JOHANNES KEPLER
UNIVERSITY LINZ

SPECIAL TOPICS

Audio and Music Processing - Lectures 2–3:
Analyzing Sound and Music
344.032
KV, 2h, SS2020

Jan Schlüter
Institute of Computational Perception

GOALS

- provide **enough** knowledge about digital signal processing (DSP) to follow the rest of this lecture series
- what are signals? how are they represented?
- time domain / frequency domain
- foundations of DSP
- correlation / autocorrelation
- convolution
- filtering
- discrete fourier transform

SOURCE(S)

most of the material for this lecture was taken from the **freely** available, **easily** approachable and quite frankly **excellent** “The Scientist and Engineer’s Guide to DSP” by Steven W. Smith.

BASICS

WHAT ARE SIGNALS?

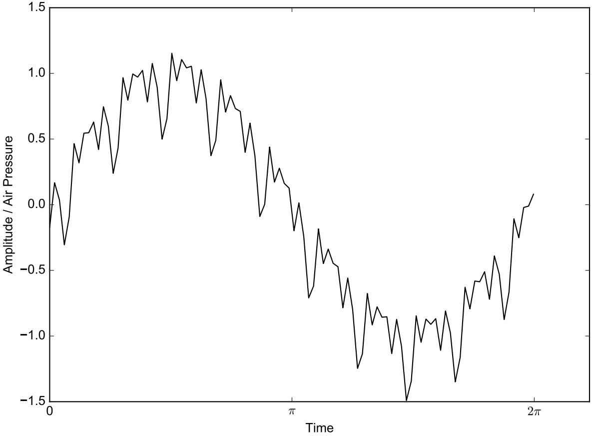
- a **signal** is any **time-varying** or **spatial-varying** quantity
- a **signal** is an **information bearing** function
- examples include: motion, images, videos, temperature, . . .
- and **sound**: variation in **air pressure** at a point in space, as a function of time

BASICS OF SOUND SIGNALS

- listeners hear sound because the air pressure changes slightly in their ears
- if the **pattern** of pressure changes **repeats**, the sound has a **periodic waveform**
- if there is **no discernible pattern** we call that **noise**
- one repetition of a periodic waveform is a **cycle**
- the length of the cycle is the **wavelength**
- number of repetitions per second is the **fundamental frequency** of the waveform
- if the **wavelength increases**, the **frequency decreases** and vice versa
- the measure for **cycles per second** is **Hertz** (Hz)

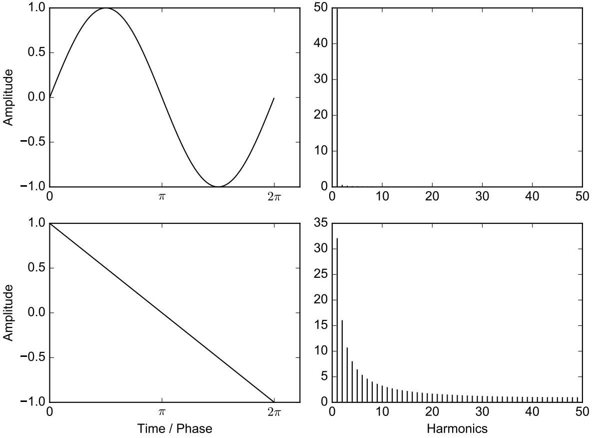
TIME DOMAIN

the simplest method to depict sound waveforms is to graph the **air pressure** versus **time**



FREQUENCY DOMAIN

we can look at waveforms in the **frequency domain** as well

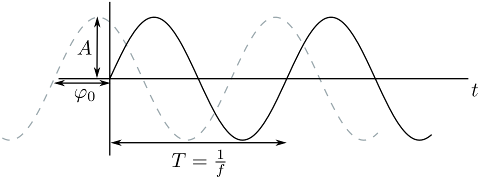


▷ sinus

▷ sawtooth

PHASE

$$\begin{aligned} y(t) &= A \cdot \sin(\varphi(t)) \\ \varphi(t) &= \int_0^t 2\pi f(t) dt + \varphi_0 \\ \varphi(t) &= 2\pi f \cdot t - 2\pi f \cdot 0 + \varphi_0 \text{ iff } f(t) = const. \\ y(t) &= A \cdot \sin(2\pi f t + \varphi_0) \end{aligned}$$

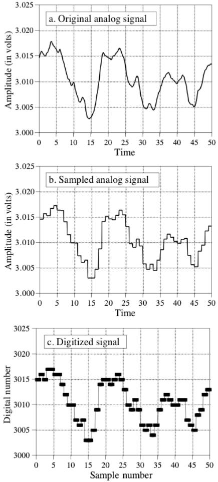


the **initial phase** φ_0 is the **starting point** of a periodic waveform on the y-axis
(in general, the phase is insignificant to the human ear)

ANALOG TO DIGITAL (1)

- **signals** in the **real world** are **continuous**
  - **infinite number of points** in a given period of time
  - **infinite number of possible values** at a point
- a **finite** computer **cannot** interact with **continuous** signals
- we need to **transform** the **analog signal** into a **digital signal**
- a **digital** signal is only an **approximation** of the **analog** signal

ANALOG TO DIGITAL (2)

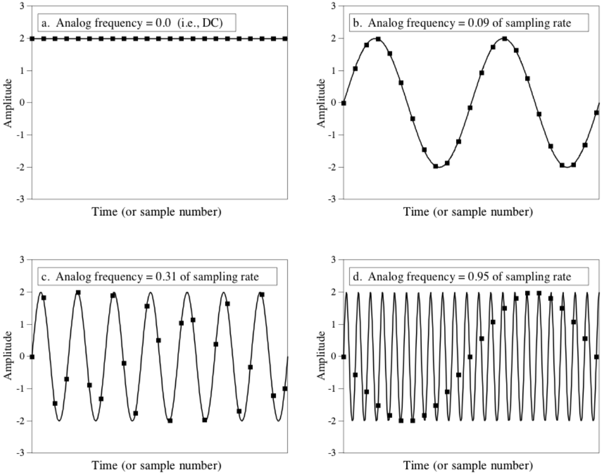


- **digitization** is a **two-step** process
- **discretization** / **sampling** **sample** an analog signal at **regular time intervals**
- **quantization** / **rounding** **round** each sample to a **fixed set** of values

THE SAMPLING THEOREM

- the **nyquist-shannon sampling theorem** [7] says:
  "if a function $f(t)$ contains no frequencies higher than W cycles per second, it is completely determined by giving its ordinates at a series of points spaced $\frac{1}{2W}$ seconds apart"
- this means that a signal can be **properly sampled**, if it contains **no frequencies above half** the **sampling rate**
- from a **properly sampled** signal we can **reconstruct** the **original** continuous sampled signal
- **nyquist frequency** = $\frac{\text{sampling rate}}{2}$
- most common sample rate ("CD quality") is 44100 Hz,
  nyquist frequency is 22050 Hz

