

# Speech signal processing using MATLAB

Basics and applications

Stephan Kuberski

(`kuberski@uni-potsdam.de`)

April/May 2016

Slides and MATLAB scripts at <https://github.com/murtex/spl>

## Digital signals

- Sampling

- Time domain

- Frequency domain

- Filters

- Noise

## Acoustic signals

- Short-time analysis

- Spectrograms

- Activity detection

- Landmarks detection

- Formants detection

# Digital signals/Sampling

- ▶ **continuous signal** (normalized magnitude, length  $L$  in seconds)

$$x(t) \in [-1, 1] \quad \text{with} \quad t \in [0, L]$$

```
>> x = @( t ) sin( 2*pi*f * t ); % continuous sine with frequency f
```

- ▶ **sampling rate**  $f_S$ , quantization of time

$$t \rightarrow t_i = \frac{i-1}{f_S} \quad \text{with} \quad i \in \{1, \dots, N\} \quad \text{and} \quad N = \lfloor Lf_S \rfloor$$

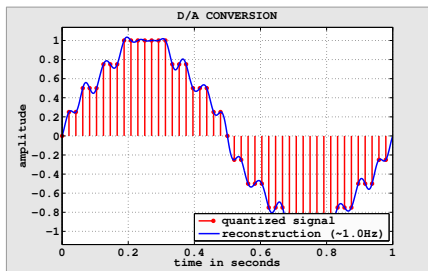
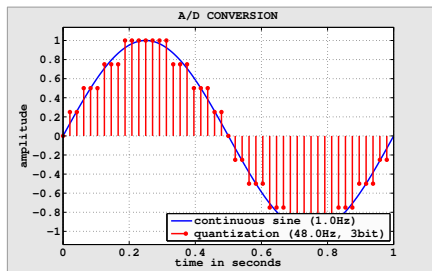
```
>> N = floor( L * fS ); % number of samples  
>> ti = (0:N-1) / fS; % quantized time values
```

- ▶ **bits per sample**  $n_S$ , quantization of amplitude

$$x(t) \rightarrow x_i = \frac{\lfloor 2^{n_S-1} x(t_i) \rfloor}{2^{n_S-1}}$$

```
>> xi = round( 2^(nS-1) * x( ti ) ) / 2^(nS-1); % quantized amplitudes
```

- ▶ example: matlab/sampling.m



- ▶ exercise:

- ▶ verify from reconstruction that Nyquist frequency holds

$$f_{\text{Ny}} = \frac{f_s}{2}$$

- ▶ compare commonly used **sampling standards** (telephony, Audio-CD, professional audio equipment, ...)

## Digital signals/Time domain

- ▶ **total energy, average power and root mean square**

$$E = \sum_{i=1}^N x_i^2, \quad P = \frac{1}{N} \sum_{i=1}^N x_i^2 \quad \text{and} \quad RMS = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2}$$

```
>> E = sum( xi .* xi ); % total energy  
>> P = mean( xi .* xi ); % average power  
>> RMS = sqrt( mean( xi .* xi ) ); % root mean square
```

- ▶ **decibel full scale, different for power- and magnitude-like quantities, e. g.**

$$P_{\text{dB}} = 10 \log_{10}(P) \quad \text{and} \quad RMS_{\text{dB}} = 20 \log_{10}(RMS)$$

```
>> PdB = 10 * log10( P ); % power-like  
>> RMSdB = 20 * log10( RMS ); % magnitude-like
```

- ▶ **zero-crossings rate**

```
>> fZ = sum( abs( diff( xi >= 0 ) ) ) / N * fS;
```

## Digital signals/Frequency domain



- ▶ **discrete Fourier transform**, time domain  $\rightarrow$  frequency domain

$$X_k = \sum_{i=1}^N x_i e^{-2\pi i \frac{(i-1)(k-1)}{N}} \in \mathbb{C} \quad \text{with} \quad k \in \{1, \dots, N\}$$

```
>> Xk = fft( xi ) / N; % complex Fourier coefficients
```

- ▶  $k$  is a **frequency index** (as  $i$  was a time index)

$$k \rightarrow f_k = \frac{k-1}{N} f_s$$

```
>> fk = (0:N-1) / N * fs; % frequency values
```

