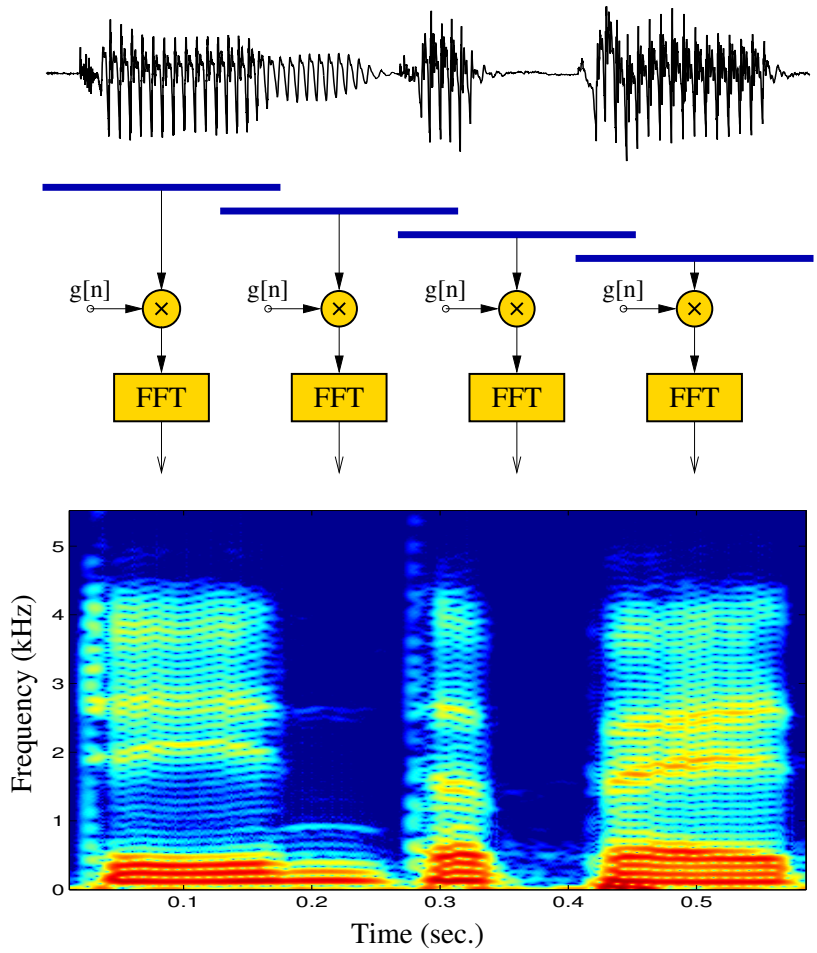
G. Doblinger TU-Wien

Signal Processing Using MATLAB®



Lecture notes, collection of problems, projects, and supplementary course material

Signal Processing Using MATLAB®, version Sept. 2014

supplement to courses 389149, 389150 "Signal Processing Using MATLAB®" presented by the author at Vienna University of Technology

Dr. Gerhard Doblinger
Institute of Telecommunications
Vienna University of Technology
Gusshausstr. 25/389
A-1040 Wien

Phone 58801 38927, Fax 58801 938927
Email: Gerhard.Doblinger@tuwien.ac.at
Internet: www.nt.tuwien.ac.at/about-us/staff/gerhard-doblinger/

Preface

This collection of problems is compiled from the course book examples.[1] Starting with winter semester 2014, this book is freely available for students of my courses. Solutions of some of the more advanced problems are already available at the author's homepage

`www.nt.tuwien.ac.at/about-us/staff/gerhard-doblinger/`

(file `Mbook.tar.gz` or `Mbook.zip` in section "Teaching").

When working out solutions to the problems or projects, I recommend that you check your results with some theoretical investigation. Remember that computer simulations may show rounding errors, and inaccurate results. Examples are ill-conditioned systems of equations, local minima in case of optimization problems, and root finding of polynomials. In addition, due to rounding errors small imaginary parts may occur in cases where real-valued results are expected. Note that MATLAB® has its own philosophy to deal with NANs (see `help NaN`). As an example, you can use NANs in a vector to skip plotting of some data (e.g. to obtain a piecewise plot of a curve).

Although some of the problems can be solved with built-in MATLAB® functions or with toolbox functions, you should try to develop your own solutions. However, functions already available may be used to check your results.

In general, you should concentrate on implementing the signal processing methods and algorithms. Graphical interfaces (GUIs) and pompous graphical output are not required. However, you might do so if you are familiar with and enthusiastic about writing graphical interfaces.

As an alternative to solving a selected set of problems you can work out a project. A list of selected projects with brief descriptions is covered in this collection. The theoretical background of most of these projects will be discussed in the lecture part of this course. However, further reading is usually required to carry out a project.

I have included some additional course material in the appendix to offer you a written documentation of some topics I am presenting in the lecture part of the course.

Important notes to prepare your final report:

In order to get a grade, you must deliver a final report which contains at least an introduction, your solutions, MATLAB® code, and a discussion of results followed by a list of references.