PROPER SAMPLING

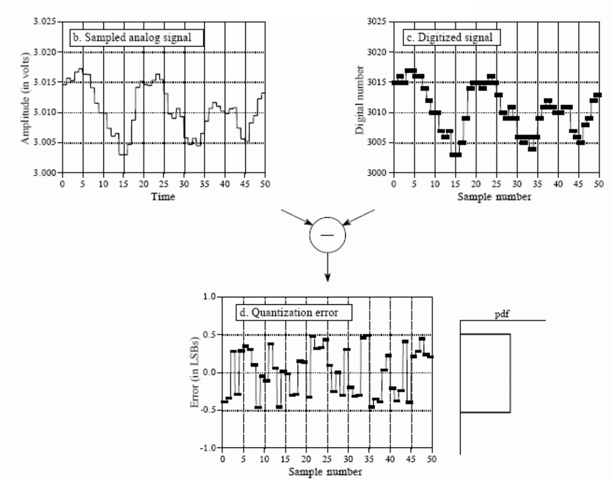


▷ proper (a, b, c) ▷ improper (d) see also [4] and ▷

QUANTIZATION

- unfortunately, there is **no proper quantization**
- it is **impossible** to **reconstruct** the **original** values after the signal has been quantized
- maximum error introduced is $\pm\frac{1}{2}$ LSB (least significant bit)
- the LSB is the **distance** between **adjacent** quantization levels
- **precision** depends on **number of bits** used for representation
- the **error** introduced looks like **uniform random noise**

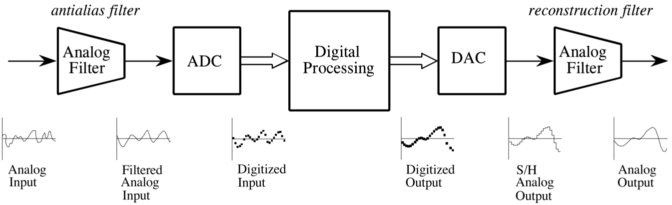
QUANTIZATION ERROR



JKU

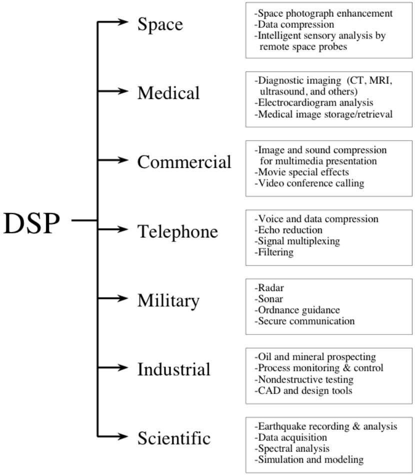
ANALOG FILTERING

- to **properly convert** an analog signal, we have to **filter** out frequencies **higher** than the sampling rate
- we use a **low pass** filter to let only frequencies be**low** a **cutoff** value **pass**
- we need to do this with an **analog filter**
- the details are left to the appropriate lecture [6]



DIGITAL SIGNAL PROCESSING

DIGITAL SIGNAL PROCESSING



- **representing** real world signals as a **sequence of numbers**
- **sequence processing** with a computer
- DSP has revolutionized many areas in science and engineering
- from now on, this lecture is only concerned with **digital signals**

LINEAR SYSTEMS

- we will be mainly concerned with **linear** systems
- examples for (approximately) linear systems include
  - wave propagation of sound waves
  - electronic amplifiers and filters
  - signal changes such as echo, reverberation and resonance
  - differentiation (finite differences)
  - integration (running sums)
  - convolution
  - correlation
  - ...

NOTATION AND TERMINOLOGY

- in DSP literature **discrete signals** are commonly denoted as $x[n]$ where x is the **signal** and n is the **time index** (in samples) into the signal
- most of the time $x[n]$ is a **source**, and $y[n]$ the **result**
- **offsets** in the signal are denoted as $x[n \pm k]$
- a **system** is any **process** that generates an **output** signal in **response** to an **input** signal
- we can **roughly** equate a **system** with a **function**

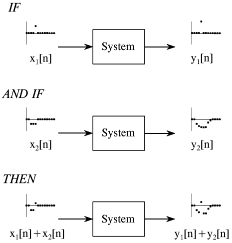
LINEAR SYSTEM PROPERTIES

assuming that f, g are linear systems, k, s are constants, and $x[n], y[n], x_1[n], x_2[n]$ are some discrete signals, they have the following properties:

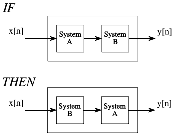
- **commutativity**: $f(g(x[n])) = g(f(x[n]))$
- **homogeneity**: $f(k \cdot x[n]) = k \cdot f(x[n])$
- **additivity**: $f(x_1[n] + x_2[n]) = f(x_1[n]) + f(x_2[n])$
- **shift invariance**: $f(x[n]) = y[n], f(x[n+s]) = y[n+s]$

PROPERTIES ILLUSTRATED

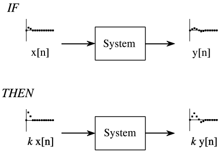
additivity



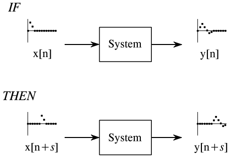
commutativity



homogeneity



shift invariance



SUPERPOSITION

- given a **linear** system f
- an **input** $x[t]$ which is a **sum of signals**

$$x[t] = \sum_{k=1}^{K} x_k[t]$$

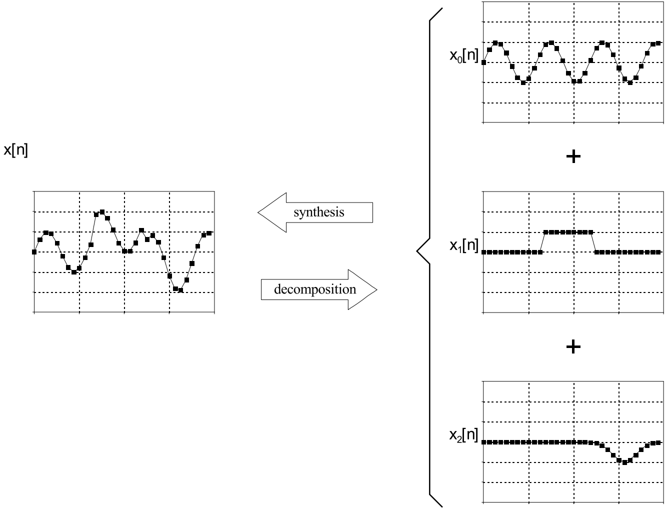
- the **response** $y[t]$ will be the **sum** of the **individual** responses

