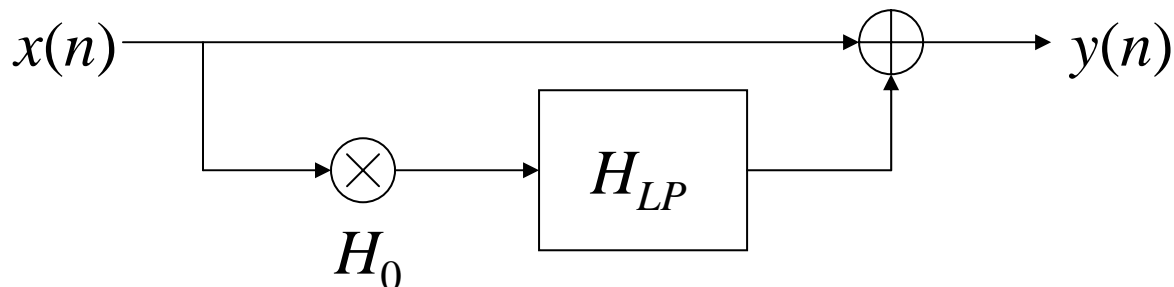
Example: "Boost" low frequencies

- Butterworth-type IIR filters are well-suited for this task because
 - steep transition band is *not* desirable, but smoothly-varying response
 - Butterworth response is maximally flat at passband
 - phase response at passband is near to linear: waveform shape preserved

- Frequency response of a second-order Butterworth filter:



- *A shelving filter*



4.3 Choosing between FIR and IIR

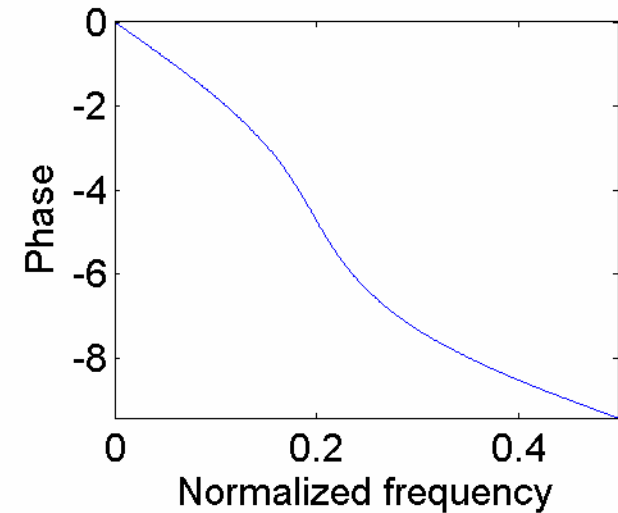
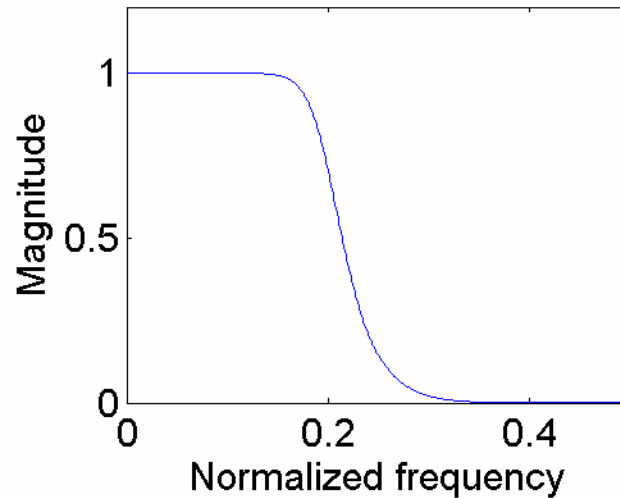
Some important considerations:

- IIR filters are computationally *more efficient*
 - small number of filter coefficients achieve steep filters
- FIR filters allow *linear phase response*
 - = at the passband, signal is a delayed copy of the original
 - human auditory system is *not* very sensitive to phases (\neq vision)
- FIR filters allow *more perfect control of the response*
 - for example, perfect reconstruction analysis-synthesis filterbands
 - facilitates the design of other parts e.g. in an audio codec
- FIR filters can be guaranteed to be *numerically stable*