- ▶ frequencies beyond Nyquist frequency are **negative frequencies**

$$f_k \rightarrow f_k = \begin{cases} f_k - f_s & \text{if } f_k > f_{Ny} \\ f_k & \text{otherwise} \end{cases}$$

```
>> fk(fk > fNy) = fk(fk > fNy) - fs; % imply negative frequencies
```

- ▶ **power spectral density** (also known as **power spectrum**)

$$P_k = |X_k|^2 \in \mathbb{R} \quad \Leftarrow \quad \sum_{k=1}^N P_k = P$$

```
>> Pk = abs( Xk ) .^ 2; % power spectral density
```

- ▶ **real valued signals** ( $x_i \in \mathbb{R}$ ) imply a special symmetry

$$X_{f_k} = X_{-f_k}^* \quad \Rightarrow \quad P_{f_k} = P_{-f_k}$$

- ▶ restrict to **one-sided spectrum**

```
>> Pk(fk < 0) = []; % remove negative frequency components
>> Xk(fk < 0) = [];
>> fk(fk < 0) = [];
>> Pk(2:end) = 2 * Pk(2:end); % rescale to match total power
>> Xk(2:end) = sqrt( 2 ) * Xk(2:end);
```

- ▶  $P_1$  is **DC offset**,  $P_{k>1}$  are **contributions of sines** with frequencies  $f_{k>1}$

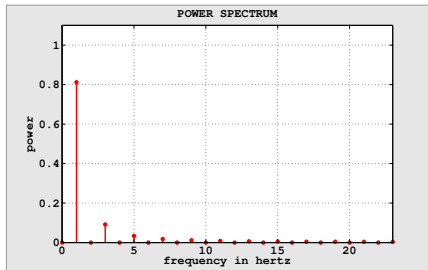
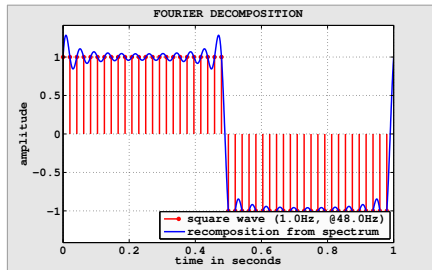
$$x(t) = \sqrt{P_1} + \sqrt{2} \sum_{k>1} \sqrt{P_k} \sin(2\pi f_k t)$$

# Frequency domain

- ▶ complex valued but without loss of phase information

$$x(t) = X_1 + \sqrt{2} \sum_{k>1} X_k e^{2\pi i f_k t}$$

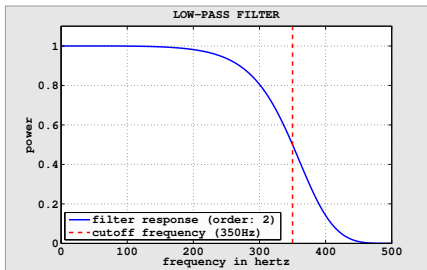
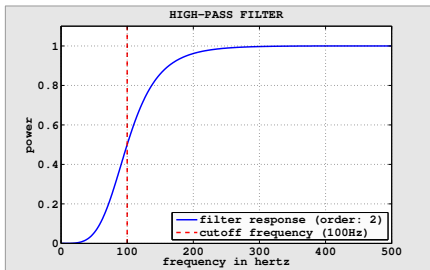
- ▶ **example:** matlab/fdomain.m



- ▶ **exercise:**
  - ▶ is there any loss of information by changing to real values?
  - ▶ TODO!

# Digital signals/Filters

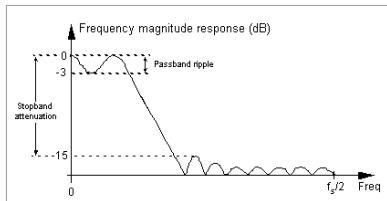
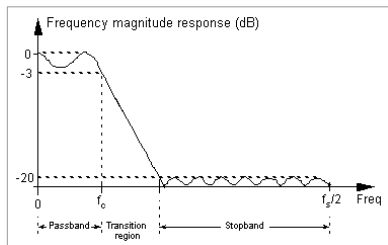
- ▶ **general filter types:**
  - ▶ **low-pass:** passes low frequencies (cuts high ones)
  - ▶ **high-pass:** passes high frequencies (cuts low ones)
  - ▶ **band-pass:** passes a range of frequencies (combination of low- and high-pass)
  - ▶ **band-stop (notch):** cuts a range of frequencies (opposite of band-pass)
- ▶ **cutoff frequency** at which output power is (generally) reduced by  $-3$  dB
- ▶ **example:** `matlab/filters.m`