$$y[t] = \sum_{k=1}^{K} f(x_k[t])$$

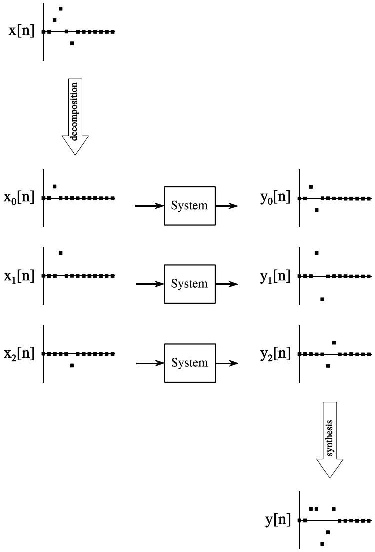
STRATEGIES

- most DSP techniques utilize **divide-and-conquer** strategies
  - **break** the signal into **simpler components**
  - **process** each component **individually**
  - **recombine** individual **results**
- **synthesis**: multiple signals are added to form another signal
- **decomposition**: break one signal into multiple components

SYNTHESIS AND DECOMPOSITION



THE FUNDAMENTAL CONCEPT

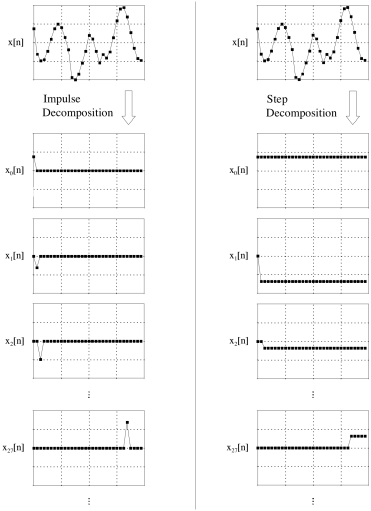


- any signal $x[n]$ can be decomposed into a group of additive components
- 3 components in this example: $x_1[n], x_2[n], x_3[n]$
- passing these components through a linear system produces $y_1[n], y_2[n], y_3[n]$
- adding these output signals, we form $y[n]$

SIMPLE SIGNALS

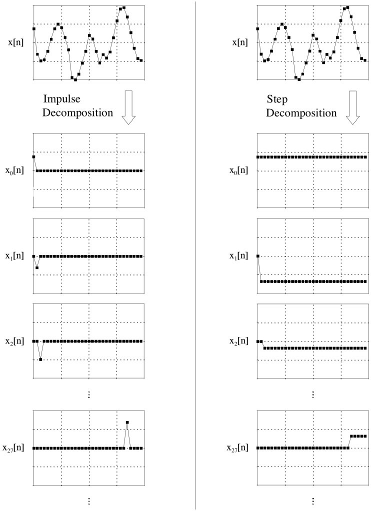
- **instead** of trying to understand how **complex** signals are changed by **complex** systems, **all we need to know** is how **simple** signals are changed
- **input** and **output** signals are a **superposition (sum)** of **simple**r signals
- we can **decompose** the complex signal into simpler ones
- this is **the basis** for nearly all signal processing techniques

COMMON DECOMPOSITIONS



- **impulse** decomposition: a signal with N points is broken into N components, each consisting of **one** single nonzero point - the impulse
- **step** decomposition: a signal with N points is broken into N components, each being a **step function**
- **fourier** decomposition: we will look at it in some detail later on . . .

COMMON DECOMPOSITIONS



- each of the three decompositions is connected to a way of describing a linear system
- a linear system is fully defined by
  its **impulse response**,
  its **step response** or
  its **frequency response**

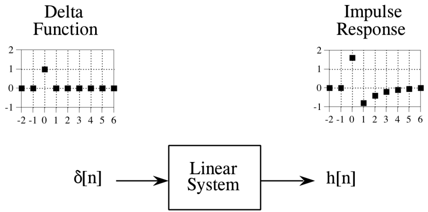
DELTA FUNCTION

- the **discrete delta function** $\delta[n]$ is also called **Kronecker Delta**
- this **unit impulse**, or **normalized** impulse is defined as follows:

$$\delta[n] = \begin{cases} 0 & : n \neq 0 \\ 1 & : n = 0 \end{cases}$$

IMPULSE RESPONSE

- the **impulse response** $h[n]$ of a system is the **resulting signal** that exits the system, when the **input** was the **delta function**
- if two systems are **different**, they have different **impulse responses**
- **vice versa**, the **impulse response defines** the system
- the impulse response of filters is often called the **filter kernel**, **convolution kernel** or simply **kernel**



THE CONNECTION (1)

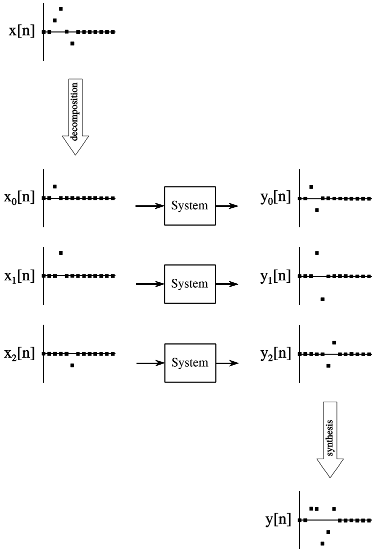
- let's consider a signal $a[n] = \begin{cases} 0 & : n \neq 8 \\ -3 & : n = 8 \end{cases}$
- expressed in terms of the **delta function** this reads as $a[n] = -3 \cdot \delta[n-8]$
- lets assume an **arbitrary system**, S
- we know the **impulse response** $h[n]$ of this system
- if the input to the system is $a[n]$, what is its output?