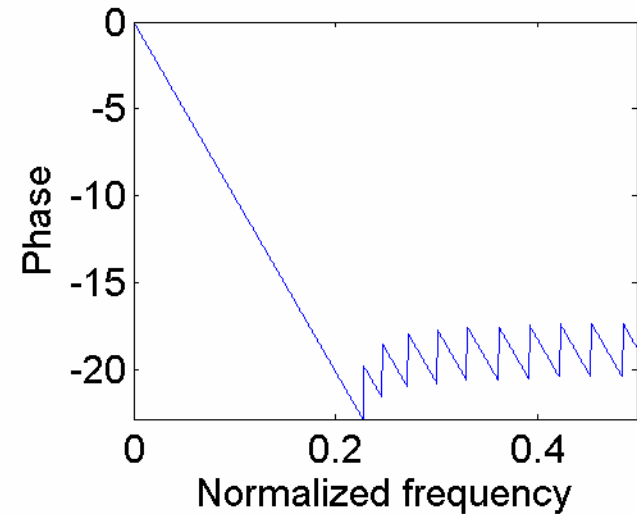
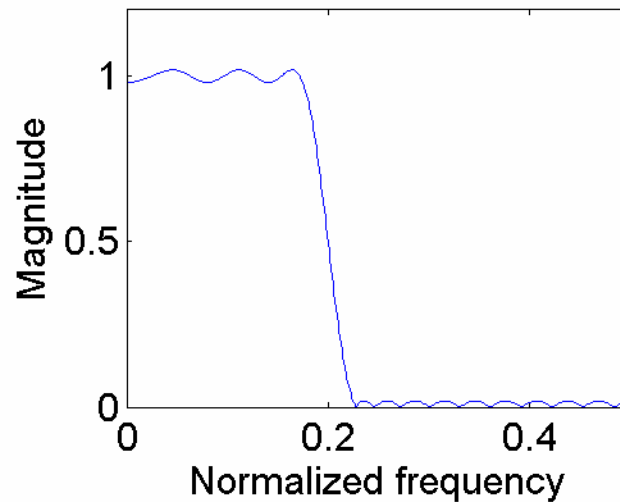
→ Choice depends on the application

Phase responses (right-hand panels)

- IIR:
(6th order)



- FIR:
(32th order)



Phase response

- The phase response differences of the above filters are really not audible, so why bother?
- Exactly linear phase response facilitates the design of complex systems
 - example: several subbands → avoid artefacts at band boundaries if all frequency components are delayed by exactly the same amount
- A *noncausal* trick to achieve exactly linear phase response with IIR filters (Matlab "filtfilt")
 - filter a signal
 - reverse the signal and filter it again
 - reverse the signal again
 - *phase response is zero* to all frequencies! (no delay at all)

