The AsTeRICS Academy

for cross-cultural education in Assistive Technology

AsTeRICS academy

MA23 Wirtschaft, Arbeit und Statistik

StaDt+Wien

FH University of Applied Sciences TECHNIKUM WIEN

Module 4c: Bioelectric Signals

Processing and Classification (Basics)

sTeRICS academy

MA23 Wirtschaft, Arbeit Statistik

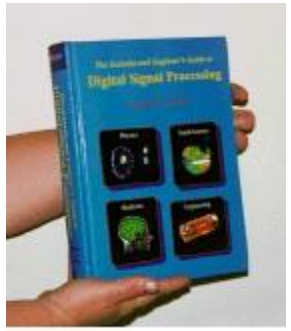
StaDt+Wien

FH University of Applied Sciences TECHNIKUM WIEN

Topics

Foundations of DSP
Basic Operations: convolution
Digital Filters: FIR and IIR
Some Classification methods
Biosignal Libraries and Applications

Practical Demonstrations: Matlab and FiView
Review of Project exercises
Firmware - programming



Recommended online book for todays topics:

The Scientist and Engineer's Guide to Digital Signal Processing
by Steven W. Smith, Ph.D.

http://www.dspguide.com/

Signal characteristics

In microprocessor based biomedical applications we have Signals that:

- **are discrete (digitized with a sampling rate)**
- **are to a certian degree spoiled by noise or artifacts**
- **have some properties we are especially interested in**
- **have stochastic and sometimes nearly-deterministic behaviour**

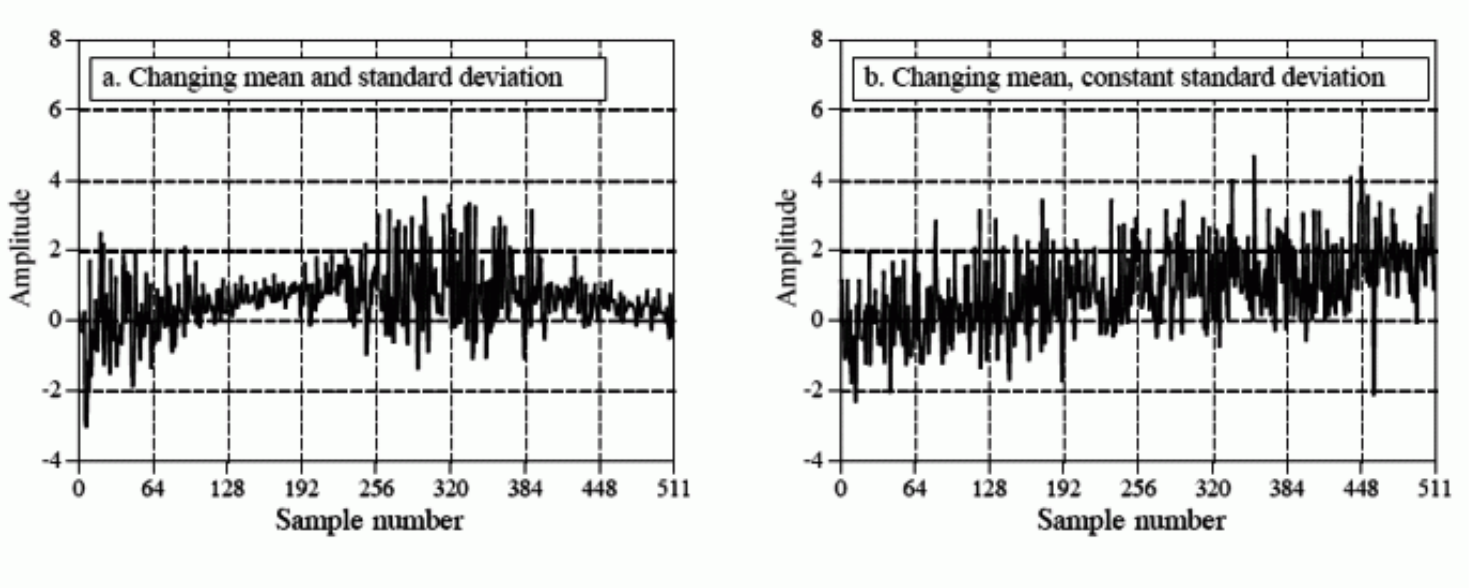
Basic Signal Statistics

Mean:

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$

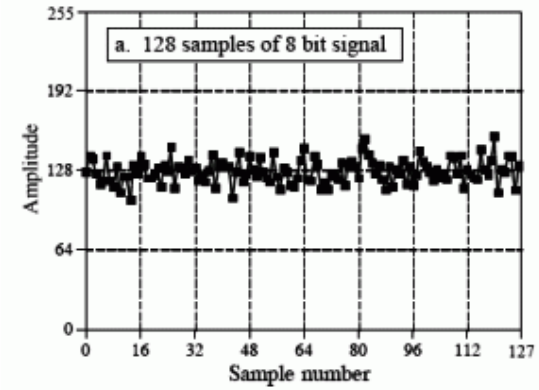
Standard deviation:

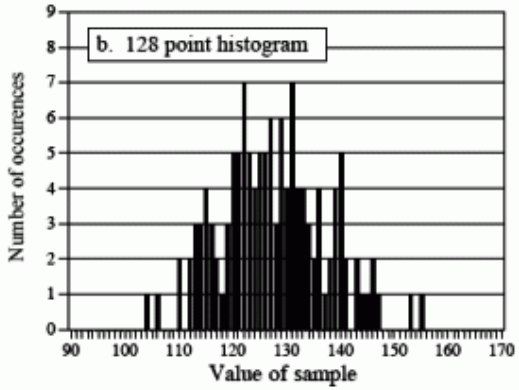
$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \mu^2)$$

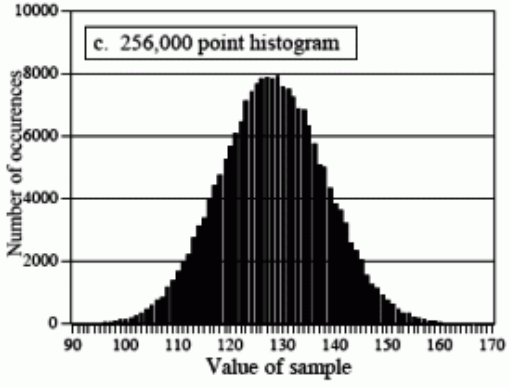


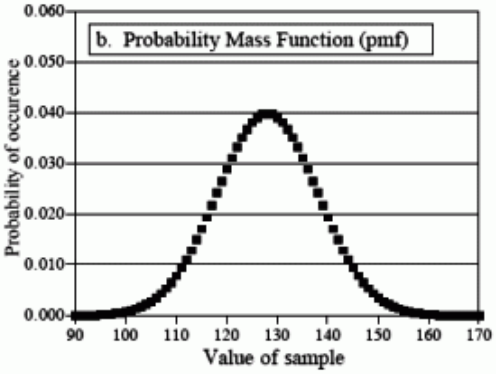
Basic Signal Statistics

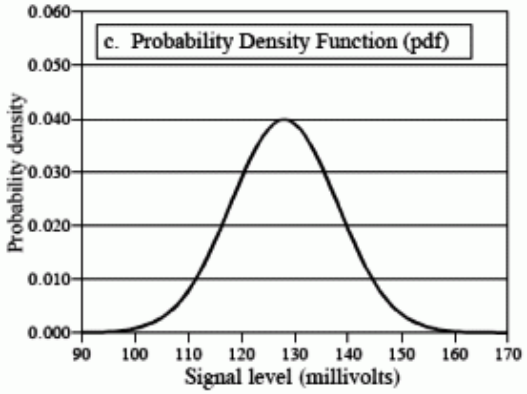
Histogram, Probability mass function, Probability density function:







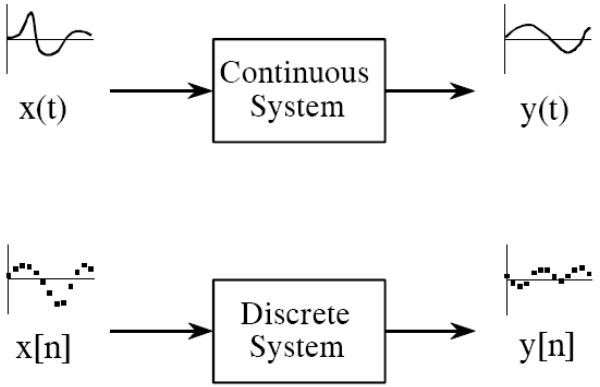




Foundations of DSP

Signals and LTI-Systems:

- Generation of an output signal in response to an input signal
- discrete and continuous systems



Linear, Timeinvariant Systems:

- additivity

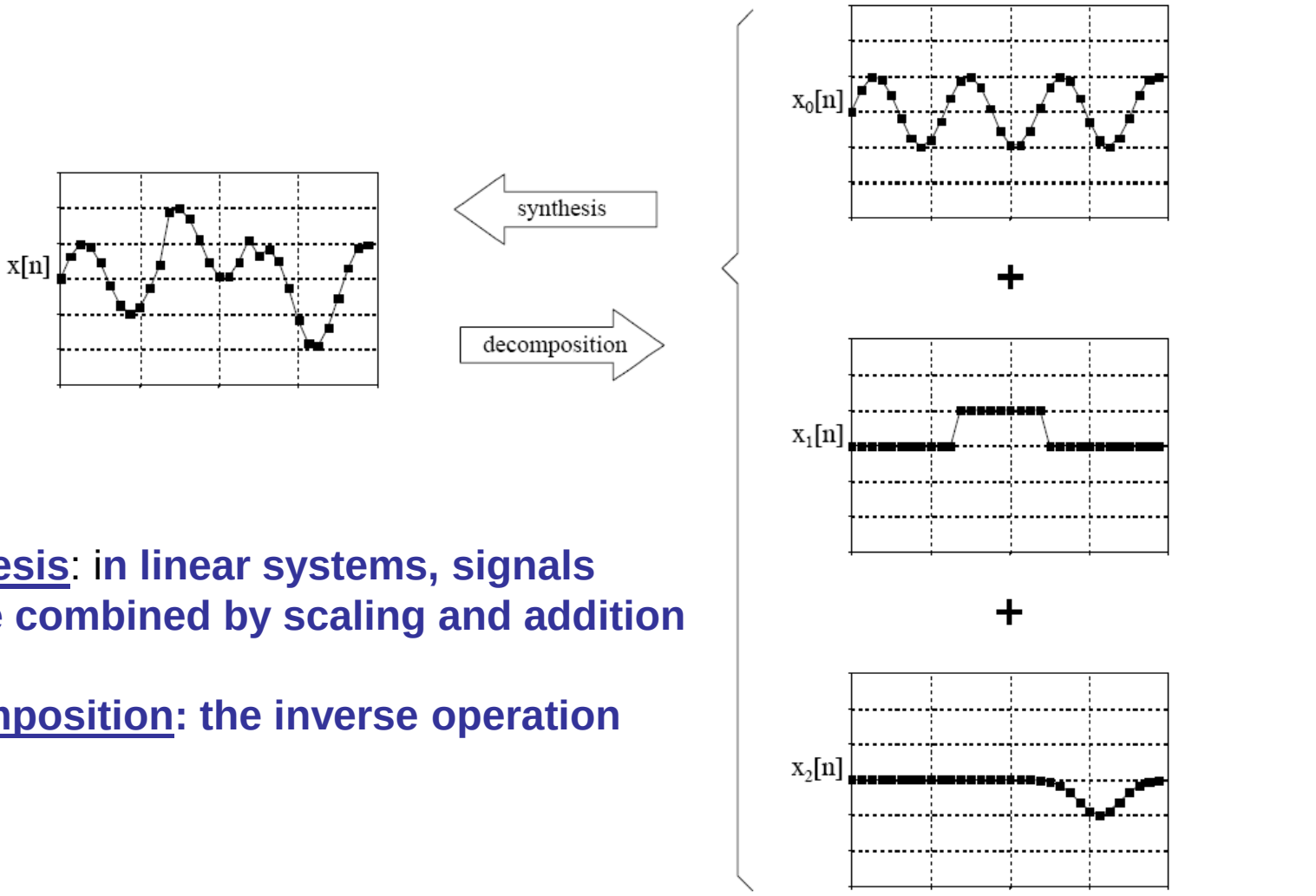
$$x[n] \xrightarrow{System} y[n] \Rightarrow kx[n] \xrightarrow{System} ky[n]$$

- homogeneity

$$x_1[n] \xrightarrow{System} y_1[n], x_2[n] \xrightarrow{System} y_2[n] \Rightarrow x_1[n] + x_2[n] \xrightarrow{System} y_1[n] + y_2[n]$$

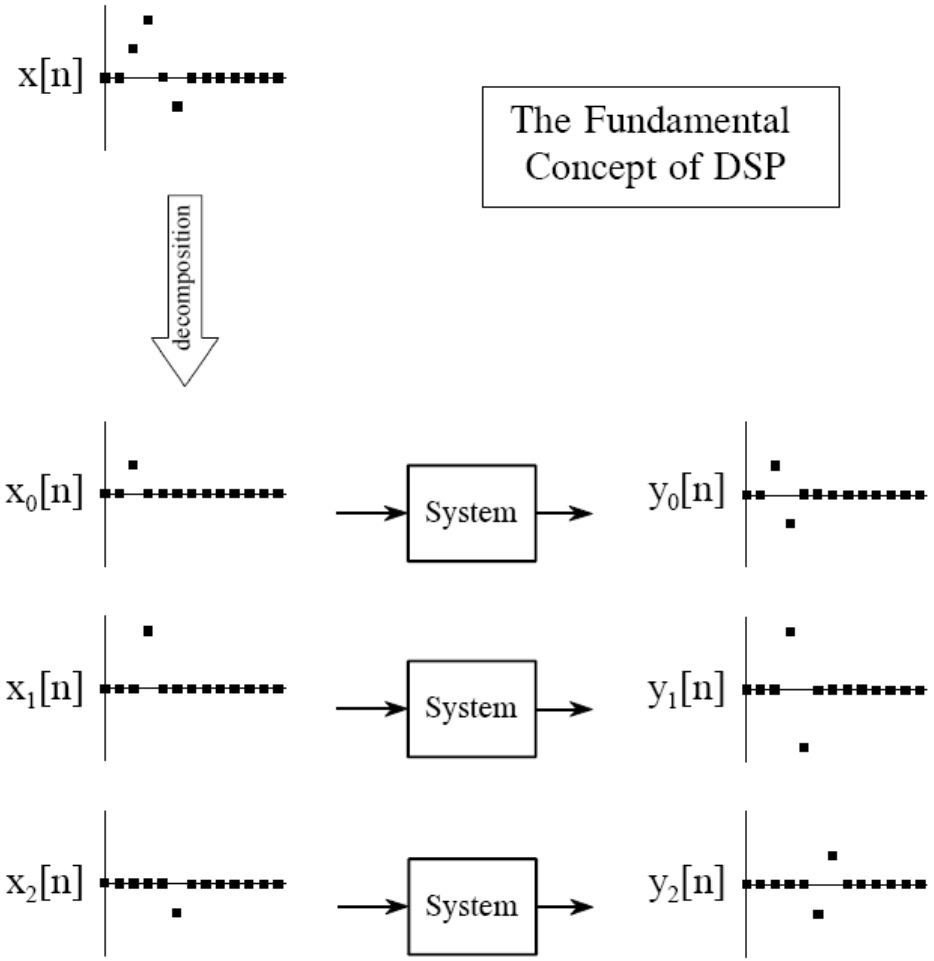
- shift invariance

Foundations of DSP



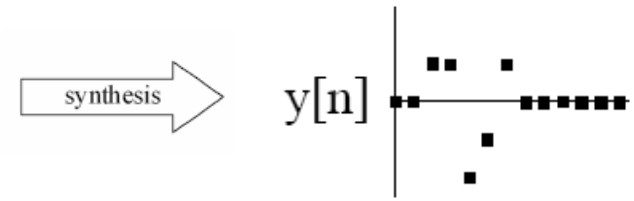
- **Synthesis**: in linear systems, signals can be combined by scaling and addition
- **Decomposition**: the inverse operation