THE CONNECTION (1)

- let's consider a signal $a[n] = \begin{cases} 0 & : n \neq 8 \\ -3 & : n = 8 \end{cases}$
- expressed in terms of the **delta function** this reads as $a[n] = -3 \cdot \delta[n-8]$
- lets assume an **arbitrary system**, S
- we know the **impulse response** $h[n]$ of this system
- if the input to the system is $a[n]$, what is its output?
- we know $\delta[n]$ results in $h[n]$ for the system
- from the system's homogeneity and shift invariance, it follows that $-3 \cdot \delta[n-8]$ must result in $-3 \cdot h[n-8]$
- in words: the **output** is a **version of the impulse response** that has been **shifted** and **scaled** by the **same amount** as the **delta function** on the **input**

JKU

THE CONNECTION (2)



- the **output** for a scaled and shifted **impulse** is the scaled and shifted **impulse response** of the system
- a signal of N samples can be trivially decomposed into N scaled and shifted impulses
- the **output** of the system for this **signal** is the addition of N scaled and shifted **impulse responses**

STEP RESPONSE

- the **unit step function** is defined as

$$u[n] = \begin{cases} 0 & : n < 0 \\ 1 & : n \geq 0 \end{cases}$$

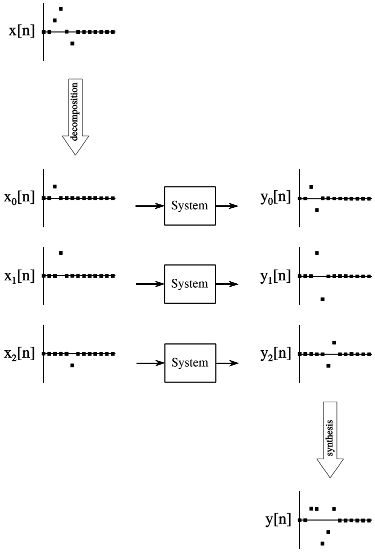
- the **step response** describes how the system output changes over time in response to a **unit step function** as its input
- knowing the **step response** of a system gives us information about its **stability** and the **time** it takes to reach a **stationary** state

FREQUENCY RESPONSE

- the **frequency response** can be found by taking the discrete fourier transform of the impulse response (more about that later)
- the **frequency response** of a linear system consists of its
  - **magnitude response** which indicates the **ratio** of a sine wave's **output amplitude** to its **input amplitude** depending on its frequency, when passing through the system
  - **phase response** which describes the **phase offset**, or **time delay** experienced by a sine wave passing through the system, depending on its frequency

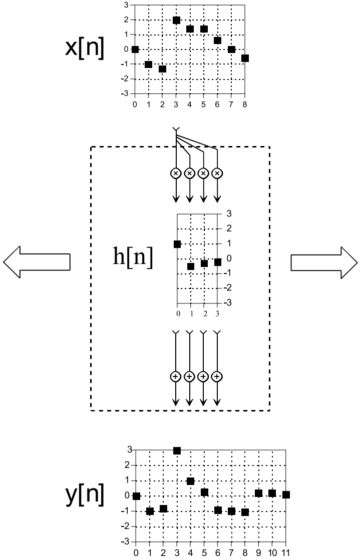
CONVOLUTION

CONVOLUTION IDEA



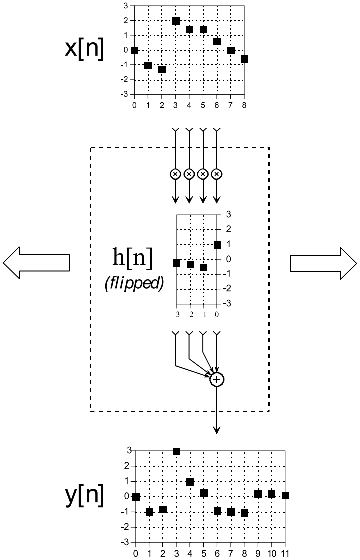
- the **output** of a system for a **signal** of N samples is the addition of N scaled and shifted versions of its **impulse response**
- the operation that takes a **signal** x and an **impulse response** h and produces the sum of correctly scaled and shifted versions of h is called (you guessed it): **convolution**

CONVOLUTION ILLUSTRATED (1)



- the “convolution machine” (dotted box) is free to **move one step at a time** to the left or right
- it **reads** one sample from the **input**
- it **multiplies** this sample with each sample of the impulse response
- it **adds** this scaled and shifted impulse response to the (initially zero) output signal

CONVOLUTION ILLUSTRATED (2)



- this is actually equivalent to a machine that reads M **samples** at once,
- **multiplies** each sample with the corresponding value of the **reversed** impulse response,
- **adds** the products and writes them to the output signal.
- Focus on one output sample on the previous slide and note down which four $x[k] \cdot h[j]$ are added to it; then compare to here.

CONVOLUTION FORMULATED

x[n] ∗ h[n] = y[n]

- is the single most important technique in DSP
- provides the mathematical framework for DSP

$$y[n] = \sum_{m=0}^{\infty} x[m] \cdot h[n-m]$$

sometimes also written as

$$y[n] = \sum_{m=0}^{\infty} x[n-m] \cdot h[m]$$

$x[n]$	$\ldots$	input signal
$*$	$\ldots$	convolution operator
$h[n]$	$\ldots$	impulse response
$y[n]$	$\ldots$	output signal

CONVOLUTION DEMONSTRATED

Jupyter Notebook "Convolution Demo" (also available in KUSSS)

WHAT TO REMEMBER

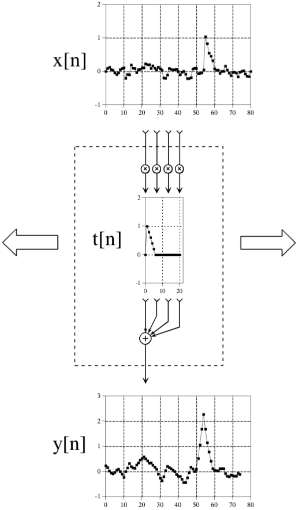
the **response** of any **linear, shift-invariant** system is the **convolution** of the **input** signal with the **impulse response**

CORRELATION

CORRELATION

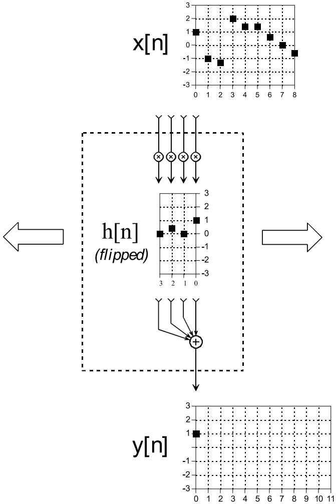
- **correlation** is a way to detect a **known** waveform against a **noisy** background
- correlation **combines** two signals into a third - the **cross correlation**
- if a signal is correlated with **itself** the result is called **auto correlation**
- correlation and convolution are **mathematically very similar**, yet in DSP they are viewed as **different** techniques

CORRELATION ILLUSTRATED

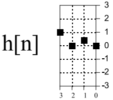


- the “correlation machine” (dotted box) is free to **move one step at a time** to the left or right
- its principle of operation is **identical** to that of the convolution machine
- but its impulse response is **not reversed**

WHY REVERSE $h[n]$ AT ALL?