It is legitimate to use MATLAB® code from examples found in the Internet, on my home page, and elsewhere. However, you must cite the sources (publication details, authors, addresses, emails, links, etc.). The same holds when using copy-and-paste to insert text material (e.g. from Wikipedia). In addition, copied text must be emphasized (e.g. using italic fonts) and put in quotation marks. At the end of the quoted text, the source must be cited (like [1]). Pictures, diagrams, and tables not created by yourself must also be referenced.

If you use MATLAB® code from other people as a starting point of your own program, or if you modify the original program, then you must include the original author(s) in

[1] G. Doblinger, "MATLAB-Programmierung in der digitalen Signalverarbeitung", J. Schlembach Fachverlag, Wilburgstetten 2001.

a comment at the beginning of your source code. I will check your solutions, text, and programs in regard to plagiarism. **If there is a high correlation to sources not referenced and quoted properly in your report and MATLAB® code, then I will not accept your work once and for all.**

Gerhard Doblinger Vienna, Sept. 2014

Contents

1 Introduction

In the following lecture notes, we do not present an introduction to basic digital signal processing concepts. We refer to course 389055 "Signals and Systems 2" where such an introduction is given. Course 389055 is based on my book "Zeidiskrete Signale und Systeme, 2. Auflage," J. Schlembach Fachverlag 2010. Alternatively, you may use the book by S. J. Orfanidis, "Introduction to signal processing," 2010, which is freely available at `www.ece.rutgers.edu/~orfanidi/i2sp`.

We focus on two main topics in signal processing based on the Fourier transform, and on the wavelet transform. These important topics are fundamental to classical and to modern signal processing methods, and the discrete-time versions (FFT, discrete-time wavelet transform) offer a huge variety of applications. An excellent introduction to modern signal processing methods can be found in the book of S. Mallat, "A wavelet tour of signal processing, the sparse way," Elsevier, 2009.

We start with a brief overview on the windowed Fourier transform (WFT), also called short-time Fourier transform. In contrast to the normal Fourier transform which offers a global frequency analysis, the WFT enables a localized time-frequency analysis. Such an analysis offers a more detailed look to signals with time-varying frequency content, and other non-stationary signal components. The WFT can efficiently be implemented by an FFT filter bank presented in some detail in the next section.

We will continue with a closer look to the wavelet transform (WT), starting with the continuous-time version (CWT). The CWT can give us a very detailed analysis to different types of natural data like audio signals, biomedical signals, and images. The WT also enables a compressive signal representation where only the dominant coefficients are saved, and weak or noise components are removed. Such signal approximations in combination with auditory or visual effects form the basis of modern data compression techniques. We finish this introduction with filter bank implementations of the discrete-time wavelet transform (DTWT).

After presenting the selected signal processing concepts, we summarize all the problems which can be worked out with MATLAB® in this course. As already mentioned in the Preface, a project can be selected instead of solving a set of problems. A more or less comprehensive list of projects with short overviews is included after the problem section.

Finally, some additional course material is collected in the Appendix. Most of these topics will be discussed during the lecture part, and they may be useful when working on the practical part of the course.

2 Windowed Fourier Transform

The Fourier transform of a continuous-time, finite-energy signal $x(t) \in \mathbf{L^2}(\mathbb{R})$

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\omega t}\, dt \tag{1}$$

measures the frequency content (spectrum) by integrating over the total signal support. Thus, only a global signal analysis is possible. Many natural signals like speech and audio

show time-varying spectra which cannot be observed in detail by such a global analysis. A localized analysis is possible by passing the signal through a time window to get a time-dependent spectrum:

$$X_{\mathrm{WFT}}(u,\xi) = \int_{-\infty}^{\infty} x(t)\, g(t-u)\, e^{-j\xi t}\, dt\,. \tag{2}$$

The **windowed Fourier transform (WFT)** (2) can be interpreted as cross-correlation of signal $x(t)$ with window function $g_{u,\xi}(t) = g(t-u)\, e^{-j\xi t}$ which resembles a modulated and time-shifted lowpass signal $g(t)$. The local spectrum is measured at time-shift u and modulation frequency ξ. The window length is given by the support of $g(t)$ and determines the time and frequency resolution of the WFT analysis. According to to uncertainty principle of the Fourier transform, short windows resolve fine details in the time domain accompanied by a poor frequency resolution. Large windows offer a high frequency resolution at the expense of a loss in time localization. Note that the time/frequency resolution is the same for all frequencies ξ and is determined by the window size only. In practice, we have to chose a compromise between time and frequency resolution fitted to signal properties. We will show this fact with several examples during the lecture.

As an example, the WFT of a sequence of modulated Gaussian signals is shown in Fig. 1 (small window size), and Fig. 2 (large window size), respectively.