Foundations of DSP - Superposition



- Any signal can be *decomposed* into a group of additive components xi

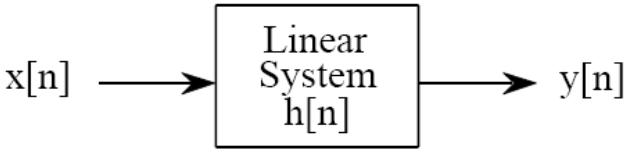
- Passing these components through a linear system produces signals, *yi*



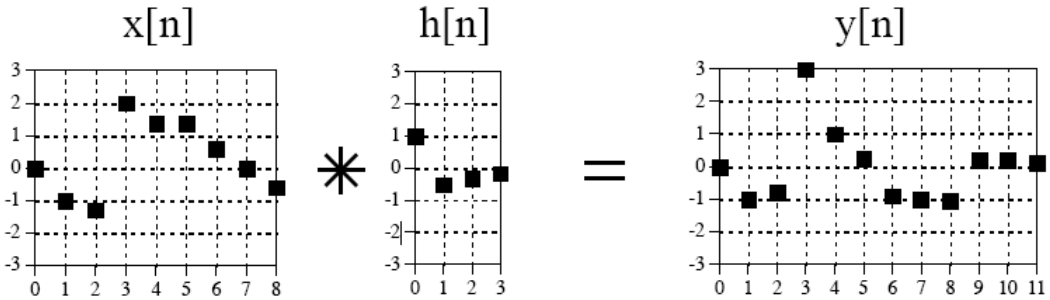
- The *synthesis* of these output signals produces the same signal as when *x* [*n*] is passed through the system

Foundations of DSP - Convolution

- **combined two signals into a third one**
- **applies a linear system to a signal via it‘s impulse response, wich fully describes the system behaviour**



$$x[n] * h[n] = y[n]$$



$$y[i] = x[i] * h[i] \Leftrightarrow y[i] = \sum_{j=0}^{M-1} h[j]x[i-j]$$

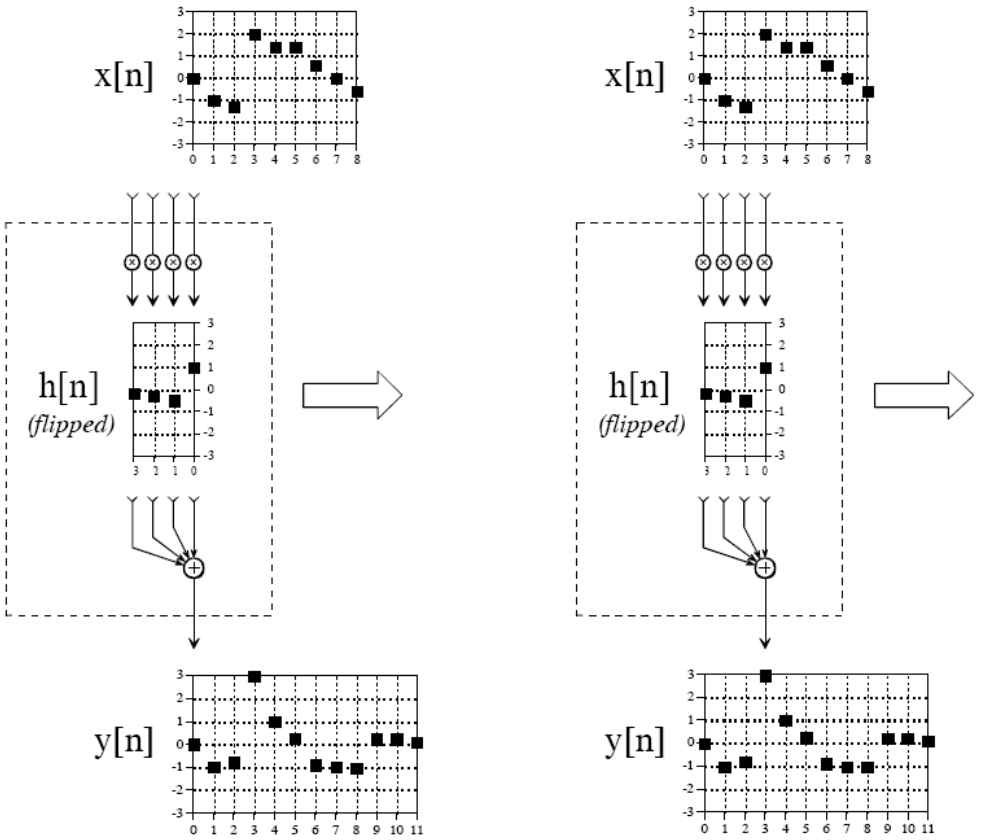
M .. length of impulse response

Foundations of DSP – Convolution

Application of a LTI:

1) multiplication of the input samples with the flipped impulse response

1) addition of the values gives result for the corresponding output sample

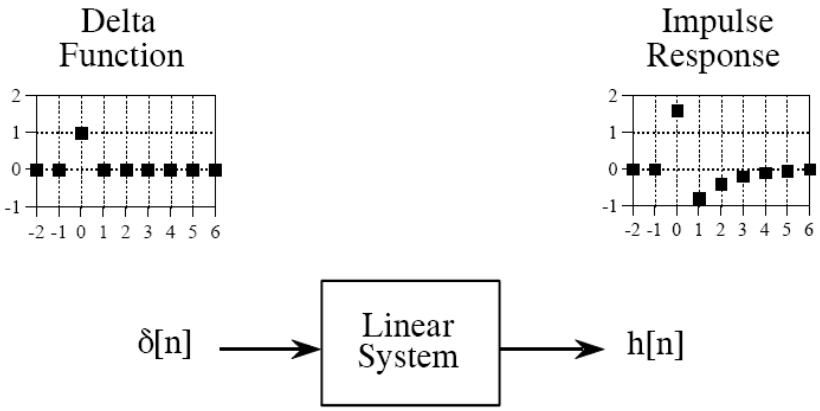


a. Set to calculate y[0]

b. Set to calculate y[3]

Foundations of DSP - Convolution

- many samples of the input signals contribute to one output sample
- the samples of the impulse response act as weighing coefficients
- feeding a delta function into a linear system gives the impulse response:



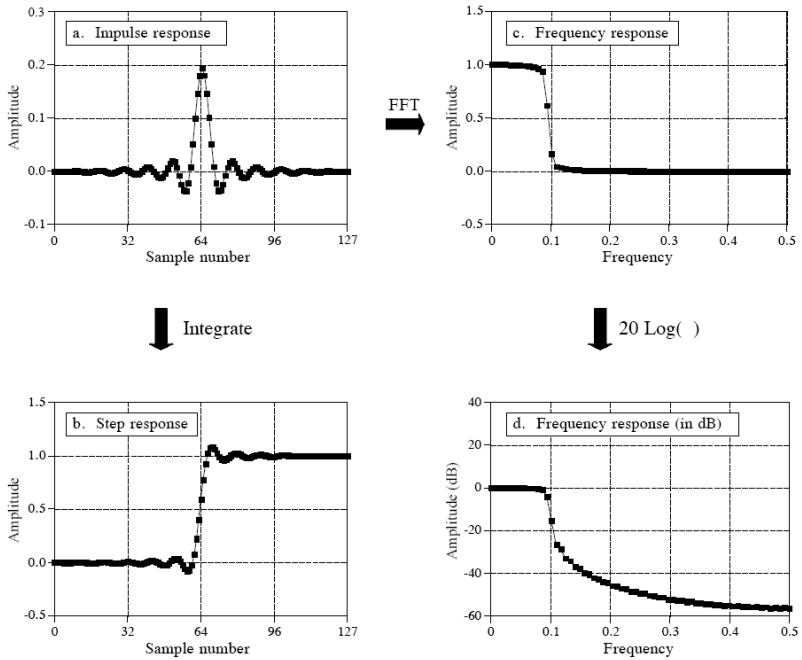
Foundations of DSP - Convolution

- **simple implemenation of convolution:**

```
// Input Signal: x[80], Impulse Response: h[30]
// Output Signal: y[110]

int i,j;
for (i = 0; i< 110; i++)
   for (y[i]=0,j=0; j<30; j++)
       if ((i-j >= 0) && (i-j<80))
          y[i] = y[i] + h[j] * x[i-j]
```

Impulse-, Step- and Frequency Response

Relationships between impulse-, step- and frequency response:



Note: Convolution in time domain = multiplication in frequency domain !

FIR Filters

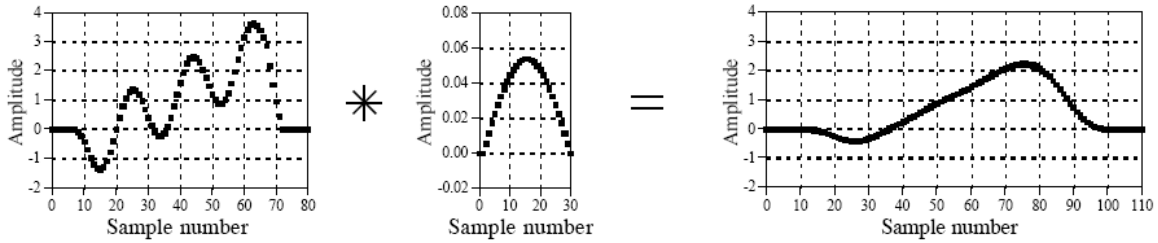
The shape of the impulse response determines phase- and frequency response of an LTI system. The impulse response is also called „filter kernel“.

- **Finite Impulse Response Filters can be implemented by a single convolution of an input signal with the filter kernel**
- **Several positive vaules in the impulse response give an averaging (low-pass) filter**
- **Substracting a low-pass filter kernel from the delta function gives a high pass filter kernel**
- **A symmetrical impulse response gives a linear phase response**

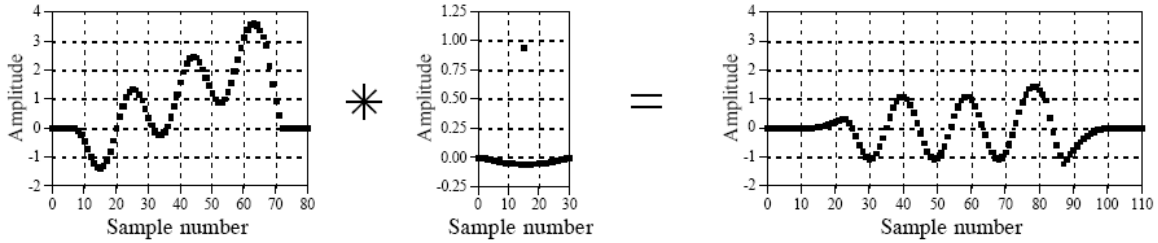
FIR Filters

Example High and Lowpass Filter-Kernels:

a. Low-pass Filter



b. High-pass Filter



Input Signal | Impulse Response | Output Signal

Z - Transform

- **Digital Filters can be described by the generalized discrete differential equation:**

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

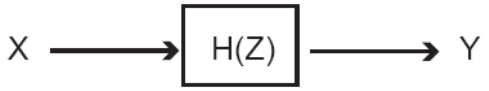
a, b : filter coefficients
x[n] : input signal
y[n] : output signal
M,N : filter order

the right side depends only on the inputs x[n] : feed-forward
the left side depends on the previous outputs y[n] : feed-back

***FIR Filters* have only feed-forward components,**
they can be calculated non-recursively, by convolution

***IIR Filters* have feed-back components also,**
they are calculated recursivelsy (infinite impulse response)

Z - Transform



- discrete version of the Laplace-transform
- using the Z-transform, the characteristics of a digital filter can be described by the transfer function:

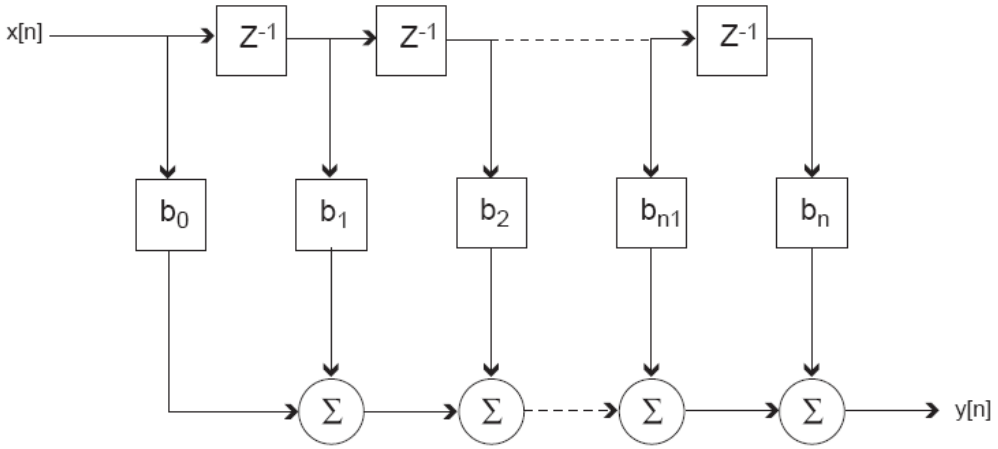
$$\sum_{m=0}^{M} a_m \cdot y[n-m] = \sum_{k=0}^{N} b_k \cdot x[n-k] \quad \left| \xleftrightarrow{Z} \right. \quad Y(z) \cdot \sum_{m=0}^{M} a_m \cdot z^{-m} = X(z) \cdot \sum_{k=0}^{N} b_k \cdot z^{-k}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k \cdot z^{-k}}{\sum_{m=0}^{M} a_m \cdot z^{-m}}$$