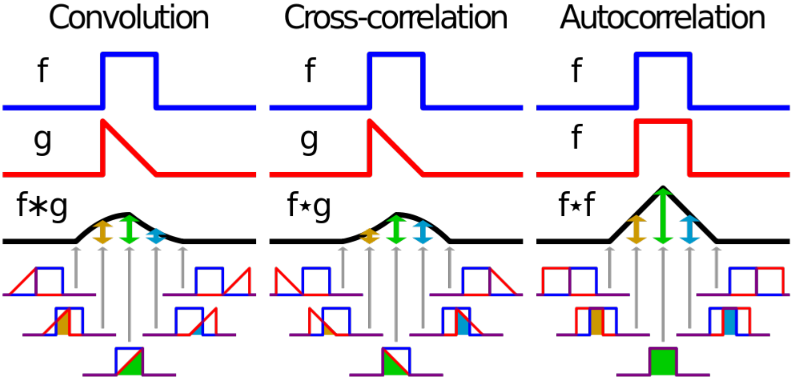


- assume the impulse response of our system looks as follows:



- that is, the system reproduces the input, followed by a softer copy
- in the convolution machine, if we would *not* reverse $h[n]$, then the *softer copy* would be produced *first*!
- in correlation, our goal is different: we want to find a template, so we use it as is

COMPARISON



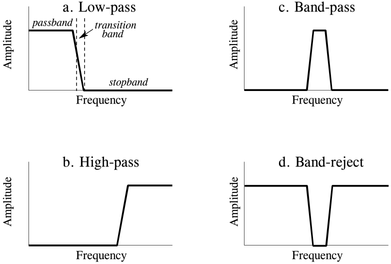
[12]

DIGITAL FILTERS

DIGITAL FILTERS

- **digital filters** are fundamental for digital audio processing
- there is time for a **cursory overview** over **filter properties**
- filters can be used in the **time** or **frequency** domain
- filters are used to
  - **separate** mixed signals
  - **restore** clean from distorted signals

FILTER TYPES



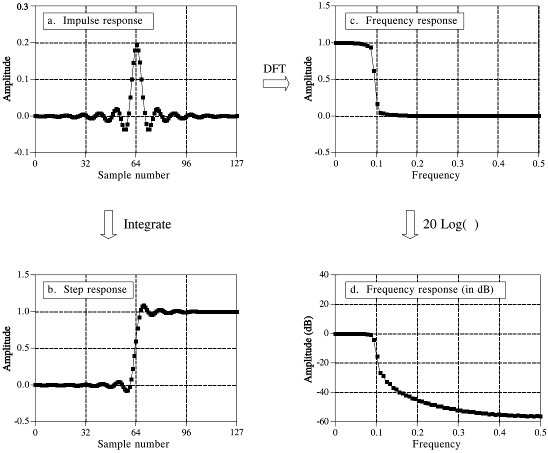
- ▷ dry
- ▷ low pass
- ▷ high pass
- ▷ band pass
- ▷ band reject

- **low pass** filters allow **only** sinusoids **below** a certain cutoff frequency to pass
- **high pass** filters allow **only** sinusoids **above** a certain cutoff frequency to pass
- **band pass** filters allow **only** sinusoids **between** two cutoff frequencies to pass
- **band reject** filters do **not** allow frequencies **between** two cutoff frequencies to pass

FILTER DEFINITION

- we can describe filters by their **impulse response**
- sometimes it is best to use their **frequency response**
- sometimes we are interested in the **step response** of a filter
- **each** of these different **responses**, contains **complete** information about **filter behaviour**

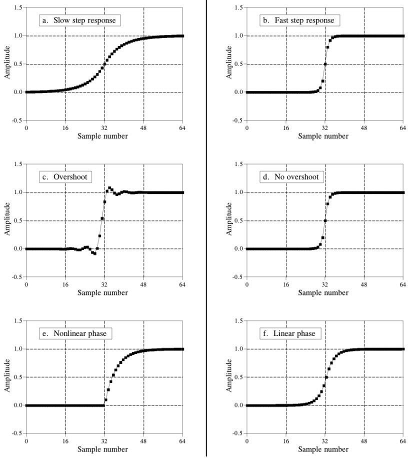
IMPULSE, STEP AND FREQUENCY RESPONSE



FILTER QUALITIES

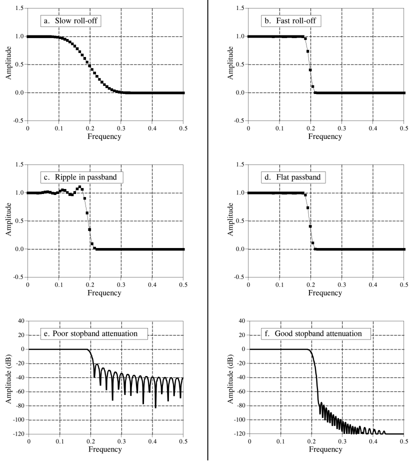
time domain

POOR | GOOD



frequency domain

POOR | GOOD



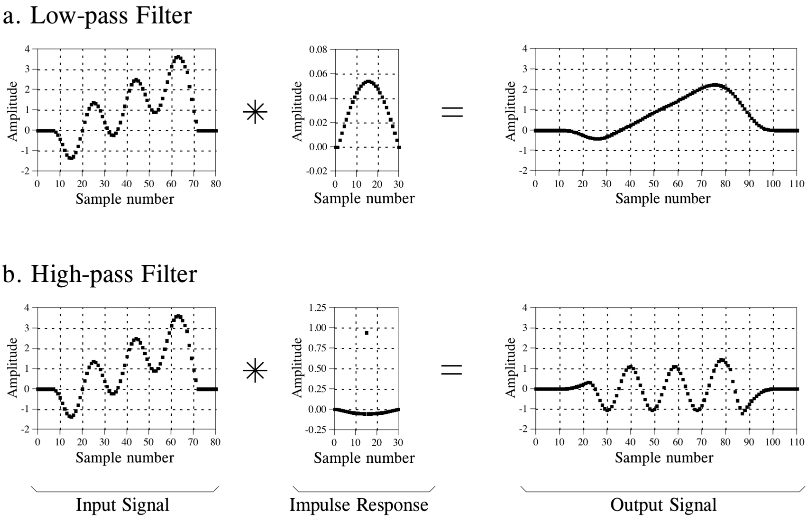
IMPLEMENTATION OF FILTERS

- **finite** impulse response (**FIR**) filters
  - implemented via **convolution**
  - impulse response is also called **filter kernel**
- **infinite** impulse response (**IIR**) filters
  - **besides** the **current** input point, **previously** calculated **output values** are used
  - IIR filters are defined by a set of **recursion coefficients**, instead of a filter kernel
  - For example, a low pass can be implemented as an exponentially-weighted moving average:

$$y[n] = \alpha x[n] + (1 - \alpha) y[n-1] \qquad \text{with } 0 < \alpha < 1$$

  - We will hardly see any IIR filters in this course

FIR FILTERS WITH CONVOLUTION



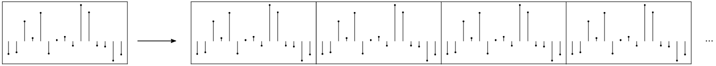
FOURIER ANALYSIS

FOURIER ANALYSIS

- fourier analysis **decomposes periodic** signals into **sinusoids**
- there are four variants:
  - **continuous fourier transform**: continuous time, continuous frequencies
  - **fourier series**: continuous time, discrete frequencies
  - **discrete time fourier transform**: discrete time, continuous frequencies
  - **discrete fourier transform**: discrete time, discrete frequencies
- we will focus on the **discrete fourier transform**