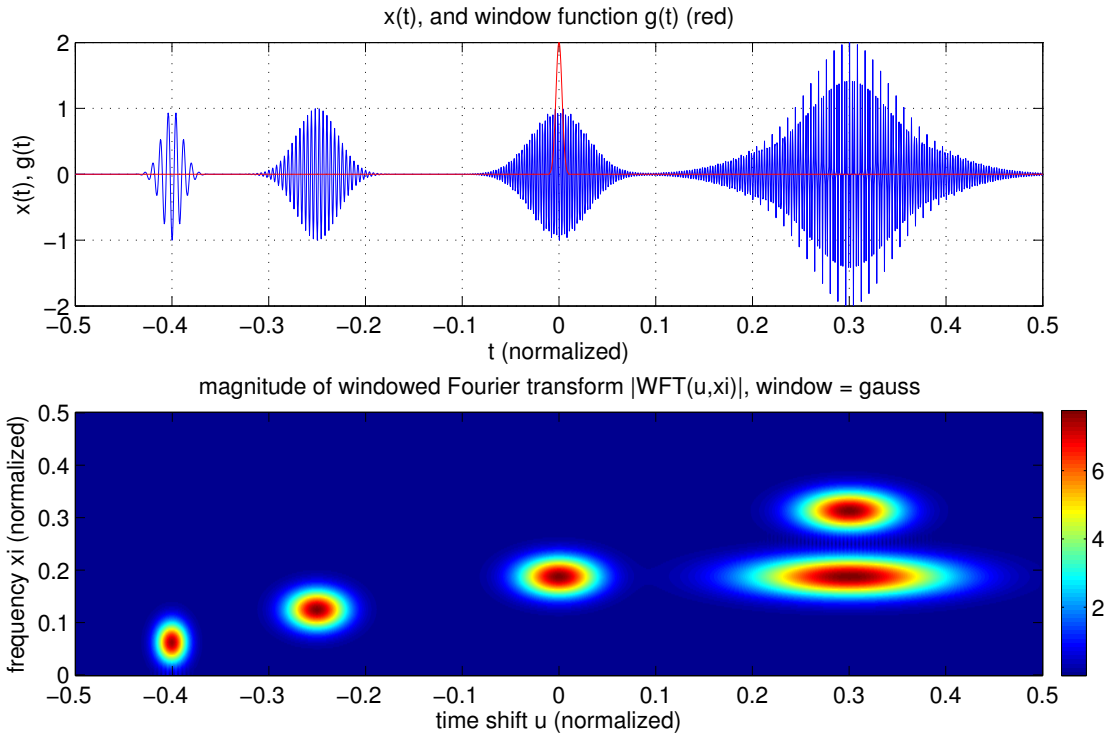


Figure 1: WFT of signal $x(t)$ consisting of modulated Gaussian impulses (small window $g(t)$ (red) shown in upper diagram).

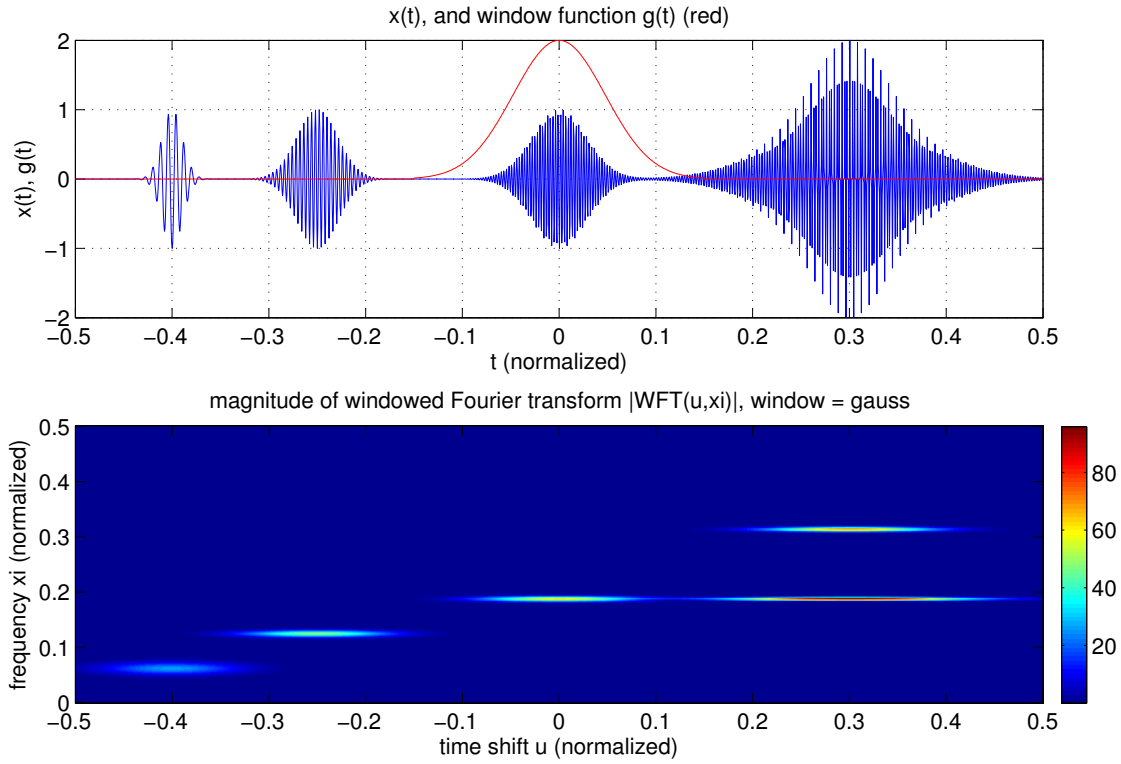


Figure 2: WFT of signal $x(t)$ consisting of modulated Gaussian impulses (large window $g(t)$ (red) shown in upper diagram).