- z^x in Z-domain represents a delay element of x discrete delays,
- the numerator describes the feedfoward part of the filter, 0 = „zeros“
- the denumerator describes the feedback part of the filter, 0 = „poles“

Digital Filters – FIR filters

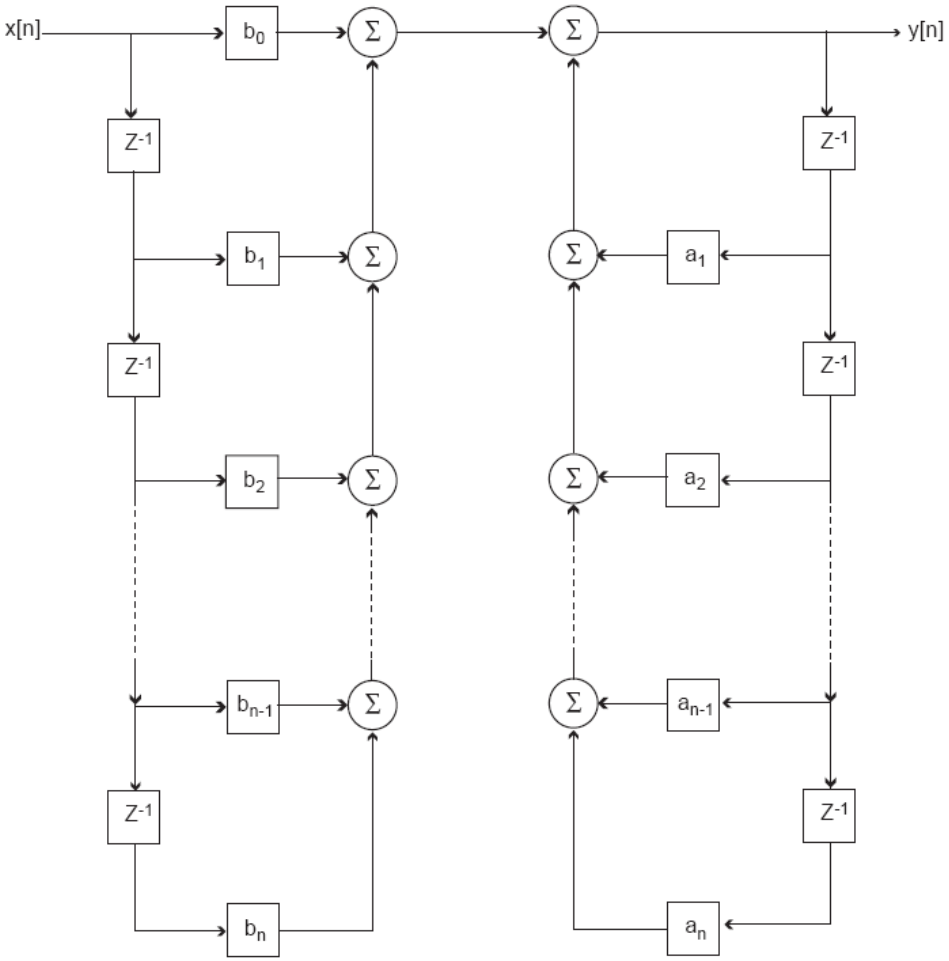
$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k]$$



- finite impulse response, no recursion
- described by multiplication coefficients
- less sufficient (need higher order)

Digital Filters – IIR filters

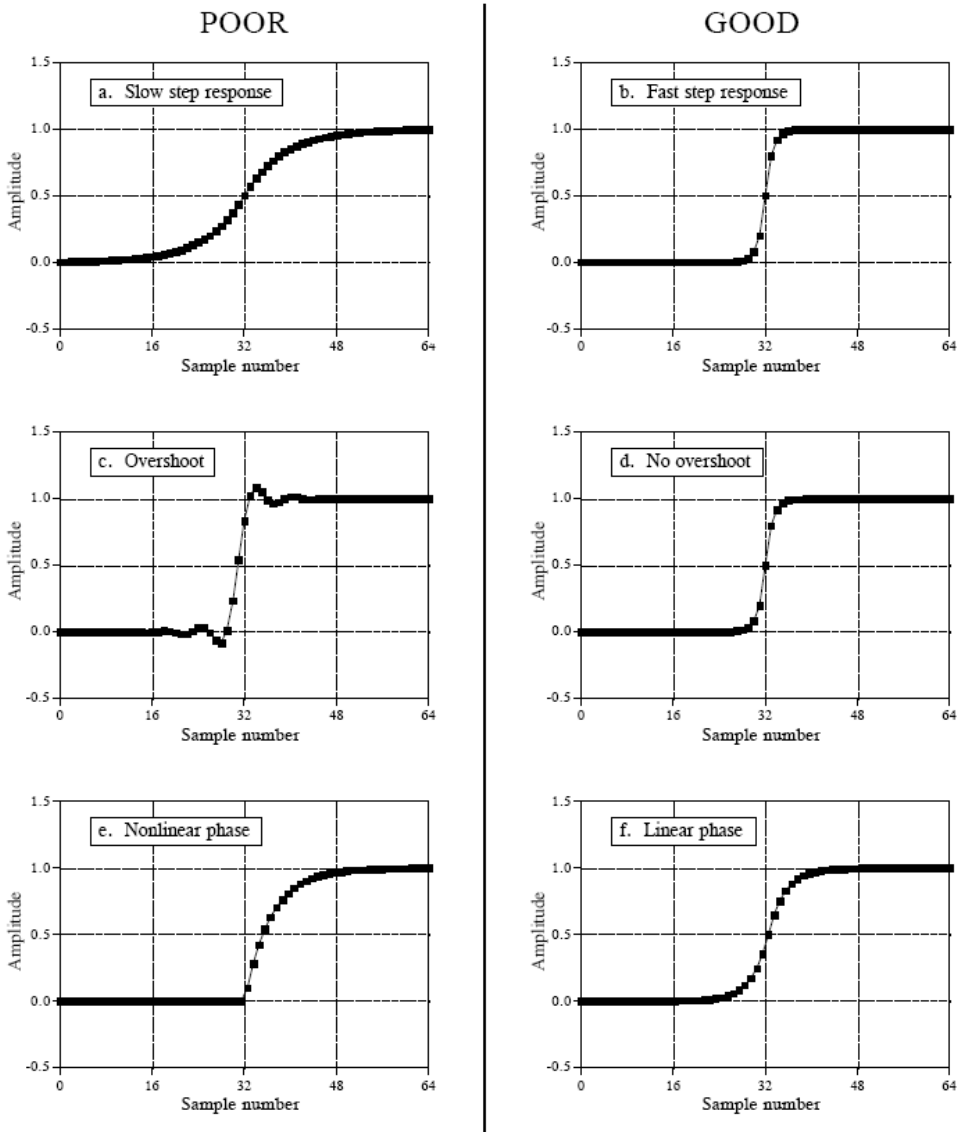
$$y[n] = \sum_{k=0}^{N} b_k \cdot x[n-k] + \sum_{m=1}^{M} -a_m \cdot y[n-m]$$



- infinite impulse response, truncated at a certain precision
- use previously calculated values from the output (recursion)
- described by recursion coefficients
- more efficient, but can be unstable