DISCRETE FOURIER TRANSFORM

- the **DFT** works on **discrete, periodic signals**
- it **transforms** signals **from** the **time** domain **into** the **frequency** domain
- if our signals are **not periodic**, we **simply pretend** they **are**
- if we wanted to compute the DFT of a signal with length 15, we would pretend it is periodic, meaning we pretend it **repeats itself** infinitely often in **both** directions



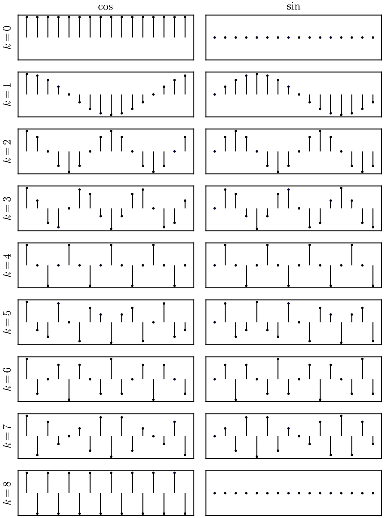
COMPLEX FORMULATION

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-2\pi kin}{N}}$$

- usually you will see the fourier transform defined in the complex domain $\mathbb{C}$ ($i^2 = -1$), using polar coordinates
- using Euler's formula $e^{ix} = \cos x + i \sin x$ we can split the above into

$$X[k] = \sum_{n=0}^{N-1} \underbrace{x[n] \cos(\frac{-2\pi kn}{N})}_{\text{real part}} + \underbrace{ix[n] \sin(\frac{-2\pi kn}{N})}_{\text{imaginary part}}$$

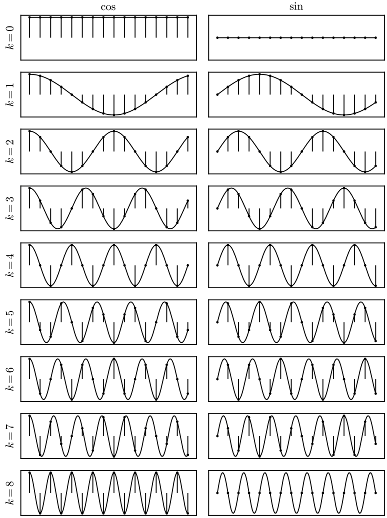
DISCRETE FOURIER BASIS (1)



- the waveforms on the left are the **basis** functions into which the signal is **decomposed**
- the basis functions are a set of **cosine** and **sine** waves with **unity** amplitude
- they are **generated** by the equations

$$
\begin{aligned}
c_k[n] &= \cos\left(\frac{-2\pi kn}{N}\right) \\
s_k[n] &= \sin\left(\frac{-2\pi kn}{N}\right)
\end{aligned}
$$

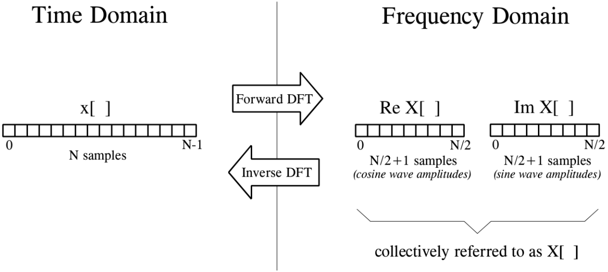
DISCRETE FOURIER BASIS (2)



- the nature of the basis functions is easier to see, if we **superimpose** the idealized waveforms
- the basis on the left is the discrete fourier basis for $N = 16$
- if the **input** to the DFT is a signal in the **time domain**, the **output** are the **values of the coefficients** for the **basis** functions in the **frequency domain**

N-POINT REAL DFT

- **time domain**: the **real** input $x[n]$ runs from 0 to $N-1$
- **frequency domain**: the **complex** output $X[n]$ runs from 0 to $\frac{N}{2}$
- alternatively we could say we have $\frac{N}{2}+1$
  - **real** output components $\operatorname{Re} X[n]$
  - **imaginary** output components $\operatorname{Im} X[n]$



DFT BY MATRIX PRODUCT

- how do we **express** the input signal $x[n]$ in the **new basis**?
- let's do a **dot product** of each basis function with the signal
- if we put the **basis functions** into a **matrix**, we can simply **multiply** this **matrix** with our **signal** to obtain the **dot products**
- this is **educational**, but unfortunately ...
- ... **horribly** slow - matrix multiplication costs $O(N^2)$

$$
\begin{pmatrix}
c_0[0] & \cdots & c_0[N-1] \\
\vdots & \ddots & \vdots \\
c_{N/2}[0] & \cdots & c_{N/2}[N-1] \\
s_0[0] & \cdots & s_0[N-1] \\
\vdots & \ddots & \vdots \\
s_{N/2}[0] & \cdots & s_{N/2}[N-1]
\end{pmatrix}
\cdot
\begin{pmatrix}
x[0] \\
\vdots \\
x[N-1]
\end{pmatrix}
=
\begin{pmatrix}
\operatorname{Re} X[0] \\
\vdots \\
\operatorname{Re} X[N-1] \\
\operatorname{Im} X[0] \\
\vdots \\
\operatorname{Im} X[N-1]
\end{pmatrix}
$$

$$F_{real} \cdot x = X$$

JKU

DFT BY FFT

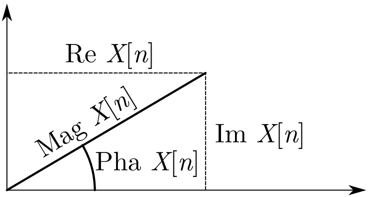
- the **F**ast **F**ourier **T**ransform is a way to make the **multiplication by the complex Fourier matrix fast** [10]
- it has a runtime of only $O(N \log N)$
- it is usually **derived** via the **complex DFT**, and it relies on the fact that the **complex N-point Fourier matrix** can be **factorized** into **two smaller Fourier matrices** of **half** the size
- we will simply use the FFT algorithm as a **gray box** where we know **exactly what** function it is computing, but do **not** care **how** it is implemented
- the **most common** FFT flavor works with **powers of two**

$$F_{2N} = \begin{pmatrix} I & D \\ I & -D \end{pmatrix} \cdot \begin{pmatrix} F_N & 0 \\ 0 & F_N \end{pmatrix} \cdot \begin{pmatrix} P_{even} \\ P_{odd} \end{pmatrix}$$