The signal $x(t)$ can be recovered from the WFT by the **inverse windowed Fourier transform (IWFT)** defined by

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} X_{\mathrm{WFT}}(u,\xi)\, g(t-u)\, e^{j\xi t}\, d\xi du. \tag{3}$$

This reconstruction formula is valid if the Fourier transforms of $x(t)$ and $g(t)$ exist, and $g(t)$ has unit norm ($\|g\| = 1$).

To prove the IWFT (3), we follow the derivation in Mallat's book on page 96 by first writing (2) as a convolution:

$$X_{\mathrm{WFT}}(u,\xi) = e^{-j\xi u} \int_{-\infty}^{\infty} x(t)\, g(t-u)\, e^{-j\xi(t-u)}\, dt \;\; = e^{-j\xi u}(x * g_\xi)(u)\,, \tag{4}$$

with $g_\xi(t) = g(t)e^{j\xi u}$, and a symmetric window function $g(t) = g(-t)$. Next, we use Parseval's equation $\int_{-\infty}^{\infty} x_1(t)x_2^*(t)dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} X_1(j\omega)X_2^*(j\omega)d\omega$, and apply it to

$$\int_{-\infty}^{\infty} \underbrace{X_{\mathrm{WFT}}(u,\xi)}_{x_1(u)}\, \underbrace{g(t-u)\, e^{j\xi t}}_{x_2^*(u)}\, du = \frac{1}{2\pi} \int_{-\infty}^{\infty} \underbrace{X(j\omega+\xi)G(j\omega)}_{X_1(j\omega)}\, \underbrace{G^*(j\omega)e^{j\omega u}e^{j\xi t}}_{X_2^*(j\omega)}\, d\omega\,. \tag{5}$$

Interchanging the integrals in (3) and inserting (5) results in

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \underbrace{\frac{1}{2\pi} \int_{-\infty}^{\infty} X(j\omega + \xi)\, e^{j(\omega+\xi)t}\, d\xi}_{x(t)}\, |G(j\omega)|^2\, d\omega. \tag{6}$$

If $g(t)$ has unit norm $\|g\|^2 = \int_{-\infty}^{\infty} |g(t)|^2\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |G(j\omega)|^2\, d\omega$, then (3) is proved by (6). Note that interchanging the integrals is valid if the Fourier transforms of $x(t)$ and $g(t)$ exist. This means that $x(t)$ and $g(t)$ must be absolutely integrable which is a stronger requirement than the finite energy property $x(t), g(t) \in \mathbf{L^2}(\mathbb{R})$.

The reconstruction of the signal in Fig. 2 using the IWFT is shown in Fig. 3. We observe a perfect reconstruction of the original signal from the WFT coefficients. This result is obtained with a discrete WDT/IWDT presented in the next section.

