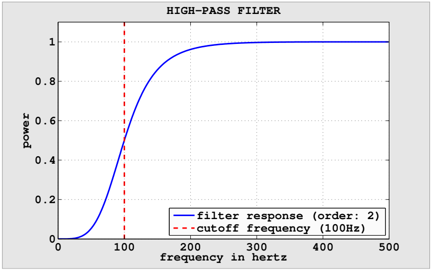
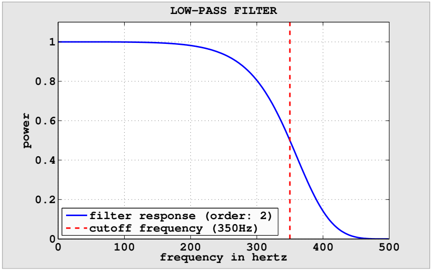
Digital signals/Filters

- 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`



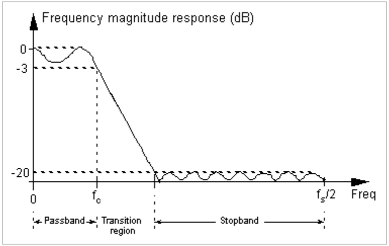


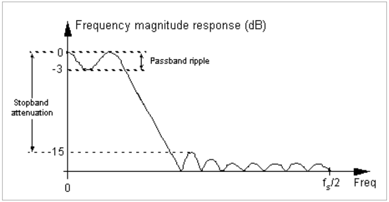
Filters

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

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

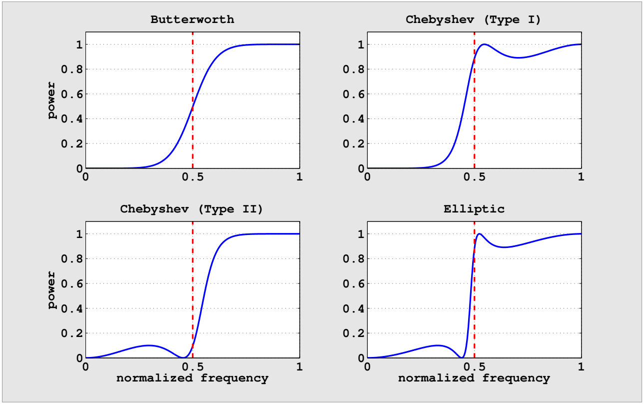
- **FIR filters** (finite impulse response) are slow to compute but stable
- **IIR filters** (infinite impulse response) are fast to compute but might be unstable
- some often used **additional terms** (images from `http://dspguru.com`)





Filters

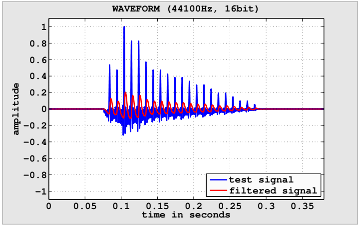
- **example:** `matlab/filters2.m`

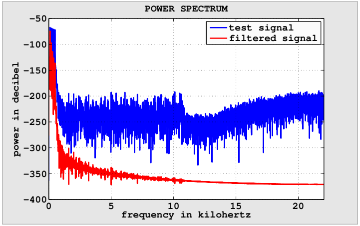


- many **filter families** with different characteristics
- **normalized frequency**

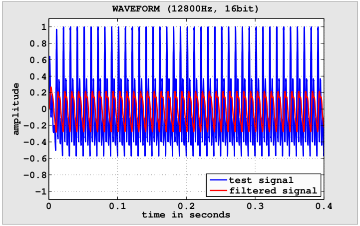
$$\tilde{f}_k = \frac{f_k}{f_{\mathrm{Ny}}} = \frac{2f_k}{f_{\mathrm{S}}} \in [0, 1] \quad \text{with} \quad k \in \{1, \ldots, N\}$$

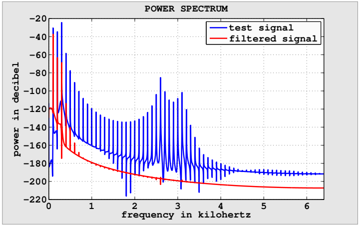
- **example:** `matlab/spectrum.m` (`matlab/sound.wav`)





- **example:** `matlab/spectrum.m` (`matlab/ivowel.wav`)





- **exercise:**
  - observe the occurrence of **filter delay**

Filters

- **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
```