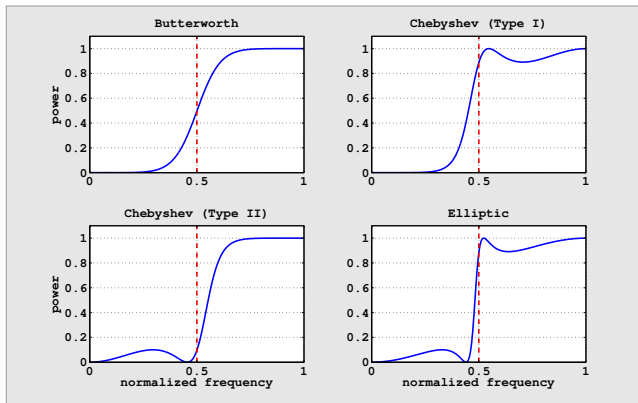
- ▶ filters are represented by **filter coefficients**  $b_j$  (feedforward) and  $a_j$  (feedback)
- ▶ high **filter order**  $m$  increases computational complexity but thereby quality

$$y_i = \frac{1}{a_1} \left( \overbrace{\sum_{j=0}^m b_{j+1} x_{i-j}}^{\text{IIR}} - \underbrace{\sum_{j=1}^m a_{j+1} y_{i-j}}_{\text{FIR}} \right) \quad \text{with } i \in \{1, \dots, N\}$$

- ▶ **FIR filters** (finite impulse response) are slow to compute but stable
- ▶ **IIR filters** (infinite impulse response) are fast to compute but might be unstable
- ▶ some often used **additional terms** (images from <http://dspguru.com>)



- ▶ many filter families with different characteristics (matlab/filters2.m), e. g.



- ▶ normalized frequency

$$\tilde{f}_k = \frac{f_k}{f_{Ny}} = \frac{2f_k}{f_s} \in [0, 1] \quad \text{with} \quad k \in \{1, \dots, N\}$$

- ▶ **Butterworth filter** (high-pass, second-order, 100 Hz cutoff)

```
>> m = 2; % filter order
>> cutoff = 100; % cutoff frequency
>> [b, a] = butter( m, cutoff / (fs/2), 'high' );
```

- ▶ **Chebyshev filter** (high-pass, 1 dB ripple, 40 dB attenuation, 100 Hz cutoff)

```
>> cutoff = 100; % cutoff frequency
>> stopband = 90; % stopband frequency
>> ripple = 1; % passband ripple
>> attenuation = 40; % stopband attenuation
>> m = cheb2ord( cutoff / (fs/2), stopband / (fs/2), ripple, attenuation );
>> [b, a] = cheby2( m, attenuation, stopband / (fs/2) );
```

- ▶ **apply any filter**

```
>> y = filter( b, a, x ); % filter signal x using coefficients a, b
```

- ▶ **or in zero-phase version (without filter delay)**

```
>> y = filtfilt( b, a, x ); % zero-phase filtering
```



► **exercise:**

- Can you image what **filter delay** means?
- TODO: home audio/hifi, bass and treble

# Digital signals/Noise

- ▶ spectral flatness measure

$$SFM = \dots$$

- ▶ spectral entropy using probability densities  $p_k$  and skipping constant DC

$$H = - \sum_{k>1} p_k \log(p_k) \quad \text{with} \quad p_k = \frac{P_k}{\sum_{k>1} P_k}$$

```
>> pk = Pk(2:end) / sum( Pk(2:end) ); % probability densities  
>> H = -sum( log2( pk .^ pk ) ); % entropy in bits
```

- ▶ additive noise
- ▶ signal-to-noise ratio
- ▶ spectral subtraction

## Acoustic signals/Short-time analysis



# Acoustic signals/Spectrograms



## Acoustic signals/Activity detection





## Acoustic signals/Landmarks detection



## Acoustic signals/Formants detection