FFT TIDBITS

- an FFT algorithm was introduced by Cooley and Tukey in [2]
- apparently Gauss knew about it as well [11]
- there are many **fast C libaries**, such as FFTW, KISSFFT,
- your favorite “scientific computing software stack” (Numpy/Scipy, Matlab, Mathematica, Julia, ...) has it in its libraries as well (in fact, if it has not, you should switch ...)
- FFT lengths other than powers of two are possible, but ideally, N should still have many small prime factors

MAGNITUDE AND PHASE



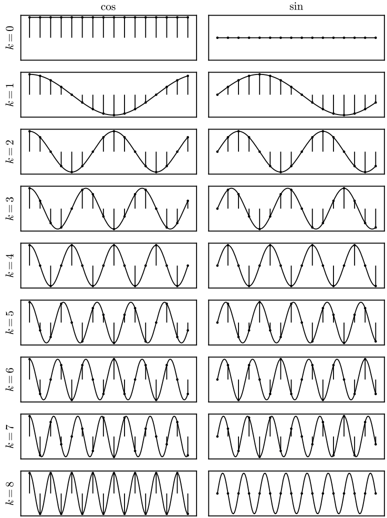
- what do we do with $\mathrm{Re}\,X[n]$ and $\mathrm{Im}\,X[n]$?
- we **convert** it into **polar form** !
- **magnitude** $\mathrm{Mag}\,X[n] = \sqrt{(\mathrm{Re}\,X[n])^2 + (\mathrm{Im}\,X[n])^2}$
- **phase** $\mathrm{Pha}\,X[n] = \arctan\left(\frac{\mathrm{Re}\,X[n]}{\mathrm{Im}\,X[n]}\right)$
- this information will help us **tremendously** to analyze music
- in a **magnitude** spectrum, the **magnitude** of a bin $X[n]$ is the **sum** of all **energy** in its frequency band

FREQUENCY RESOLUTION (1)

- the **length** of the basis functions limits the frequency **resolution**
- for an N-point DFT, a signal $x[n]$ with sampling rate S [Hz]
  - ☐ results in a **complex spectrum** $X[n]$
  - ☐ the **spectrum** has $\frac{N}{2} + 1$ **bins**
  - ☐ the **bins** are **equally spaced** in the range $[0, \frac{S}{2}]$ [Hz]

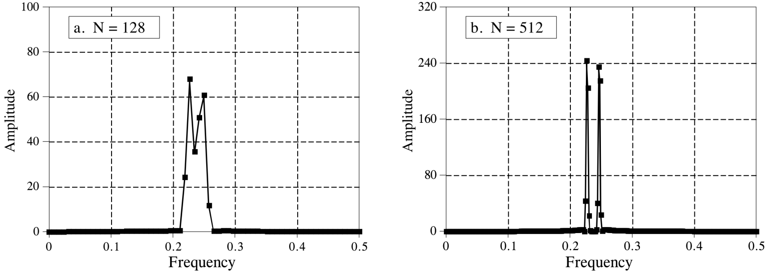
    $\rightarrow$ i.e., from 0 to the **nyquist frequency** $\frac{S}{2}$
  - ☐ the **resolution** (in [Hz]) is $r \approx \frac{S}{N}, \left(r = \frac{S/2}{N/2+1}\right)$
  - ☐ a **bin** $X[b]$ starts at $r \cdot b$, and stops at $r \cdot (b+1)$ [Hz]
  - ☐ the **center frequency** of a bin is therefore $r \cdot (2b+1)$
  - ☐ the 0-th bin holds the **DC** component of the signal

FREQUENCY RESOLUTION (2)



- the basis on the left is the discrete fourier basis for $N = 16$
- the first bin is the **DC** component, the last bin is the **nyquist frequency**
- assuming a sample rate of $S = 44100\,\text{Hz}$, each bin covers $\frac{22050\,\text{Hz}}{9} = 2450\,\text{Hz}$
- to increase the frequency resolution, we need to increase N

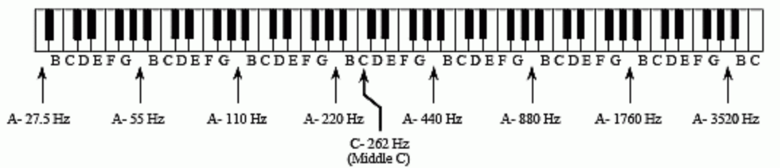
FREQUENCY RESOLUTION (3)



- if we have too few bins, we'll lump them together
- this might have detrimental effects on various things we are trying to do, such as pitch- or chord detection

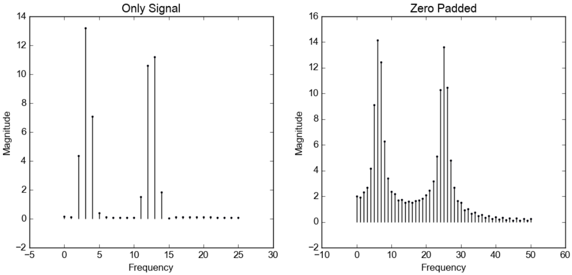
FREQUENCY RESOLUTION (3)

- DFT has **equally spaced** basis functions
- **musical** notes are on a **logarithmic scale**
- therefore we have **much fewer** bins per note for **lower** pitches than for **higher** pitches
- typical values: sample rate 44.1 kHz, 2048-point DFT (46 ms) means a frequency resolution of ≈ 21.5 Hz



ZERO PADDING

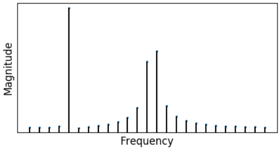
- we can **pad** the **input** signal to the DFT with a number of **zeros**
- this **does not affect** the **shape** of the **spectrum**
- however, it **reduces** the **inter-sample** spacing
- it **does not increase** the **frequency resolution**
- however **it does increase** the **resolution IN the frequency domain** (by **interpolation**)



JKU

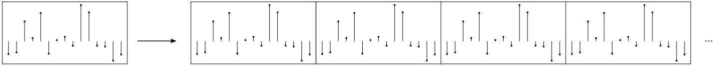
SPECTRAL LEAKAGE

- assume a signal $x[n]$ consisting of **two sine** waves
- one sine has a frequency **exactly equal** to a basis function
- one sine has a frequency **between** two basis functions
- in the spectrum, the first one is an impulse, the latter is smeared