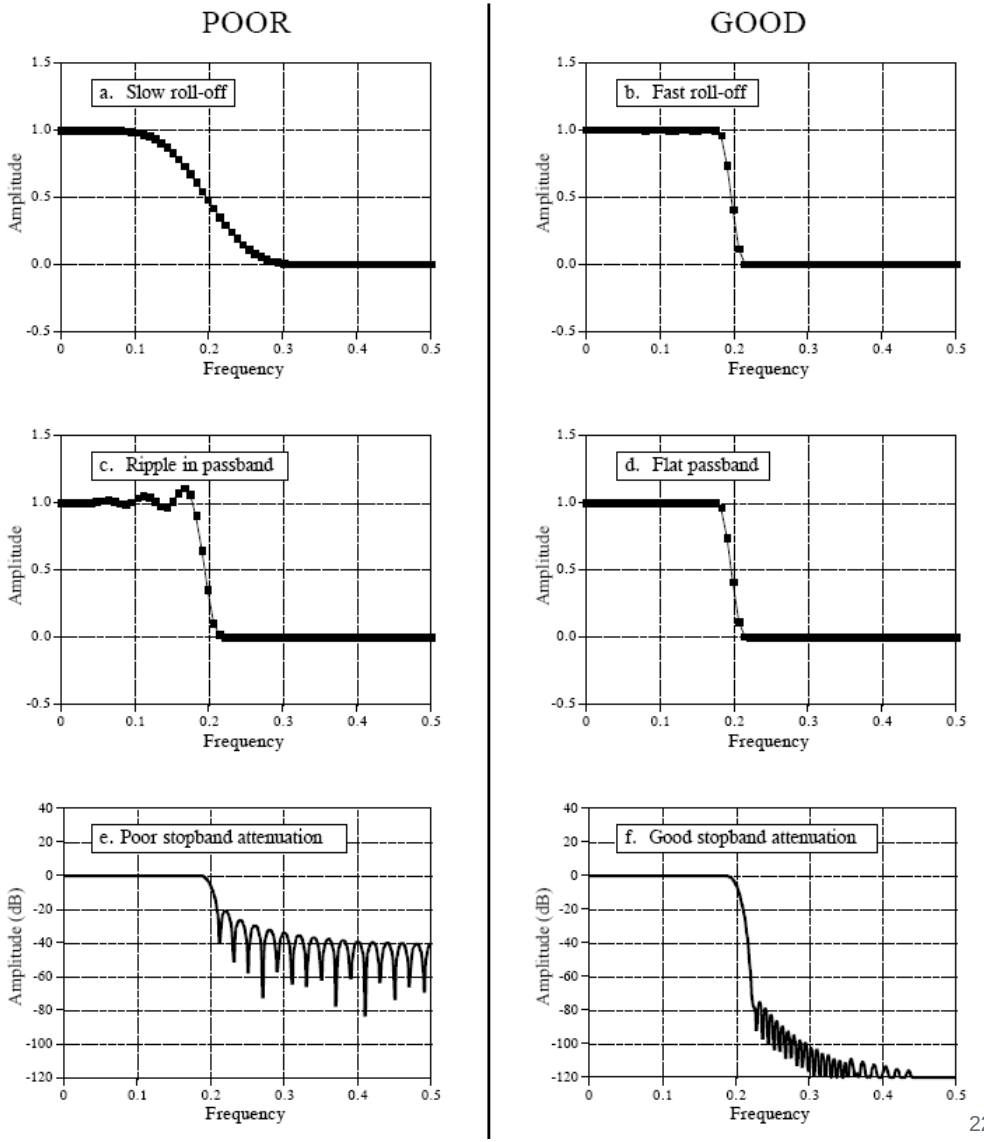
Filter characteristics

Performance in Time Domain

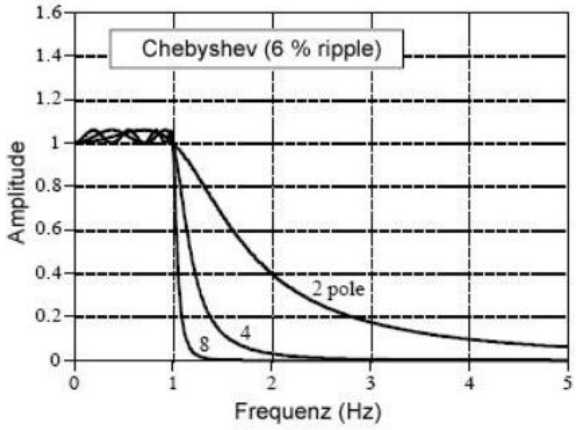


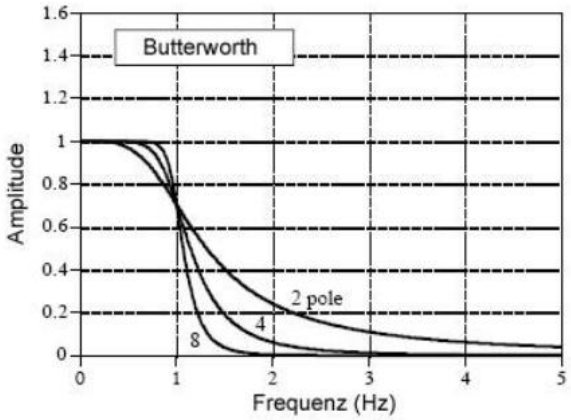
Filter characteristics

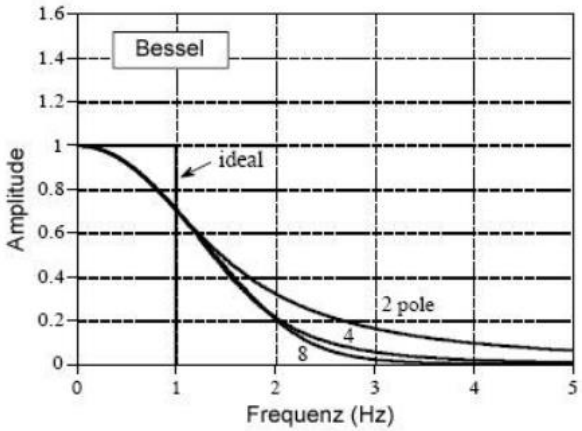
Performance in Frequency Domain

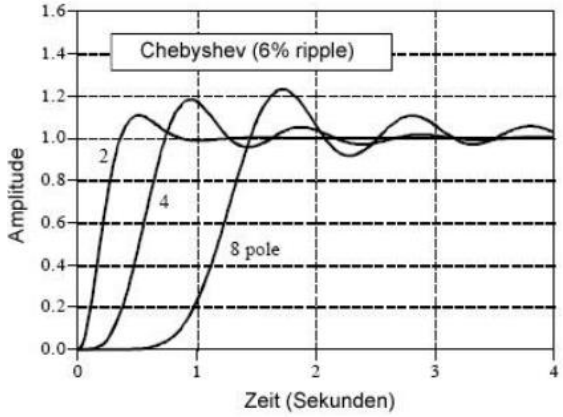


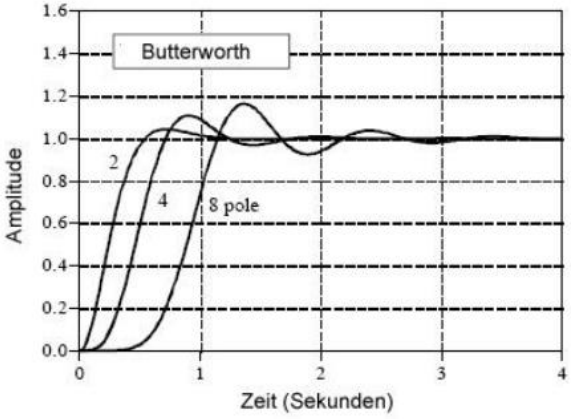
typical IIR filters

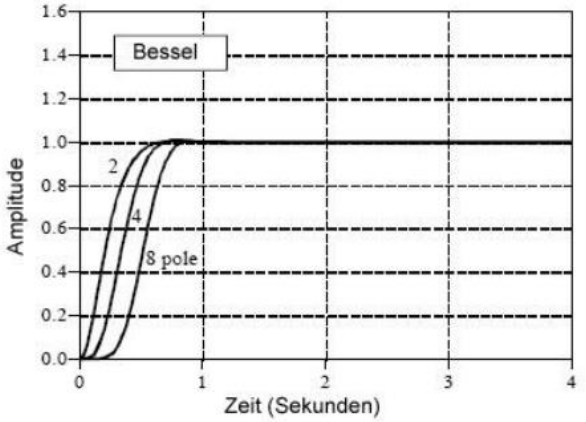








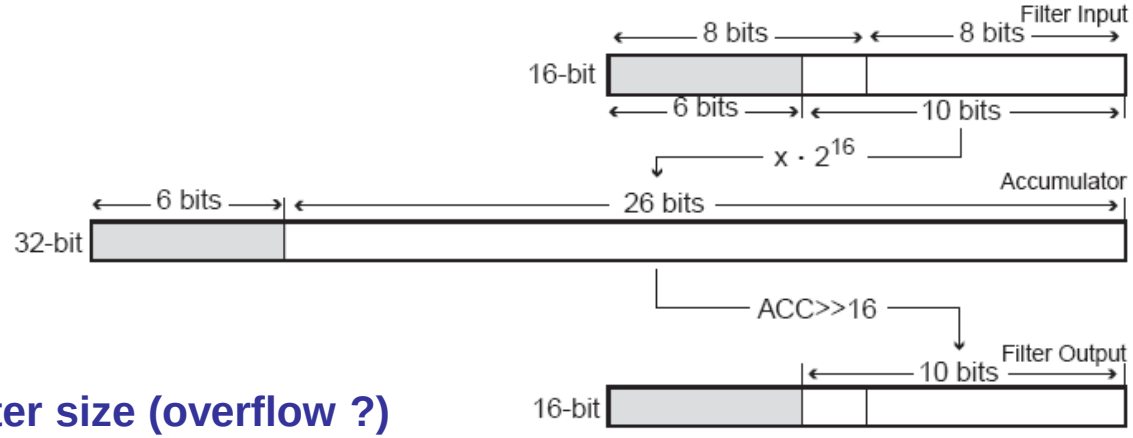




Chebyshev, Butterworth and Bessel characteristics

Implementation Considerations

- floating point multiplication in uCs is usually very slow
- many uCs provide hardware multipliers and fast MAC - operations
- fractional number arithmetics speed up filter calculation -
  scale the input data and coefficients to get the needed precision,
  use integer multiplication, shift back the result:



Issues:

- **scaling factor / register size (overflow ?)**
- **resulting resolution**

Atmel Application note AVR223 – Digital Filters with AVR

http://www.atmel.com/dyn/resources/prod_documents/doc2527.pdf

Design Digital FIR Filters

60 Hz notch example

Frequencies that define complex zeros:

$$\omega_0 := 2 \cdot \pi \cdot \frac{f_0}{f_s}$$

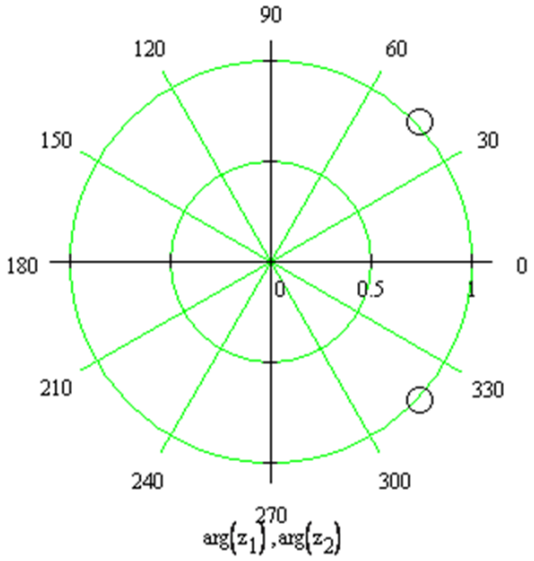
f0=60Hz - power supply frequency
fs=500Hz - sampling rate

we get w0 = 0.754

Positions of complex zeros:

$$z_1 := \cos(\omega_0) + j \cdot \sin(\omega_0)$$

$$z_2 := \cos(\omega_0) - j \cdot \sin(\omega_0)$$



Matlab-source: http://www.scienceprog.com/category/biomedical-dsp

Digital FIR Filters

60 Hz notch filter example

System Function:

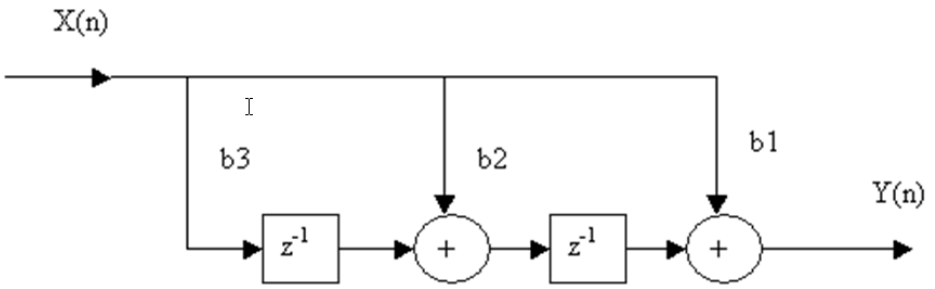
$$H(z) = \frac{Y(z)}{X(z)} = \frac{(z - z_1)(z - z_2)}{z^2}$$

$$\frac{(z - z_1)\cdot(z - z_2)}{z^2} = \frac{z^2 - z\cdot z_2 - z_1\cdot z + z_1\cdot z_2}{z^2} = 1 - z^{-1}\cdot(z_2 + z_1) + z^{-2}\cdot z_1\cdot z_2$$

$$1 + z^{-1}\cdot(-2\cdot\cos(\omega_0)) + z^{-2}$$

Filter Coefficients:

$$b_3 := 1 \quad b_2 := -2\cdot\cos(\omega_0) \quad b_1 := 1$$

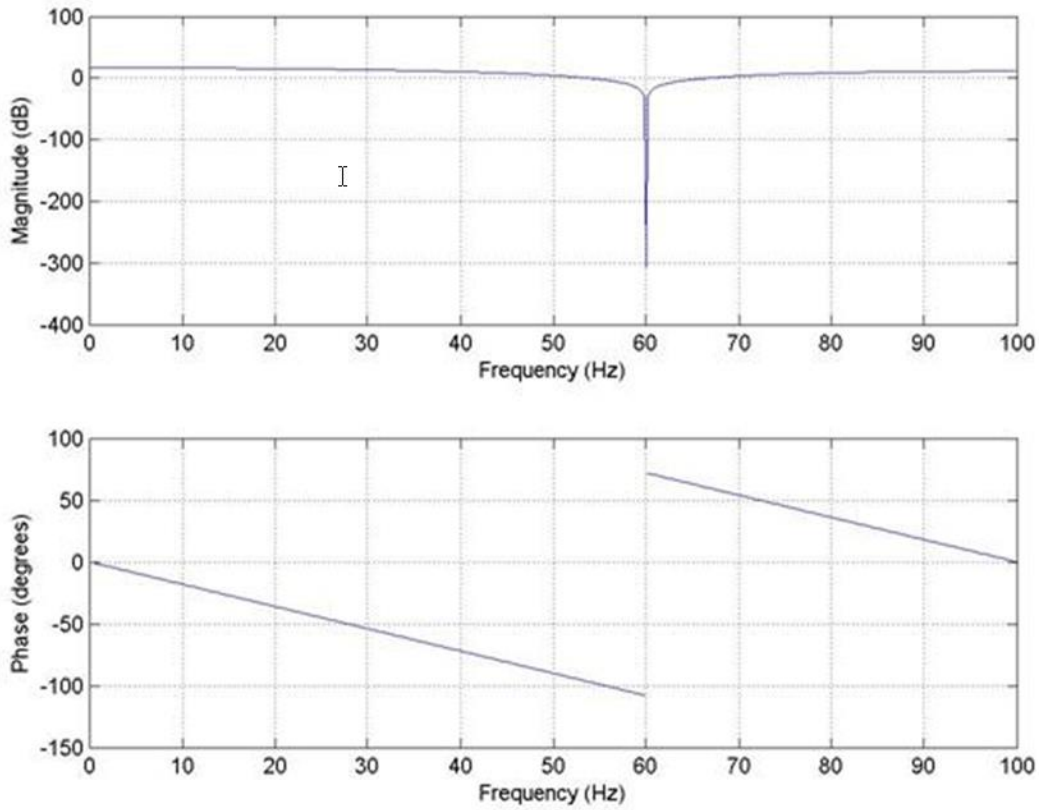


$$G := \frac{1}{(2 - 2\cdot\cos(\omega_n))}$$

scaling: $G = 1.845$

Digital FIR Filters

60 Hz notch filter example



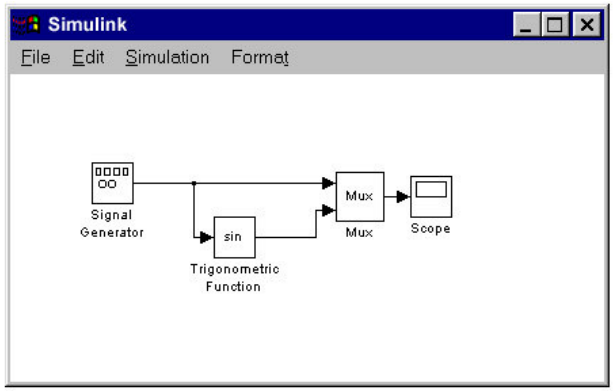
Resulting Filter characteristics

Filter Design with Matlab / Simulink

- Filter Design Functions:

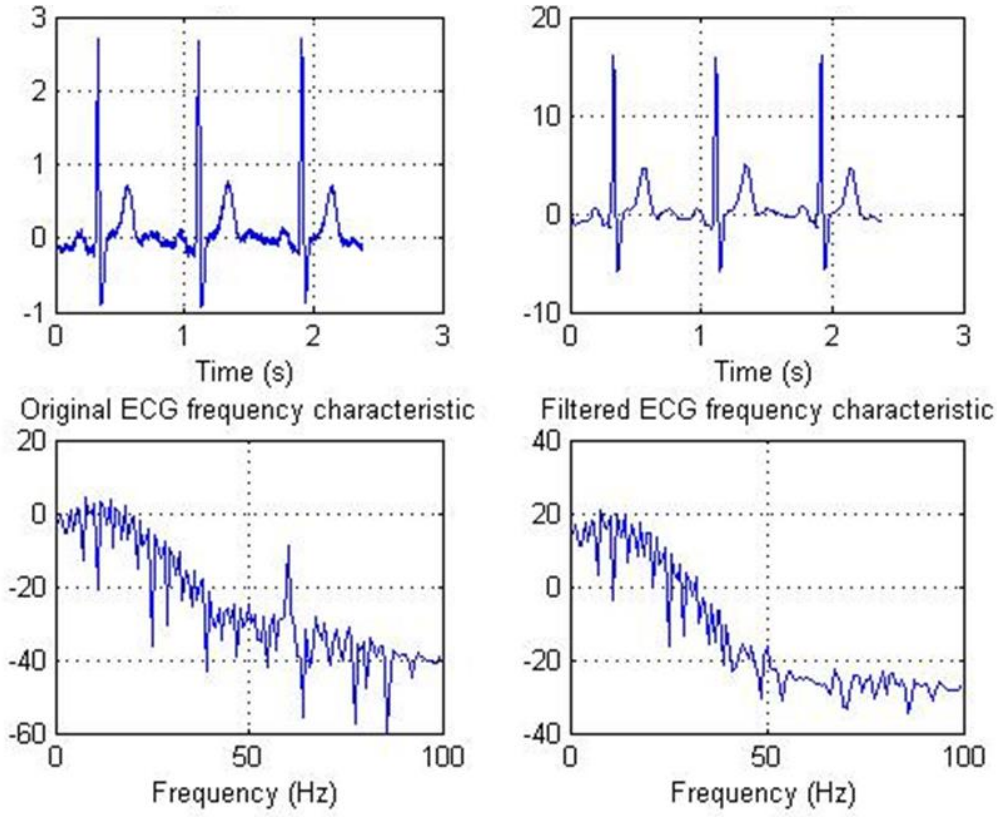
```
h= fdesign.bandpass('N,Fc1,Fc2', N, Fc1, Fc2, Fs);
Hd = design(h, 'butter');
y=filter(Hd,x);

b = fir1(N, Fc/(Fs/2), 'high', win, flag);
Hd = dfilt.dffir(b);
y=filter(Hd,x);


[b,a]= butter (N,0.1,'high');
y=filter(b,a,x);
```

- Filter Design and Analysis Tool (fdatool)
- Signal Processing Toolbox
- Simulink Signal Processing Blocksets:



Filter Implementation in Matlab

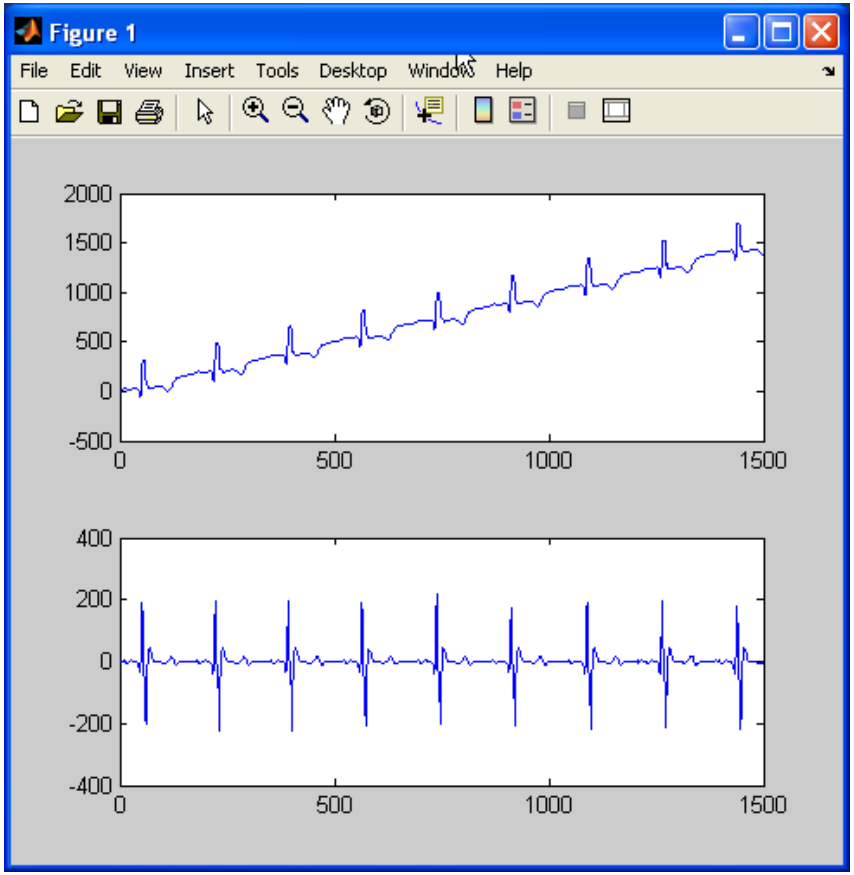
```
load ecg_signals.mat;
dim=size(ecg);

t=0:1/fs:(dim(2)/fs-1/fs);
figure
subplot(2,2,1); plot(t, ecg);
title(' Original ECG');
xlabel('Time (s)');

w0=2*pi*((60)/(fs));
G=1/(2-2*cos(w0));

z1=cos(w0)+j*sin(w0);
z2=cos(w0)-j*sin(w0);

Fecg=filter
([1/G, -2*cos(w0)/G,1/G],1,ecg);
subplot(2,2,2); plot(t, Fecg);
title('Filtered ECG signal');
xlabel('Time (s)');
```