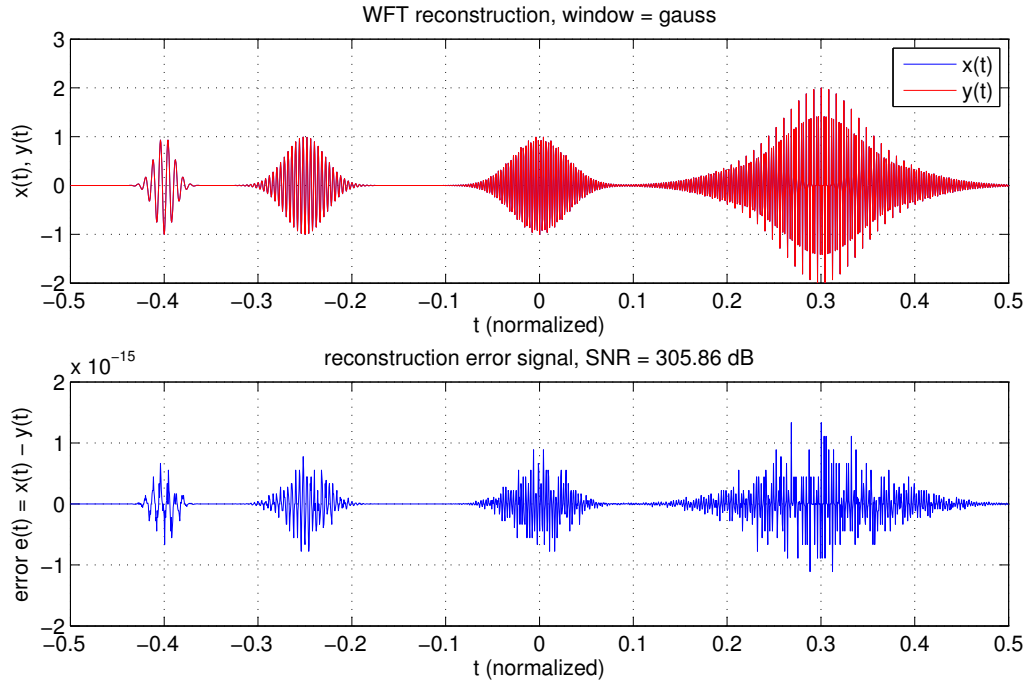


Figure 3: Reconstruction of signal $x(t)$ of Fig. 1 using the IWFT (error signal in lower diagram shows perfect reconstruction within machine precision).

In contrast to the inverse Fourier transform $x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(j\omega)\, e^{j\omega t}\, d\omega$, the IWFT (3) decomposes $x(t)$ by two-dimensional functions $X_{\mathrm{WFT}}(u, \xi)$. Such a signal representation is much more redundant than the one-dimensional inverse Fourier transform. Redundant signal decompositions, however, offer many important applications because we can selectively modify signal components in the time-frequency domain.

3 Discrete WFT and IWFT using an FFT filter bank

The **discrete windowed Fourier Transform (DWFT)** of a finite duration discrete-time signal $x[n], n \in [0, N-1]$ is equivalent to the discrete Fourier transform (DFT) if $x[n]$ is replaced by the windowed signal $x[n]g[n-m]$:

$$X_{\mathrm{DWFT}}[m,k] = \sum_{n=0}^{N-1} x[n]g[n-m]\, e^{-j\frac{2\pi}{N}nk}, \quad 0 \le m,k \le N-1. \tag{7}$$

The **inverse discrete windowed Fourier Transform (IDWFT)** to reconstruct $x[n]$ is obtained as a discrete-time and discrete-frequency version of (3):

$$x[n] = \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} X_{\mathrm{DWFT}}[m,k]g[n-m]\, e^{j\frac{2\pi}{N}nk}, \quad 0 \le n \le N-1. \tag{8}$$

Equation (8) can be proved in the same way as outlined in the proof of (3). To apply the inverse discrete Fourier transform (IDFT), we rewrite (8) as

$$x[n] = \sum_{m=0}^{N-1} g[n-m] \underbrace{\frac{1}{N}\sum_{k=0}^{N-1} X_{\mathrm{DWFT}}[m,k]\, e^{j\frac{2\pi}{N}nk}}_{\mathrm{IDFT}\{X_{\mathrm{DWFT}}[m,k]\}}, \quad 0 \le n \le N-1. \tag{9}$$

The following MATLAB® program segment computes DWFT and IDWFT. Note that we use an FFT length smaller than N since the window function $g[n]$ has length $N_w < N$.

```
N = 512;                              % signal length
x = cos(2*pi*16/N*(0:N-1));           % input signal
Nw = 128;                             % window length
tw = 2*pi*linspace(-0.5,0.5,Nw);      % window support is [-0.5,0.5]
Nf = Nw;                              % FFT length (Nf = Nw < N)
g = 0.5*(1 + cos(tw));                % window function g[n]
g = g/norm(g);                        % window must have ||g|| = 1
x1 = [zeros(1,Nw) x zeros(1,Nw)];     % add zeros to have enough samples
                                      % for x[n]g[n-m] at signal begin and end

% compute DWFT for time shift m and all Nf frequency points

DWFT = zeros(N+Nw,Nf);
for m = 1:N+Nw
   xt = x1(m:m+Nw-1);
   DWFT(m,:) = fft(xt.*g,Nf);
end

% compute IDWFT to reconstruct input signal

y = zeros(1,N+2*Nw);
```

```
for m = 1:N+Nw
   m1 = m:m+Nw-1;
   yt = ifft(DWFT(m,:),Nf);
   y(m1) = y(m1) + g.*yt(1:Nw);
end
y = real(y(Nw+1:N+Nw));              % save settled signal samples only
disp(max(abs(y-x)));                 % show maximum reconstruction error
```

This MATLAB® code is used in the author's functions `wftrans.m` and `iwftrans.m` to create the diagrams of Fig. 1, and Fig. 3, respectively.

The DWFT, and IDWFT are two-dimensional, redundant signal transformations. They require N^2 (or NN_w) coefficients to represent a signal of length N. In contrast, DFT, and IDFT need only N coefficients. The following FFT filter bank has a reduced redundancy because the DWFT coefficients are subsampled by shifting the window $g[n]$ by M samples instead of 1 sample. As a consequence, FFT, and IFFT are computed every M samples only. The window shift M is also called frame hop size.