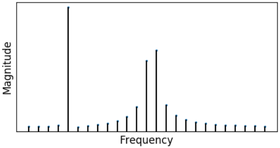


SPECTRAL LEAKAGE: WHY

- remember that DFT implicitly assumes a periodic signal

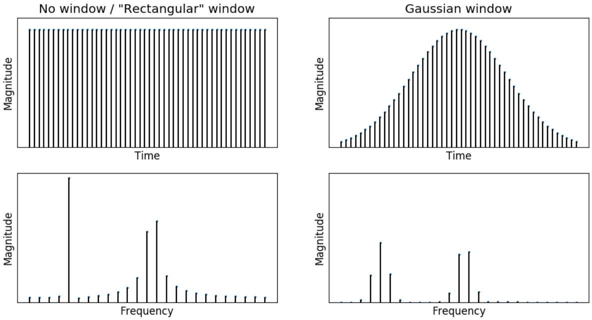


- a sine matching a basis function will seamlessly repeat
- a sine *not* matching a basis function will be cut at the border
- explaining this cut requires all the basis functions



WINDOWING

- we can multiply the signal by a window function $w[n]$
- this can be chosen to smoothen hard cuts at the border
- it results in comparable peak shapes for the two sines
- popular window functions: Hann, Hamming, Gaussian, …



CONVOLUTION THEOREM

The **convolution theorem** says: let $\mathcal{F}$ be the fourier transform operator, f, g be two arbitrary functions and let $\cdot, *$ denote pointwise multiplication and convolution respectively, then

- $\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$
- $\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$

in words:

- convolution in time corresponds to multiplication of the spectra
- multiplication in time corresponds to convolution of the spectra

REVISITING FREQUENCY RESPONSE

- convolution in time corresponds to multiplication of the spectra
- so instead of **convolving** a **signal** with a filter's **impulse response**, we can **multiply** its **spectrum** with the filter's **frequency response** (the spectrum of its impulse response)
- and: knowing the frequency response we want (e.g., a bandpass), we can compute the time domain FIR filter (by a fourier transform)

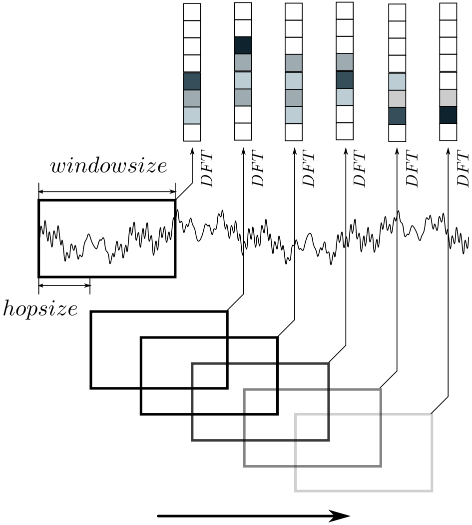
REVISITING SPECTRAL LEAKAGE

- multiplication in time corresponds to convolution of the spectra
- so **multiplying** $x[n]$ pointwise by a window function $w[n]$ is the same as **convolving** its fourier transform $\mathcal{F}\{w[n]\}$ with the fourier transform of the signal $\mathcal{F}\{x[n]\}$
- the fourier transform of a rectangular window is a sinc function
- the fourier transform of a Gaussian window is a Gaussian

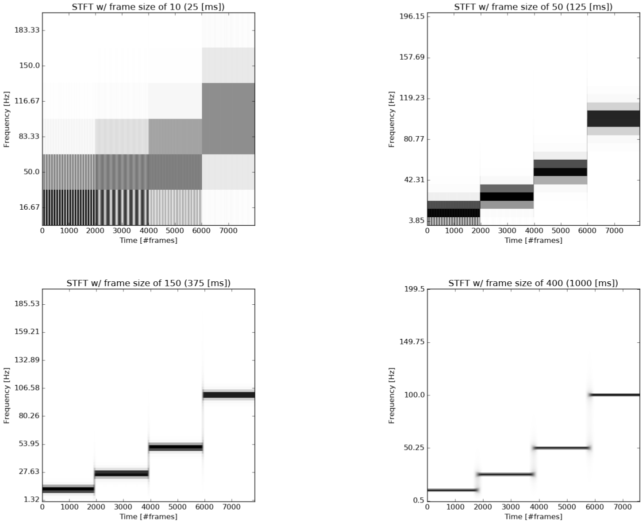
SHORT-TIME FOURIER TRANSFORM (STFT)

- the DFT of the **whole** (musical) signal yields the spectrum of the **whole** piece
- this is not too useful, because we **lose** all **timing information** if we look at the magnitudes
- to **preserve** some **timing** information, we **cut** up the original signal into **small frames**
- we then simply compute the DFT for **each frame**
- tradeoff:
  - make frames short enough so the signal within is *stationary* (does not change frequencies over time)
  - make frames long enough to have useful frequency resolution

STFT ILLUSTRATED



TRADEOFF


STFT w/ frame size of 10 (25 [ms])
STFT w/ frame size of 50 (125 [ms])
STFT w/ frame size of 150 (375 [ms])
STFT w/ frame size of 400 (1000 [ms])
Frequency [Hz]
Time [#frames]

HOMEWORK

- download SonicVisualiser [1] and some plugins
- download Audacity [5] and some plugins
- go **wild** !
- either try to analyze recordings of you playing your instrument, or synthesize weird sounds
- look at sound in the STFT representation, play with the window size and window function etc . . .

REFERENCES I

[1] C. Cannam, C. Landone, and M. Sandler.
Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files.
In *Proceedings of the ACM Multimedia 2010 International Conference*, pages 1467–1468, Firenze, Italy, October 2010.

[2] James W Cooley and John W Tukey.
An algorithm for the machine calculation of complex fourier series.
Mathematics of computation, 19(90):297–301, 1965.

[3] Fredric J Harris.
On the use of windows for harmonic analysis with the discrete fourier transform.
Proceedings of the IEEE, 66(1):51–83, 1978.

JKU

REFERENCES II

[4] Alexander Knaub.
Sampling of Analog Signals, 2016.

[5] Dominic Mazzoni, Roger Dannenberg, and LOTS more.
Audacity, 2016.

[6] Timm Ostermann.
VL Analog Circuit Design, 2016.

[7] Claude E Shannon.
Communication in the presence of noise.
Proceedings of the IRE, 37(1):10–21, 1949.

[8] Julius O Smith III.
Introduction to Digital Filters with Audio Applications, 2016.

REFERENCES III

[9] Ken Steiglitz.
A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music.
Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1996.

[10] Gilbert Strang.
MIT OCW, Complex Matrices and FFT, 2016.

[11] Charles Van Loan.
Computational frameworks for the fast Fourier transform, volume 10.
Siam, 1992.

[12] Wikipedia.
Convolution, 2016.