60Hz notch applied to ECG signal

Filter Implementation in Matlab

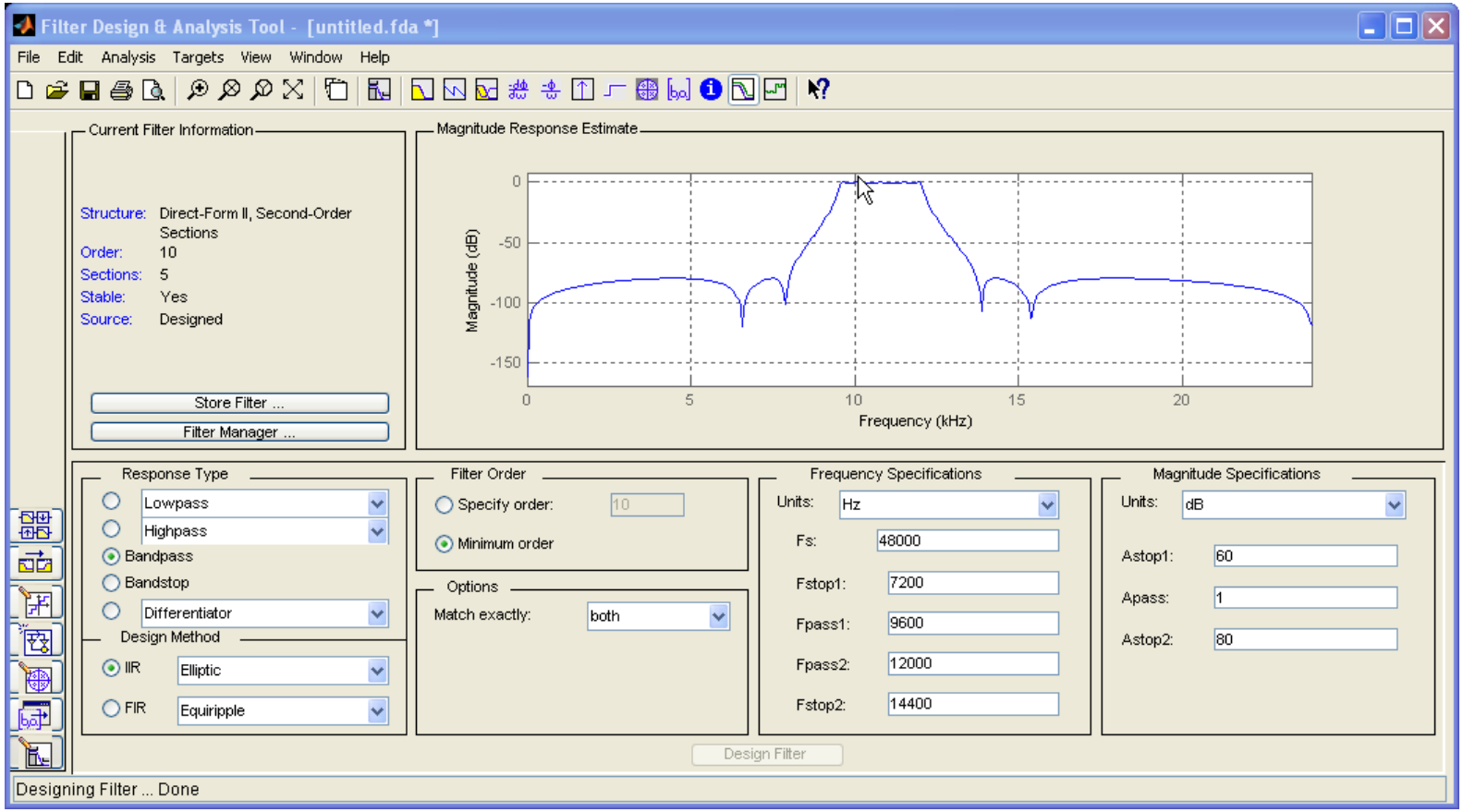
```
fid = fopen('ekg.txt','r');
InputText=textscan(fid,'%f',1500,
'delimiter','\n');
x=cell2mat(InputText);
fclose(fid);
subplot(2,1,1);
plot(x);

[b,a]= butter (2,0.1,'high');
% 0.1 = 12,8 Hz at 256 Hz
y=filter(b,a,x);

subplot(2,1,2);
plot(y);
```



Example file: read_file_filter.m

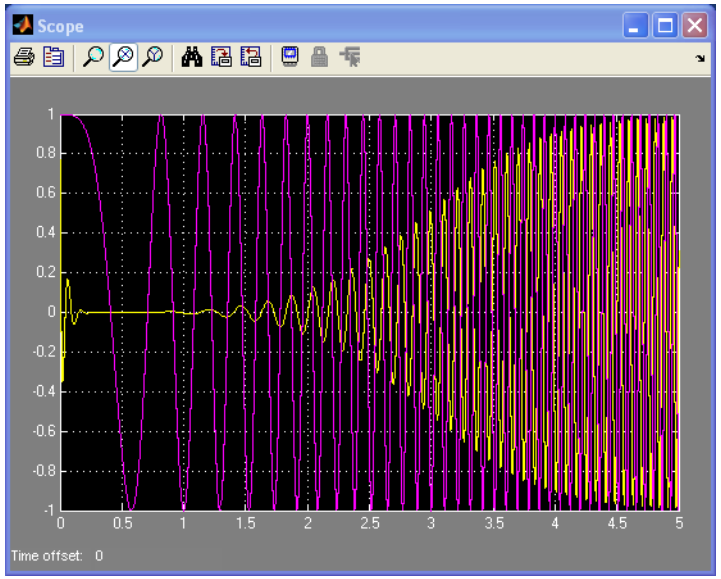
highpass for ECG,
signal parsed from a text file

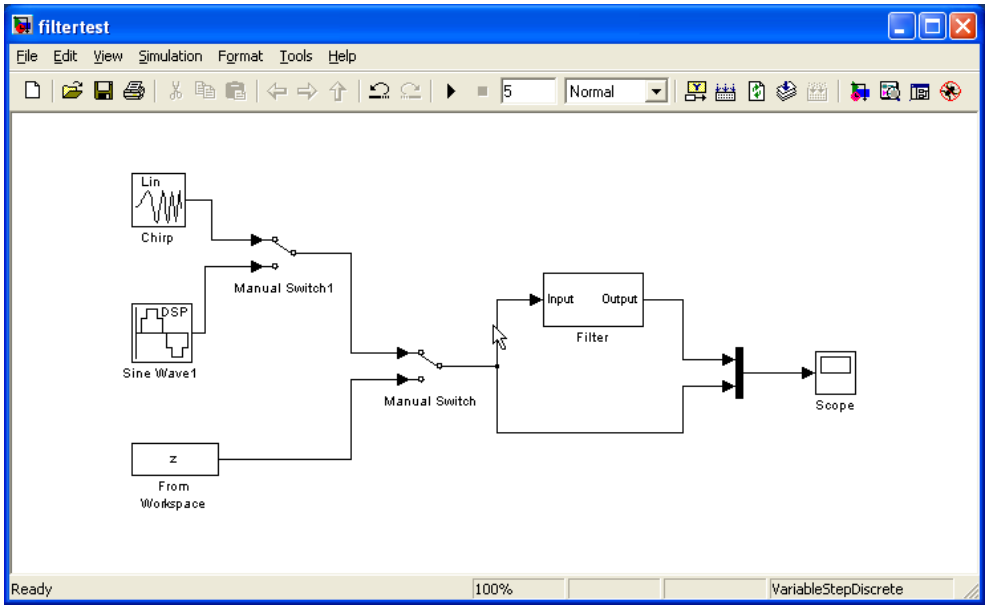
Filter Implementation in Matlab



Fdatool: Filter Design and Analsys, export to Simulink / Workspace

Filter Implementation in Matlab





Filter Application and Test in a Simulink Model

Example file: filtertest.mdl

Open Source Alternatives: FiView and FidLib

- written by Jim Peters, part of the OpenEEG Source pool
- cross platform compatible (using SDL-Library)
- graphical comparison of different filters, testing with feed-signals
- example invocation for a 4-filter comparison:

```
fiview 256 -f 10 -i LpBe4, LpBe6, LpBu4, LpBu8
```

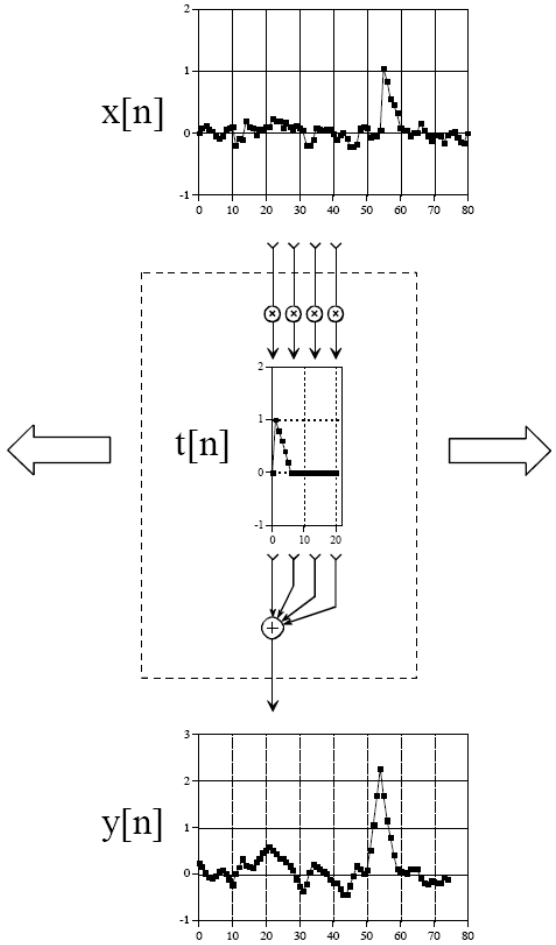
- generates source code (C – functions)
- templates for standard filter types, creation by coefficients

Download Link http://uazu.net/fiview/

Other Signal Processing Techniques

Correlation

- same operation as convolution, but non-flipped multiplication
- finds similar signals in a signal (cross correlation)
- finds perodic parts of a signal (auto-correlation)



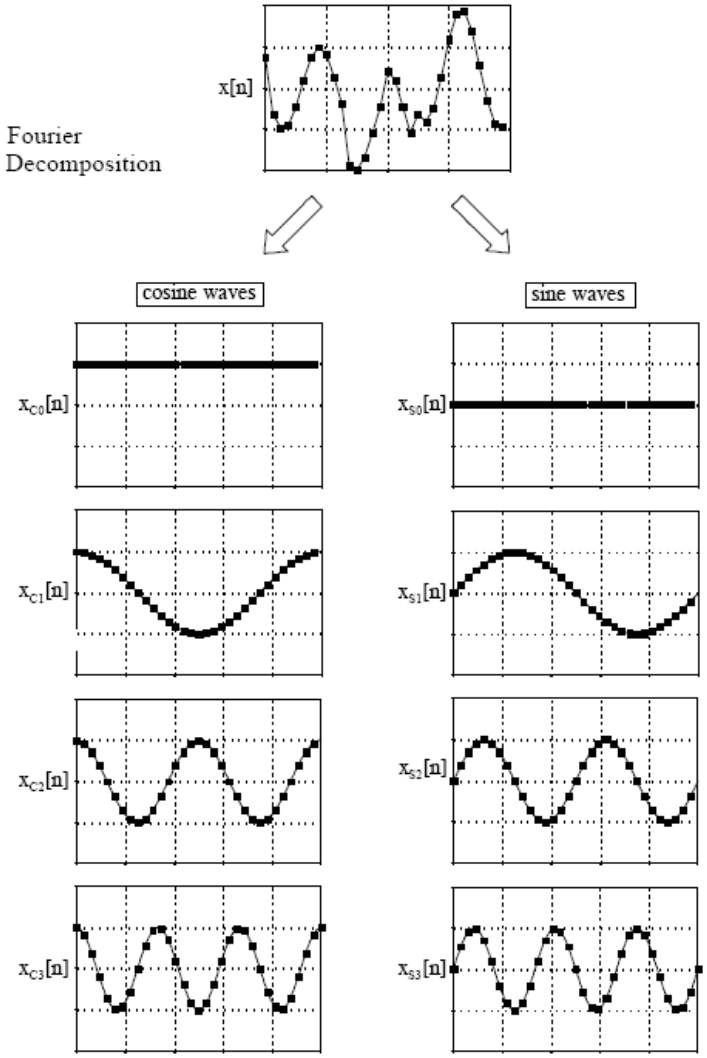
Discrete Fourier Transform

- **Decomposition into sine- and cosine waves**

$$c_k = \cos(2\pi ki/N)$$
$$s_k = \sin(2\pi ki/N)$$

k .. base function
i .. sample index
N .. number of samples

- **Finds frequency components of (periodic) signals**
- **Frequencies up to F / 2**



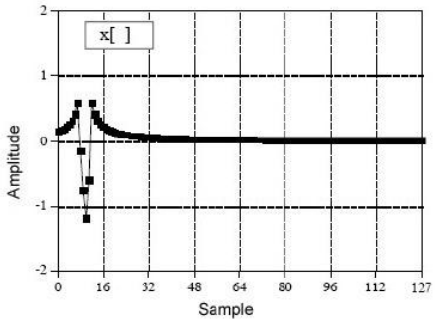
Discrete Fourier Transform

$$\mathrm{Re}\, X[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N)$$

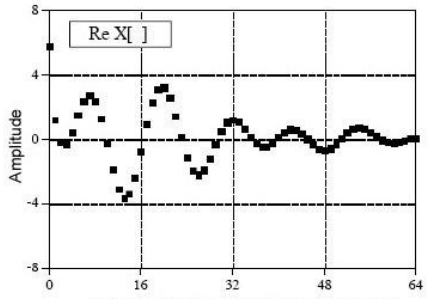
$$\mathrm{Im}\, X[k] = \sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)$$