The FFT filter bank can directly be derived from the DWFT/IDWFT using M samples window shifting. The analysis stage illustrated in Fig. 4 corresponds to the DWFT, and the synthesis stage (Fig. 5) is equivalent to the IDWFT.

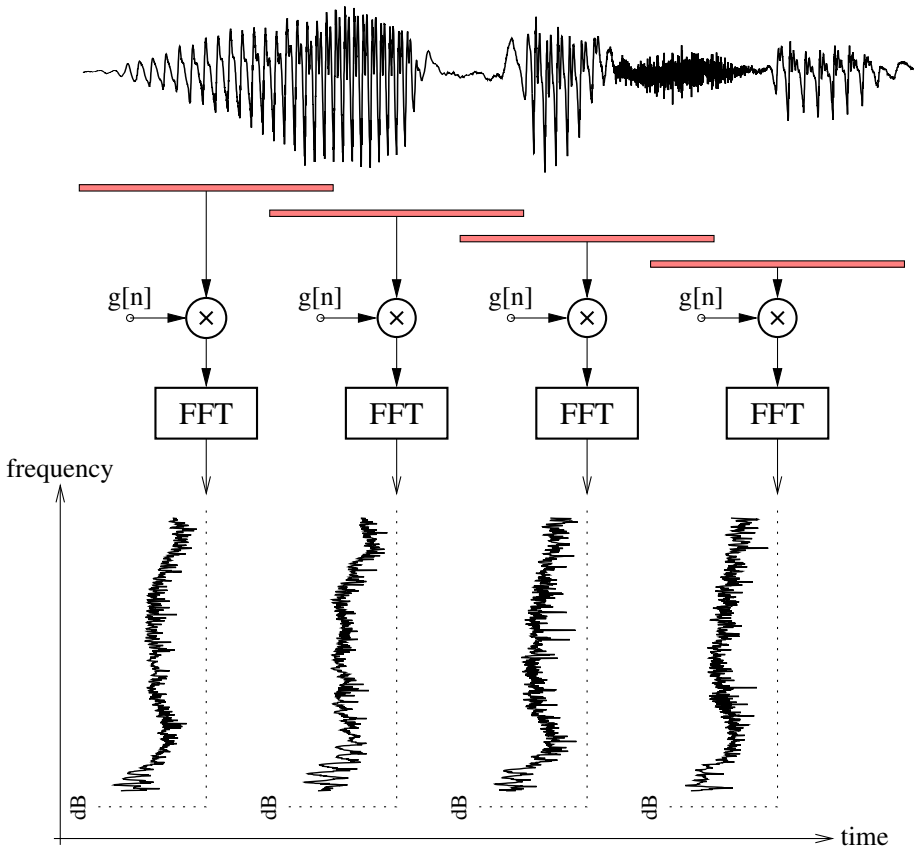


Figure 4: Analysis stage signal processing of the FFT filter bank (window shift $M > 1$)

In general, we cannot expect perfect signal reconstruction since we use a decimation of the DWFT coefficients. If the window shift M is a divisor of the window length N_w, then we will show that perfect reconstruction can be achieved. This is remarkable since a decimated FFT filter bank normally exhibits a near-perfect reconstruction property only.

In applications of the FFT filter bank, spectral components (also called subbands) at the output of the analysis stage (Fig. 4) are modified, e.g. to suppress unwanted components in the time-frequency plane. The modified spectra are then processed by the synthesis stage (Fig. 5) to obtain an enhanced output signal.