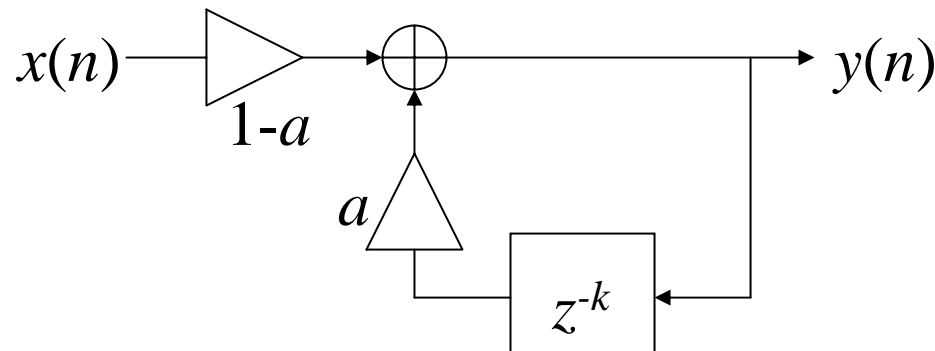
4.4 Comb filters

- Transfer function of a comb filter is of the form

$$H(z) = \frac{1-a}{1-az^{-k}}$$

where

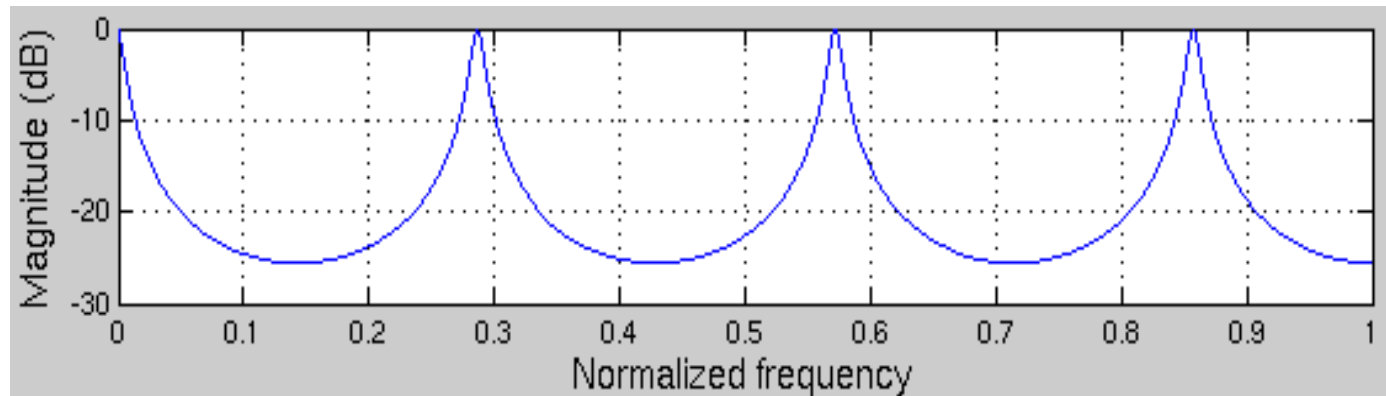
- a determines the feedback gain ($a < 1$ to be stable)
- k determines the delay
- $(1 - a)$ in the numerator normalizes the maximum response to 0dB



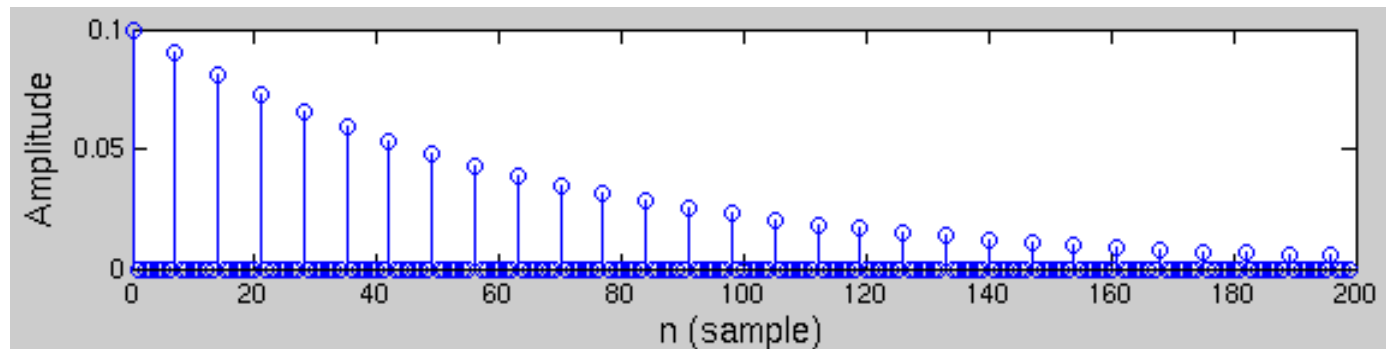
- Comb filters have many uses in audio-DSP: for example in reverberation modeling, in audio effects, in digital resonators, and in period estimation (rhythm, pitch)

Comb filter

- Let us do a comb filter in Matlab
 $a=0.9$; $k=7$;
 $B=1-a$; $A=[1 \text{ zeros}(1,k-1) -a]$;
- Frequency response ($\text{freqz}(B,A)$) looks like a "comb"



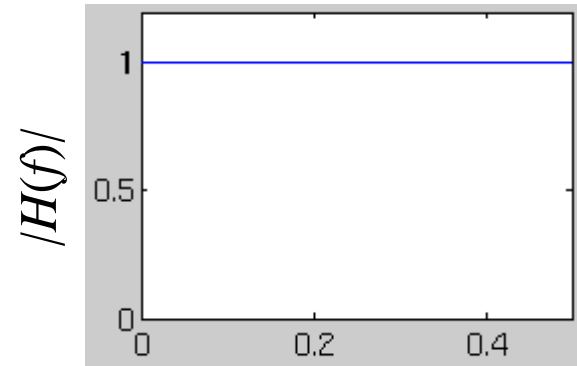
- Impulse response ($\text{impz}(B,A)$) is a geometric series with zeros in between