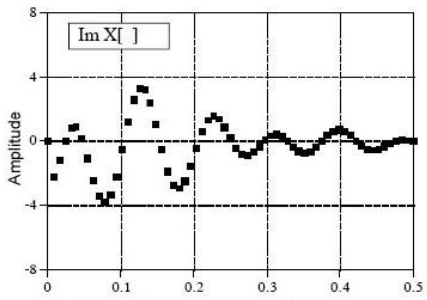
- **Inverse Transform:**

$$x[i] = \sum_{k=0}^{N/2} \mathrm{Re}\, X[k] \cos(2\pi k i / N) + \sum_{k=0}^{N/2} \mathrm{Im}\, X[k] \sin(2\pi k i / N)$$



DFT





- **FFT-Algorithms:**

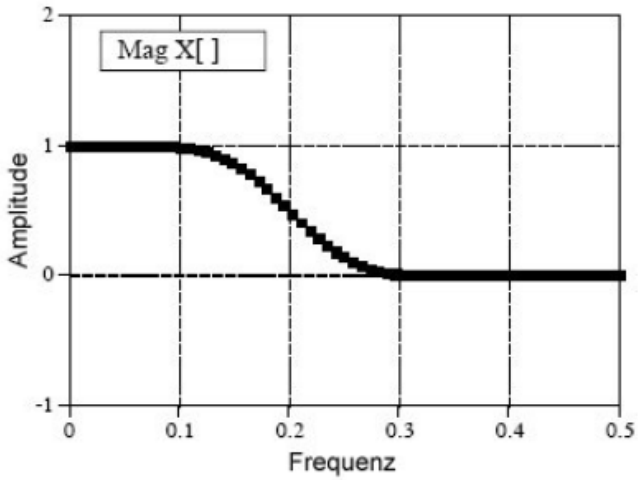
  FFTW, FFTPACK,
  Green, Ooura, Sorensen

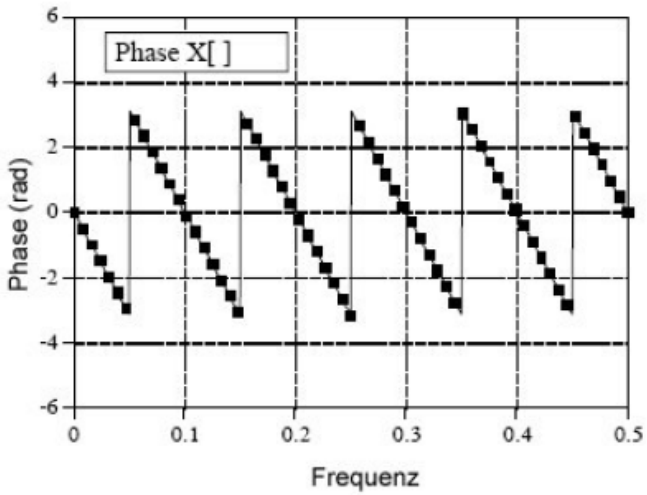
Discrete Fourier Transform

- **Calculation of Magnitude and Phase response:**

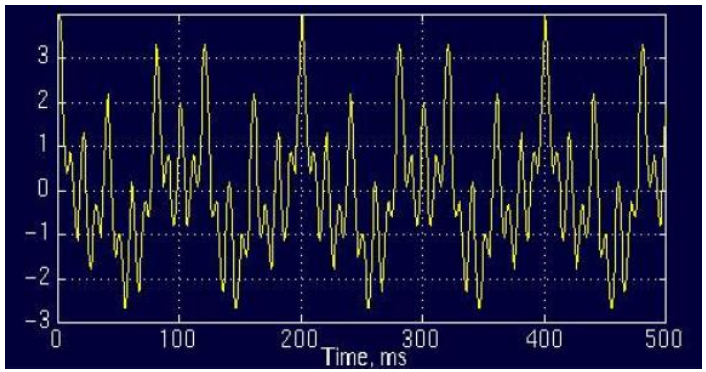
$$MagX[k] = \sqrt{(\mathrm{Re}\ X[k]^2 + \mathrm{Im}\ X[k]^2)}$$

$$PhaseX[k] = \arctan\left(\frac{\mathrm{Im}\ X[k]}{\mathrm{Re}\ X[k]}\right)$$

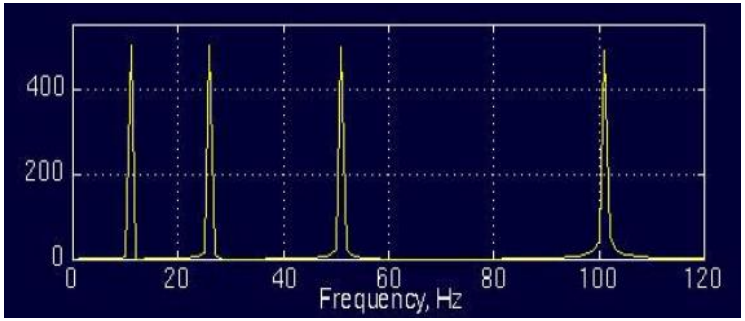




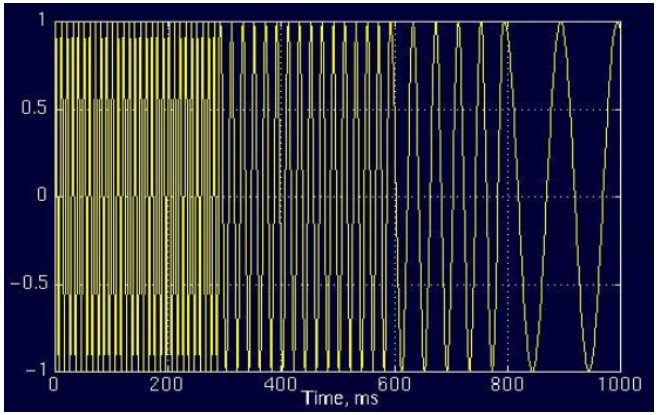
Discrete Fourier Transform - Problems



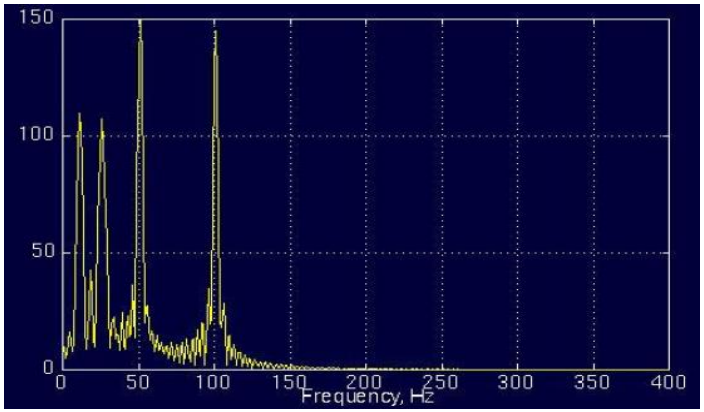
FFT



Stationary signal: correct representation of 4 frequency-components



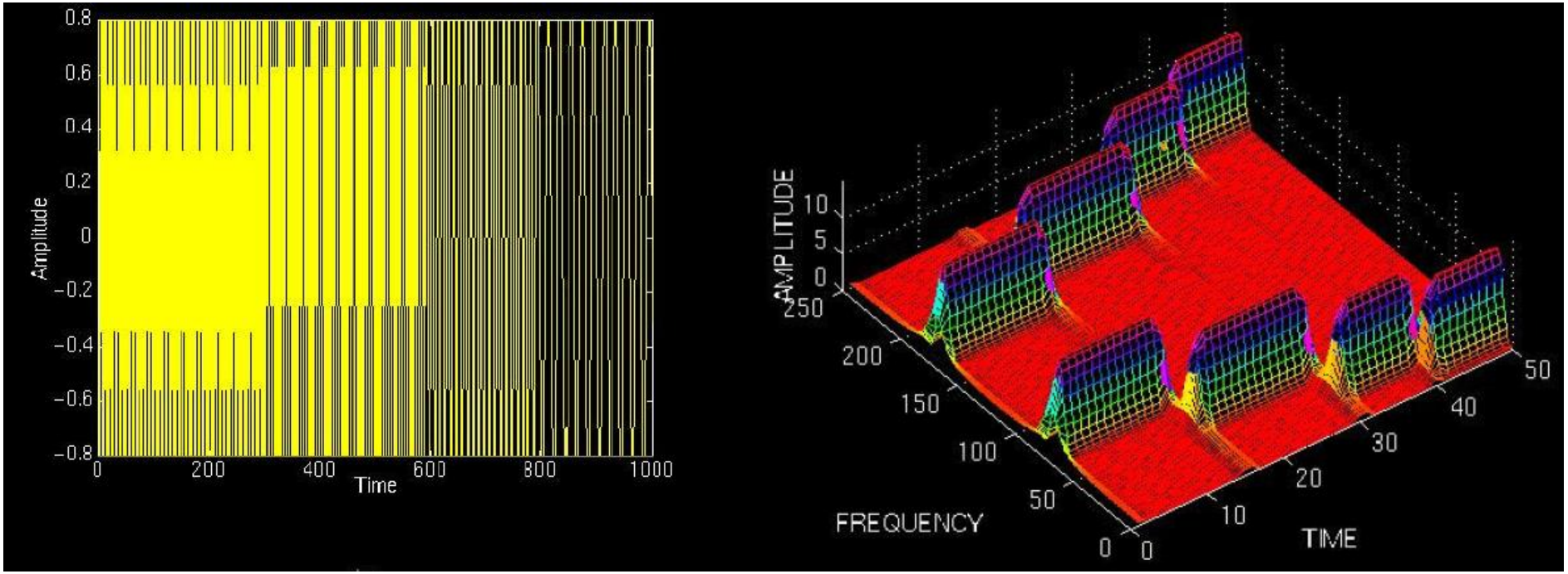
FFT



Problems with non-stationary signals

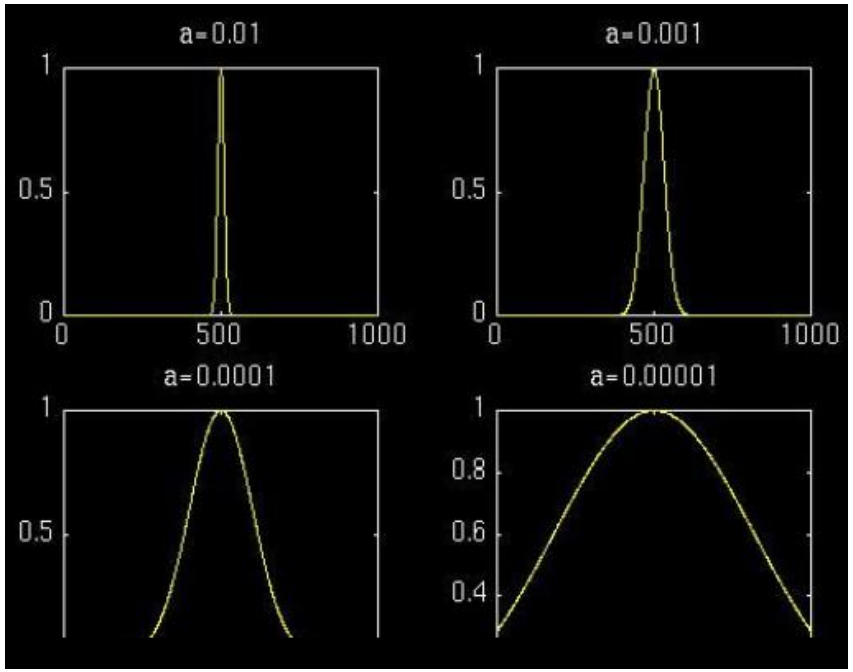
Discrete Fourier Transform -> STFT

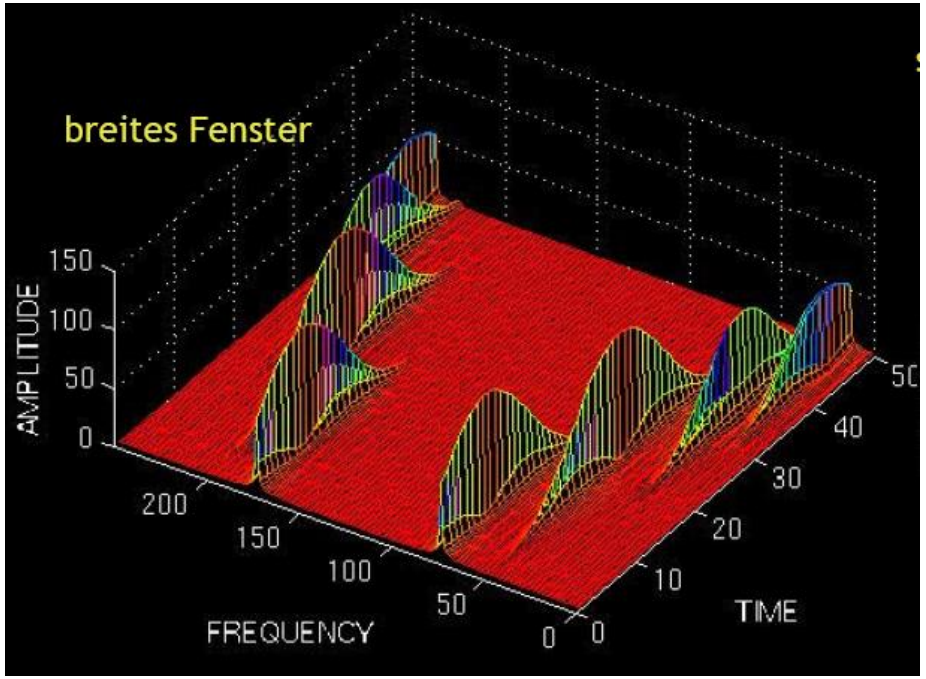
- **Solution: Short Term Fourier-Transform:**
  windowing is used to analyse small portions of (aperiodic) signals