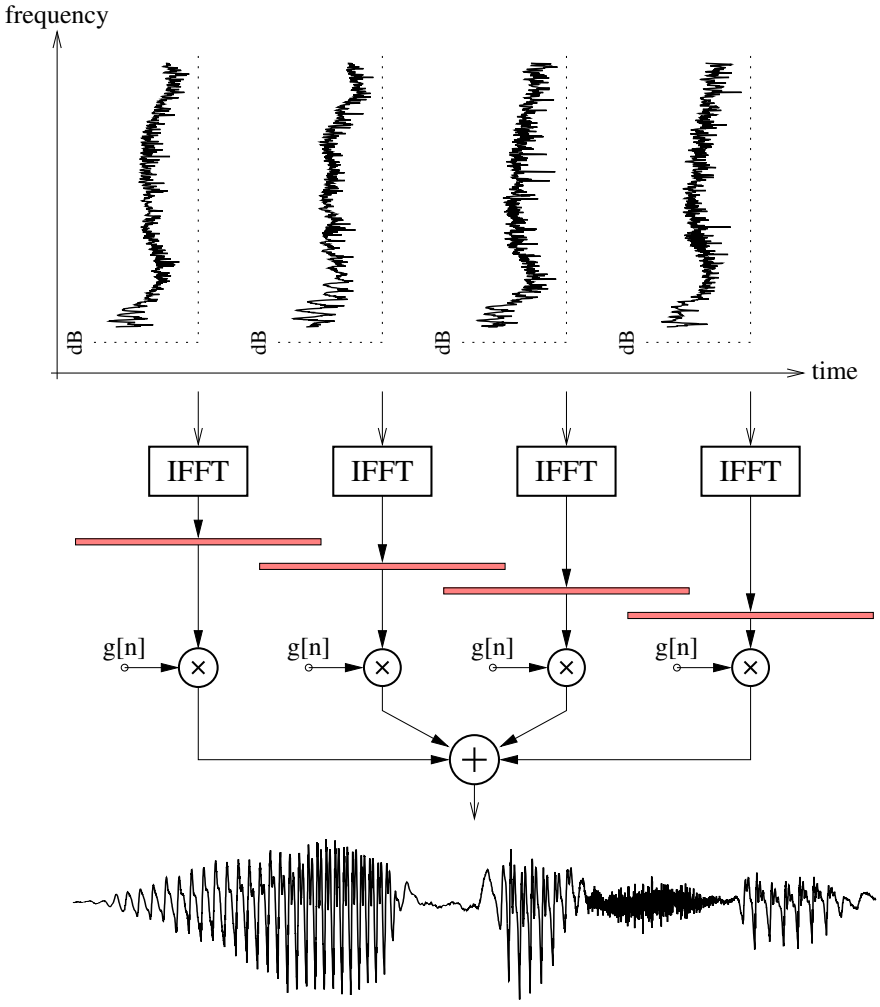


Figure 5: Synthesis stage signal processing of the FFT filter bank (window shift $M > 1$)

The decimated FFT filter bank operates computationally very efficiently since FFT and IFFT must be carried out every M input samples only. If the subbands are not modified, we can recover the original signal (except for a signal delay). In applications where only the spectral magnitudes are modified, and phases are left unchanged, the window function $g[n]$ at the synthesis stage in Fig. 5 can be omitted. This modification leads to the so-called **filter bank overlap addition method** investigated in some detail

next. The block diagram of the filter bank overlap addition method with frame length N_w (equal to FFT length) is shown in Fig. 6. At the analysis stage, a frame of N_w signal samples is stored in an input buffer and weighted by a sliding window which is advanced by $M < N_w$ samples to the next frame. The Fourier transform $X_m[k]$ of frame m is modified according to the specific application.

In the synthesis stage, N_w IFFT output samples are added to the previous frame stored in an output buffer to get the filter bank output signal. Input and output buffers operate at the sampling rate F_s of the input signal. However, FFT and IFFT processing is carried out only every M samples. As a consequence, FFT, spectral modification, and IFFT run on a reduced rate F_s/M. Typically, we use $N_w = 4096$, and $M = \frac{N_w}{4} = 1024$ at $F_s = 44.1$ kHz.

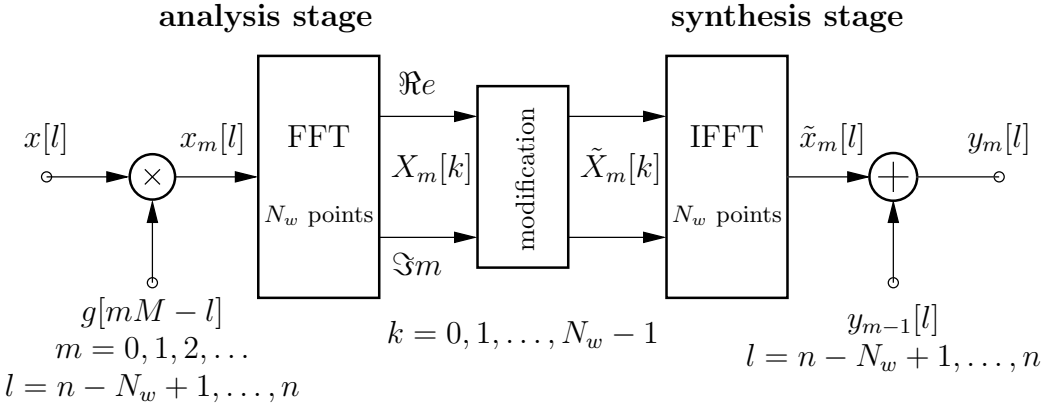


Figure 6: Filter bank overlap addition method (sampling index n, frame index m, buffer index l, frequency index k, only FFT/IFFT bin k is drawn, filter bank delay omitted)