4.5 Allpass filters

- Definition of allpass filters: magnitude response is exactly one at all frequencies

$$|H(f)| = 1$$



Normalized frequency

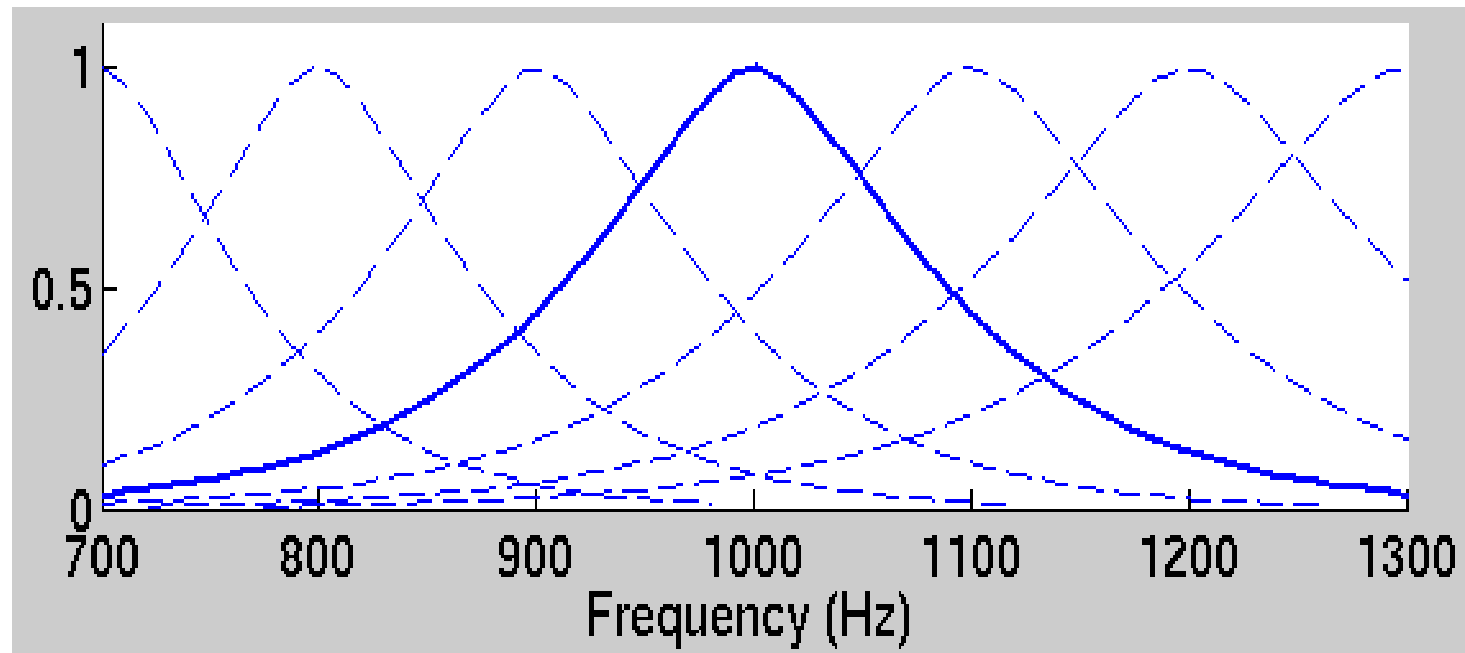
- *By manipulating the phase response*, allpass filters can achieve certain things. Some uses are e.g.:
 - "phaser" and "flanger" sound effects
 - implementing arbitrary time delays (\neq multiple of sampling period): Fourier transform has the property that

$$x(n + m) \leftrightarrow W^{-km} X(k)$$

where the delay m does not need to be an integer

4.6 Auditory filters

- *Auditory filters* are used to model the frequency selectivity of the human auditory periphery
- Desired response at the passband is not flat



4.7 Taking *quantization* into account

- Limited wordlength (numerical precision) leads to different types of quantization errors
 - *wordlength*: how many bits are used to represent one sample of one filter coefficient
- Quantization of the *filter coefficients*
 - causes distortion which appears as a deviation from the ideal frequency response (small and controllable problem)
- Quantization of signal values within an IIR filter (*filter state*)
 - IIR filter utilizes feedback...
 - quantization determines the maximal dynamic range
 - rounding errors within the filter are fed back to the input
- Practical tip for e.g. C++ implementation:
the state of an IIR should be represented with "double" precision even though the signal would be "float" or "byte"