Discrete Fourier Transform

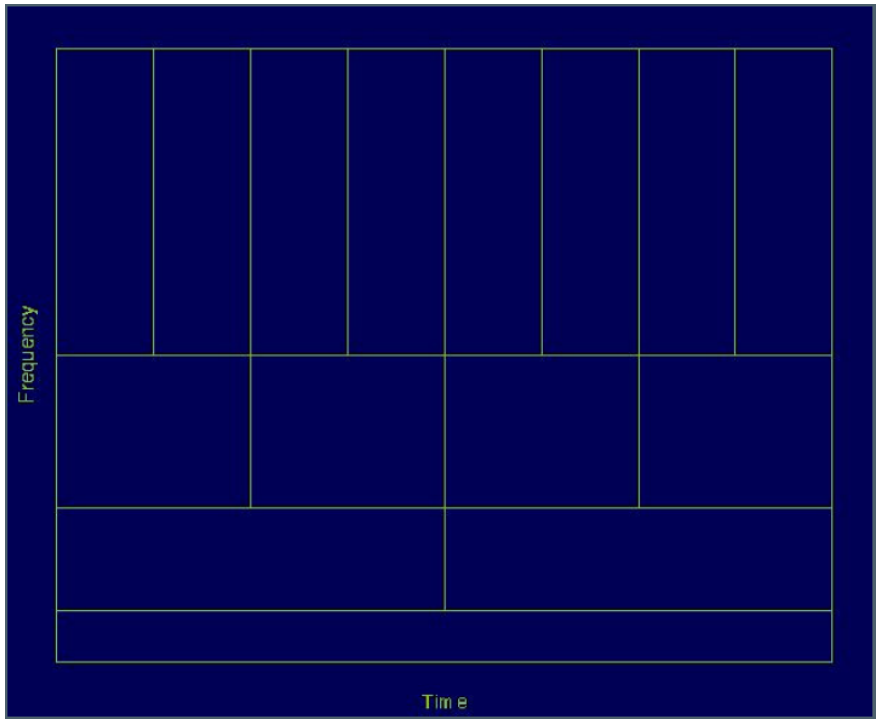
- **Problem with Short Term Fourier-Transform:**
  window-size is fixed and determines tradeoff in resolution betewwn time and frequency:

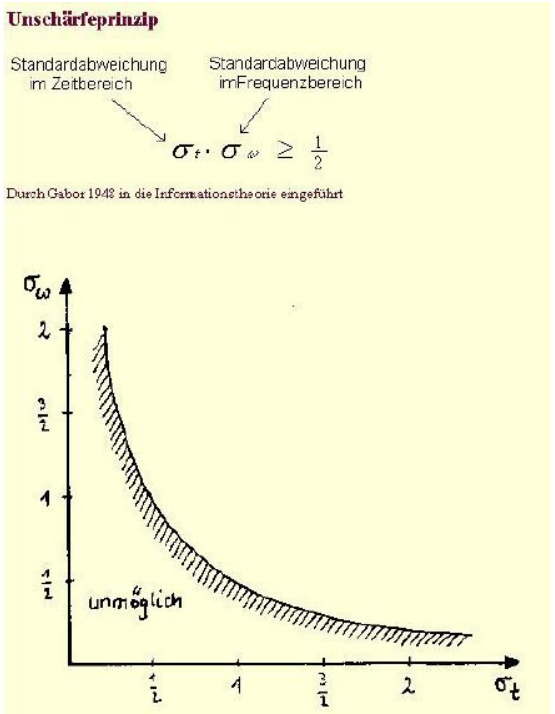




Wavelet Transformation

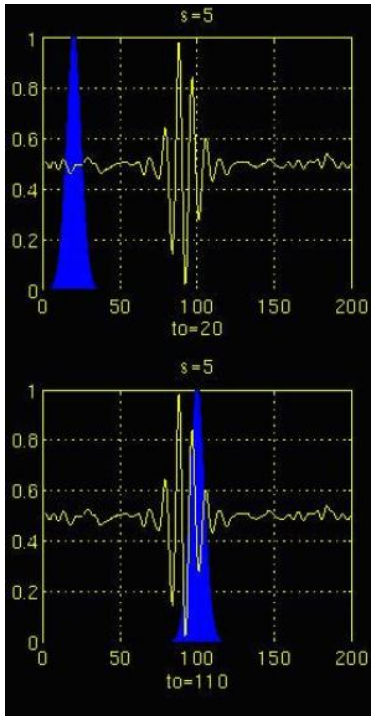
- good frequency resolution at low frequencies and
- good time resolution at high frequencies
- no work-around for the principle of entropy

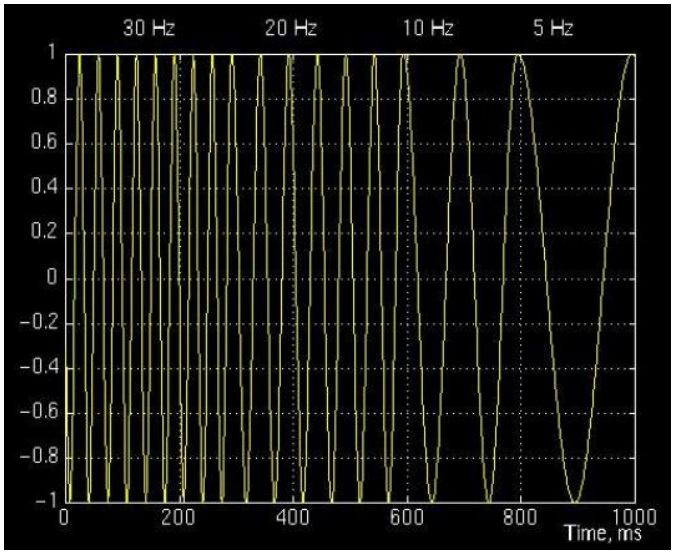


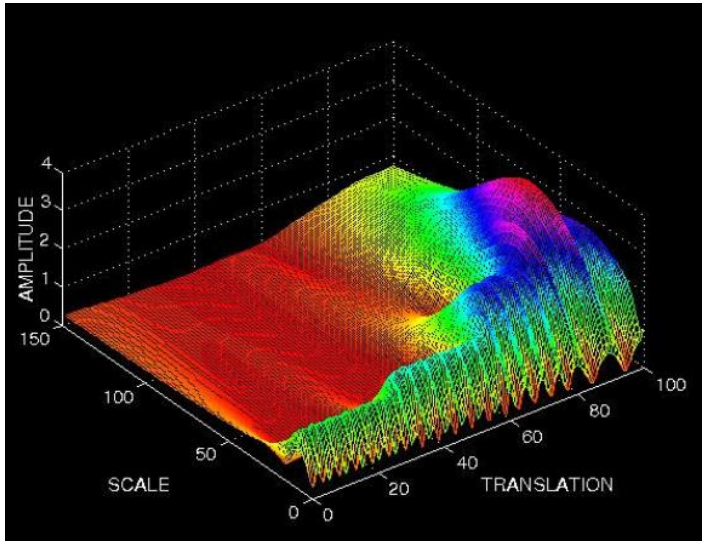


Wavelet Transformation

- scale (s) and translation (t) of the base wavelet
- convolution with the signal
- special wavelets for special purposes







A quick glance at

Classification - Methods

Classification - Methods

- Input data: time or frequency domain characteristics
- purpose: clustering, event detection, data reduction
- get „semantic information“ out of our data
- achive a high detection rate ->
  few false-positive or false-negative classifications
- feature extraction is used to handle huge amounts of data
  -> reduce the feature space

Classification - Methods

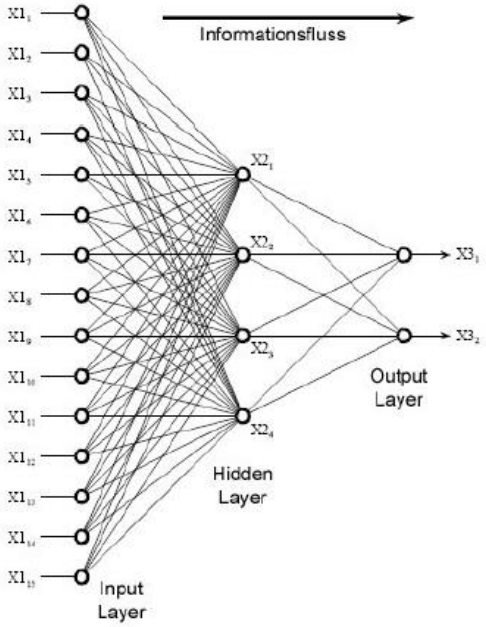
Simple classification could be based on:

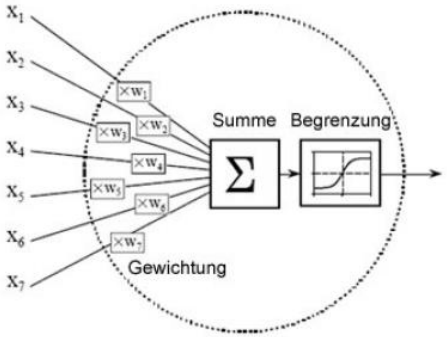
- **Thresholds (levels / intervals, adaptive thresholds)**
- **absolute values + averaging over intervals**
- **integration / difference**
- **local minima / maxima, zeros in time domain**
- **Energy, energy distribution over frequency bands**

Classification - Methods

Neural Networks:

- mimic biological signal processing
- Input-, hidden and output-layers
  units with activation functions
- learning algorithms e.g.
  error back propagation
  unsupervised learning / clustering
- internal representation unrevealed
- pattern recognition, predicition

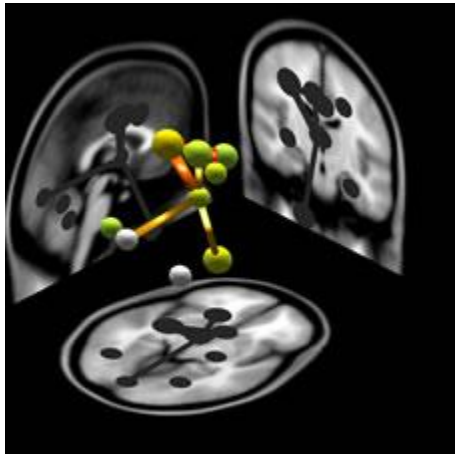




Classification - Methods

Principal Component Analysis and Independent Component Analysis:

- **reduction of complexity of the feature-space**
- **singular-value decomposition delivers component functions (base-functions) that can restore the relevant information with less features**
- **ICA: non-orthogonal base functions allowed, used for blind-source separation**
  EEG: artefact removal, signal source models



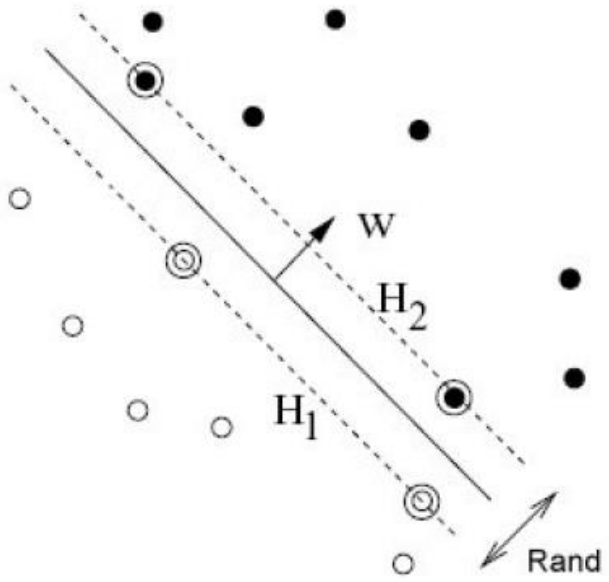
http://www.eurekalert.org

Scott Makeig: http://www.sccn.ucsd.edu/~scott/

Classification - Methods

Suport Vector Machines (SVMs):

- binary classificatin of an input vector
- training with classified data seperates the feature space in two areas, with maximal distance of positive / negative classifications
- SVMs find a global minimum (in contrast to e.g. neural networks)

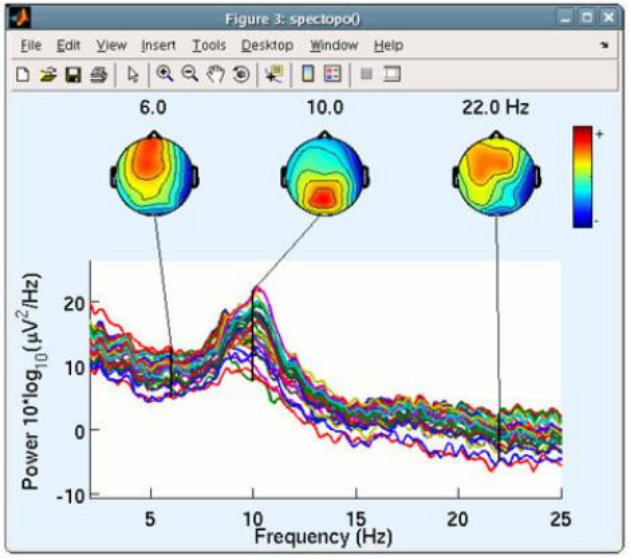


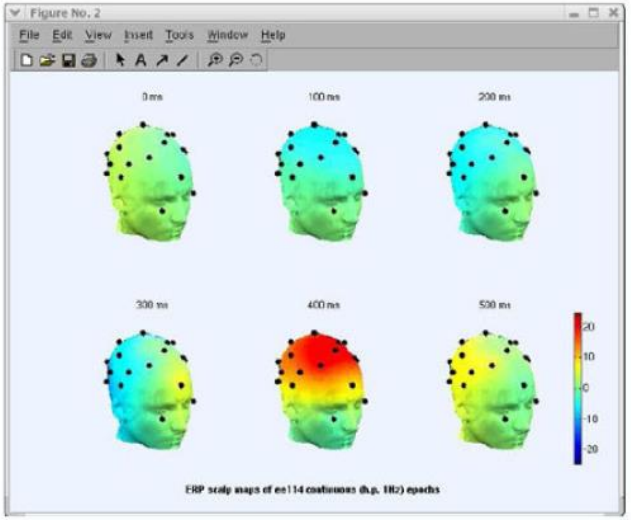
Tools and Libraries

EEGLab:

- **Matlab-based, open source application**
- **300 functions for data analsysis and feature extraction e.g:**
  - **epoch based ERP averaging**
  - **ICA-methods for source localisation**
  - **handling of MRI-data**
- **imports command file formats like EDF, BDF, …**

http://www.sccn.ucsd.edu/eeglab/

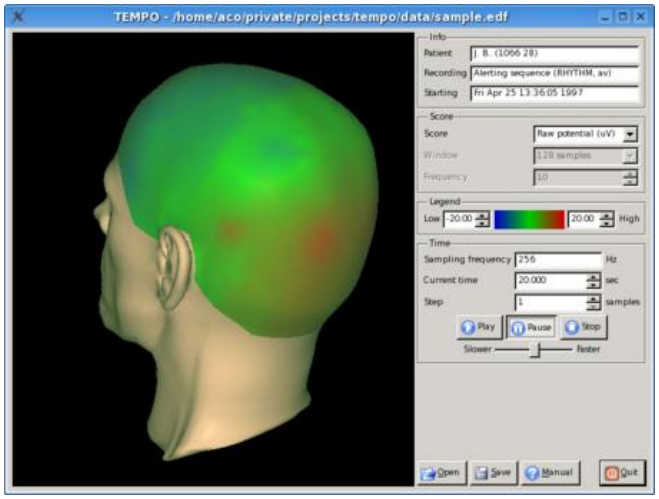


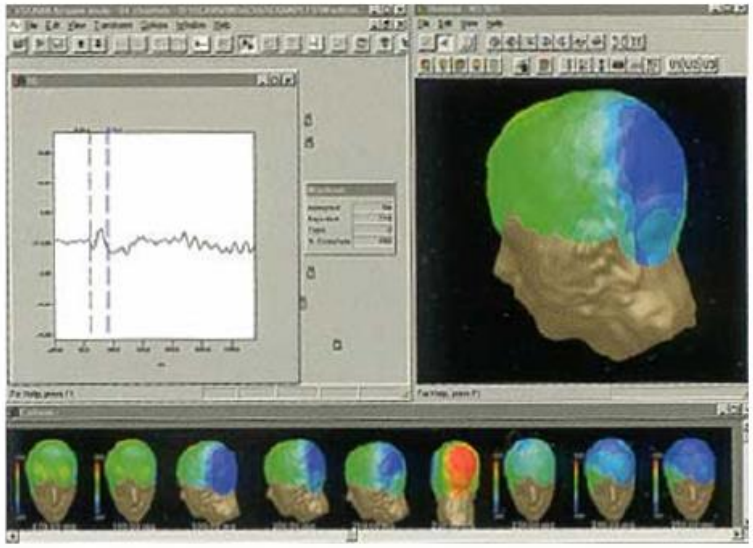


Tools and Libraries

Tempo:

- **Topographic EEG Mapping Program (open source)**
- **3d display of MRI head models,**
- **generates animated sequences**
- **imports EDF**



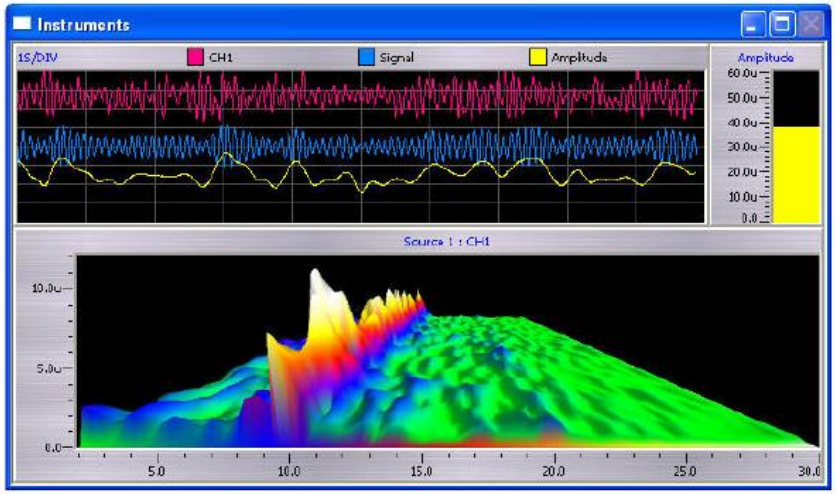


http://tempo.sourceforge.net/

Tools and Libraries

BioExplorer:

- **Real-time biosignal aquisition, analysis and classification**
- **FFT, filtering, correlation, adaptive thresholding**
- **File operations**
- **visual and acoustic biofeedback**
- **compatible with OpenEEG-hardware**



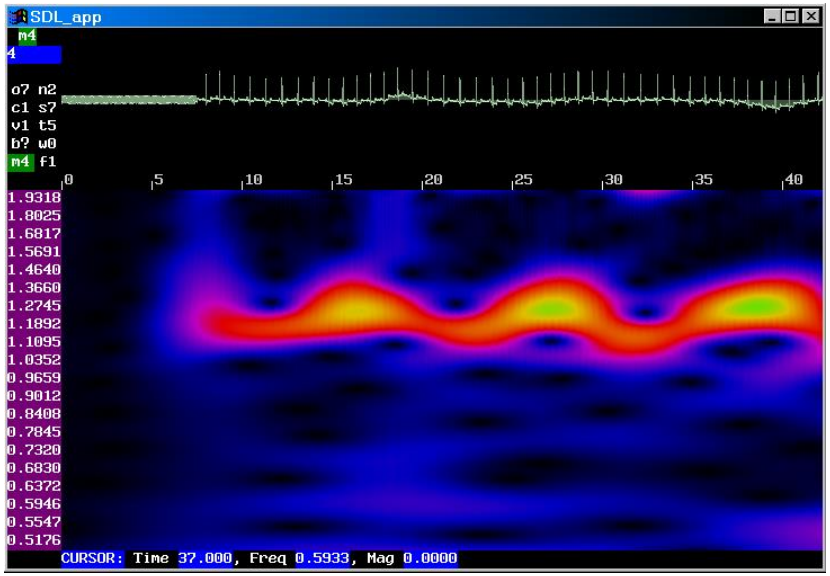
http://www.cyberevolution.com/

Tools and Libraries

BWView:

- **Wavelet-based signal analysis tool using fft-accelerated convolution**
- **open source**
- **compatible to OpenEEG-hardware**

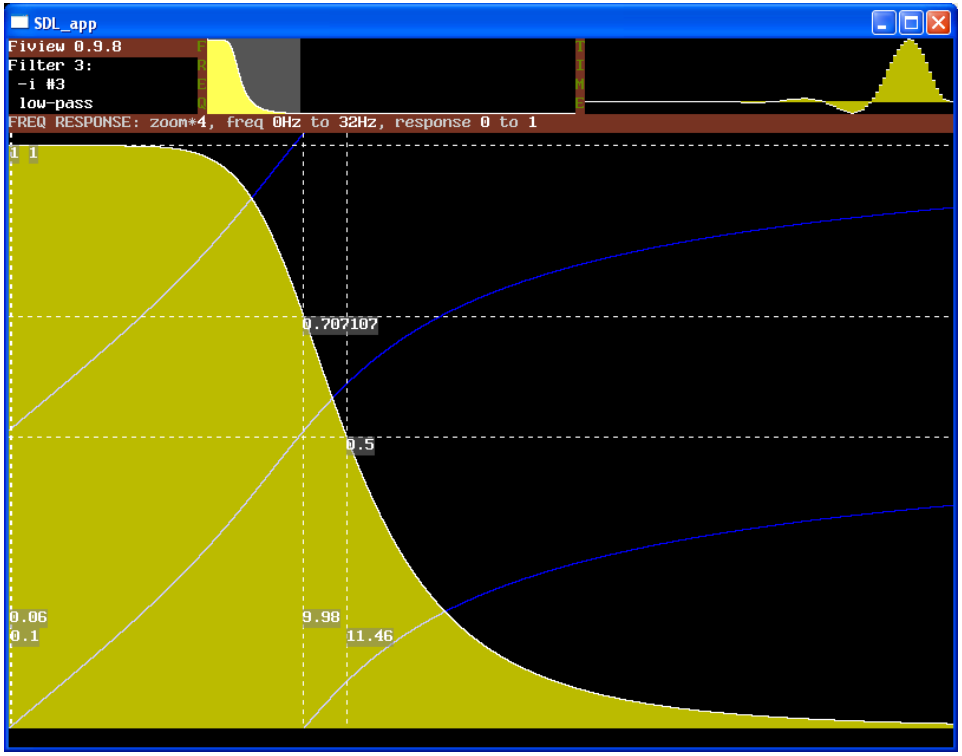
http://uazu.net/bwview/

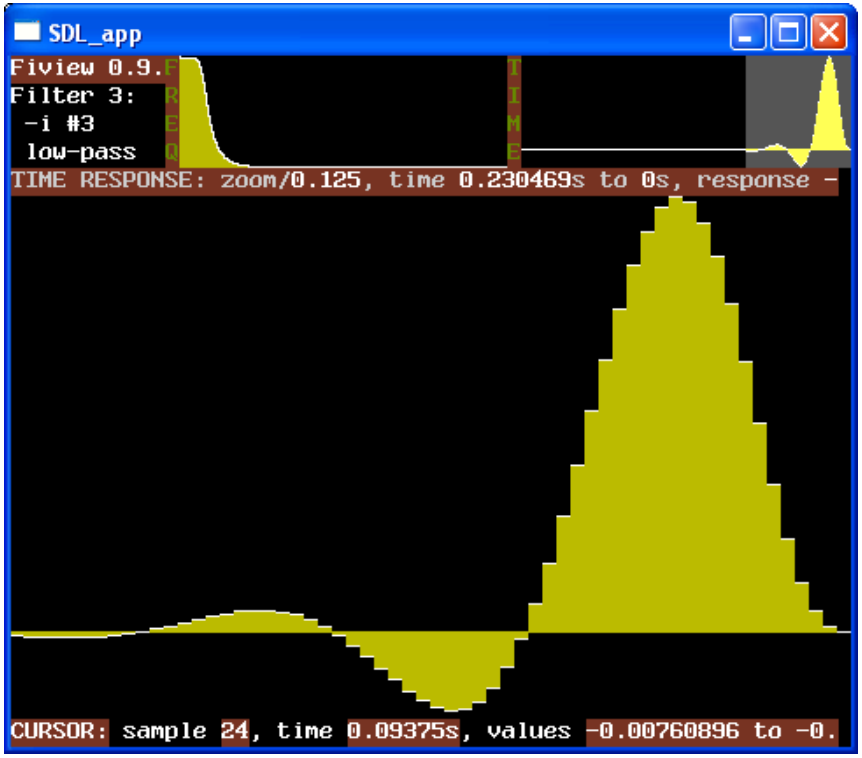


Respiratory sinus arrythmia (RSA) seen with BWview
http://jhansmann.de/eeg/experiments/index.html

Tools and Libraries

FiView and FidLib – IIR filter design tool by Jim Peters http://uazu.net/fidlib/





Frequency response and Impulse response, calculated and viewed with FiView

Tools and Libraries

FiView and FidLib – IIR filter design tool by Jim Peters

```
// Example code (readable version)
double
process(register double val) {
   static double buf[4];
   register double tmp, fir, iir;
   tmp= buf[0]; memmove(buf, buf+1, 3*sizeof(double));
   val *= 0.00069188043817877 58;
   iir= val+1.496016726996244*buf[0]-0.6177887989473995*tmp;
   fir= iir+buf[0]+buf[0]+tmp;
   tmp= buf[1]; buf[1]= iir; val= fir;
   iir= val+1.415382337323265*buf[2]-0.5062905959533784*tmp;
   fir= iir+buf[2]+buf[2]+tmp;
   buf[3]= iir; val= fir;
   return val;
}
```

Example Filter code for a butterworth IIR filter, generated with FiView

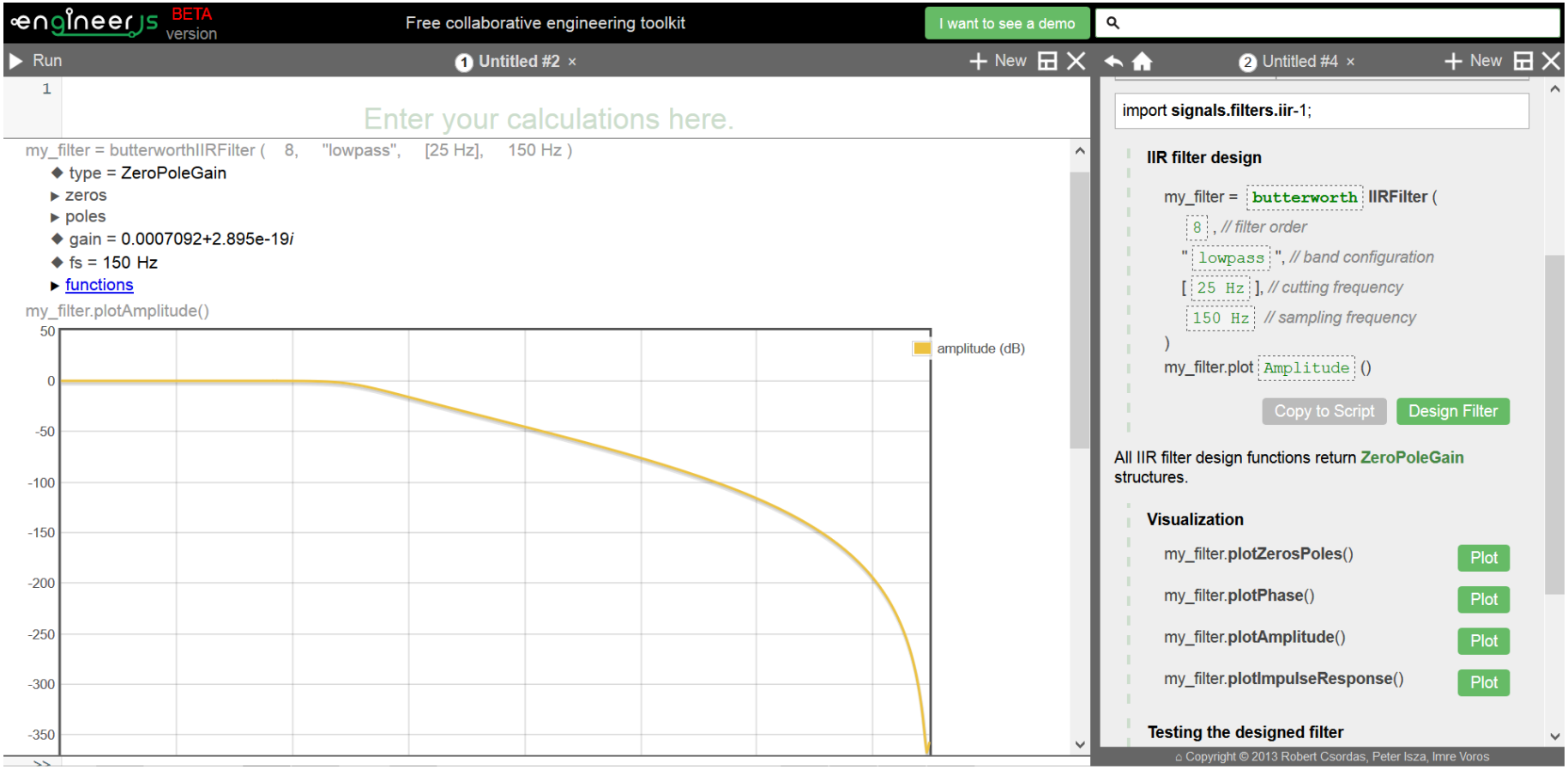
Tools and Libraries

```
#include "fidlib/fidlib.h"    // May need adjusting
FidFilter *
setup() {
   FidFilter *filt0= fid_design("LpBe4", 256, 10, 0, 0, 0);
   return filt0;
}
...
   FidFilter *filt= setup(); // Run a couple of instances using fidlib
   FidFunc *funcp;
   FidRun *run= fid_run_new(filt, &funcp);
   void *fbuf1= fid_run_newbuf(run);
   void *fbuf2= fid_run_newbuf(run);
   while (...) {
      out_1= funcp(fbuf1, in_1);
      out_2= funcp(fbuf2, in_2);
      if (restart_required) fid_run_zapbuf(fbuf1);
      ...
   }
   fid_run_freebuf(fbuf2);
   fid_run_freebuf(fbuf1);
   fid_run_free(run);
```

FidLib C-library provides efficient generic filter creation at runtime

Tools and Libraries

EngineerJS: Online JavaScript IIR filter design Tool



http://engineerjs.com/?sidebar=docs/iir.html