The reconstruction property of the filter bank overlap addition method in Fig. 6 can be shown as follows. If we do not modify the spectral components $X_m[k]$, we notice that $\tilde{x}_m[l] = x_m[l]$. With

$$\tilde{x}_m[l] = x_m[l] = x[l]g[mM - l], \quad l = n - N_w + 1, \ldots, n - 1, n, \tag{10}$$

the overlap-add operation at the synthesis stage leads to

$$y_m[l] = y_{m-1}[l] + x[l]g[mM - l]. \tag{11}$$

Evaluation of recursion (11) starting at frame $m = 0$ with initial condition $y_{-1}[l] = 0$ results in

$$y_m[l] = x[l] \sum_{k=0}^{m} g[kM - l]. \tag{12}$$

Note that finite length window $g[l]$ is defined for $l \in [0, N_w - 1]$. Therefore, the filter bank output signal $y[n] = y_m[l = n]$ is given by

$$y[n] = x[n] \underbrace{\sum_{k=\lceil \frac{n}{M} \rceil}^{\lfloor \frac{N_w - 1 + n}{M} \rfloor} g[kM - n]}_{w[n]}, \quad n = 1, 2, \ldots \tag{13}$$

with floor function $\lfloor\cdot\rfloor$, and ceil function $\lceil\cdot\rceil$. Replacing n by $n+M$ in (13), we see that $w[n]$ is periodic with M. In addition, if M is a divisor of N_w ($N_w/M \in \mathbb{N}$), then (13) can be changed to

$$w[n] = \sum_{k=1}^{\frac{N_w}{M}} g[kM-n] = \sum_{k=0}^{\frac{N_w}{M}-1} g[kM+M-n], \quad n = 1, 2, \ldots, M \tag{14}$$

because $\lceil \frac{n}{M} \rceil = 1$, $\lfloor \frac{N_w-1+n}{M} \rfloor = \frac{N_w}{M}$ for $n \in [1, M]$. We get perfect reconstruction if $w[n] = 1, \forall n$. It will be shown that $w[n] = a\frac{N_w}{M}$ for the window functions given below. As an example, the reconstruction error for $M \in [1, \frac{N_w}{2}]$ and $N_w = 200$ is displayed in Fig. 7. For $M < \frac{N_w}{4}$, the observed error is in $[-10^{-3}, 10^{-3}]$, and is negligible for all practical applications. Perfect reconstruction is achieved if $N_w/M \in \mathbb{N}$ and by selecting

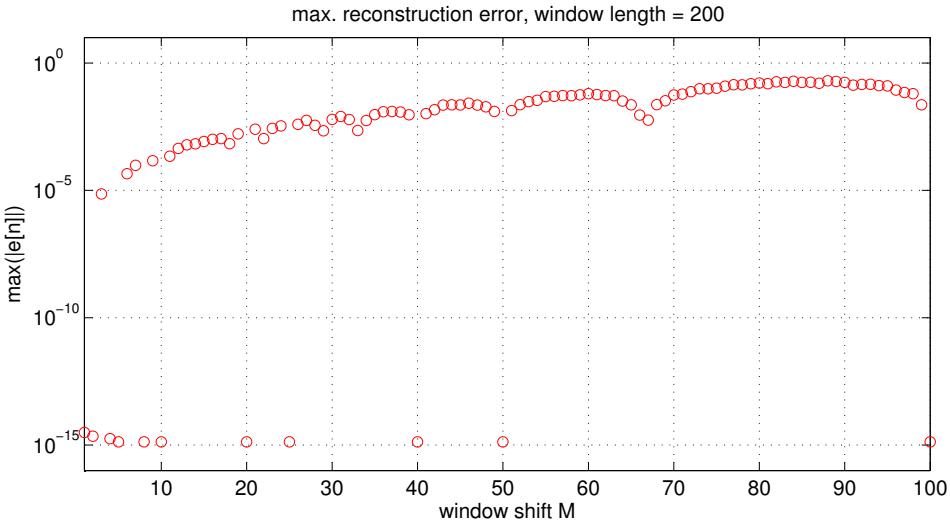


Figure 7: Overlap-add FFT filter bank reconstruction error $e[n] = y[n] - x[n]$ as a function of window shift M ($N_w = 200$, perfect reconstruction for $N_w/M \in \mathbb{N}$).

the cosine-type windows

$$\bar{g}[n] = \begin{cases} a - b\cos\left(\dfrac{\pi}{N_w}(2n+1)\right) & 0 \le n \le N_w - 1 \\ 0 & \text{else} \end{cases} \tag{15}$$

with $a = b = 0.5$, or $a = 0.54, b = 0.46$. Inserting $\bar{g}[n]$ of (15) as $g[n]$ in (14) and using the sum formula of a finite geometric series gives

$$w[n] = a\,\frac{N_w}{M} - b\,\Re e\left\{ e^{j\frac{\pi}{N_w}(2M-2n+1)} \underbrace{\sum_{k=0}^{\frac{N_w}{M}-1} e^{j\frac{\pi}{N_w}Mk}}_{0} \right\} = a\,\frac{N_w}{M}\,. \tag{16}$$

Thus, we have proved that the window function of (15) indeed ensures perfect reconstruction. According to (16), $\bar{g}[n]$ must be normalized:

$$g[n] = \frac{M}{aN_w}\,\bar{g}[n]\,. \tag{17}$$

It should be noted that MATLAB® functions `hanning()`, `hann()`, and `hamming()` use different window definitions, and do not deliver perfect reconstruction for $N/M \in \mathbb{N}$.

A MATLAB® code segment of the filter bank overlap addition method without spectral modification might look as follows:

```
N = 512;                    % signal length
Nw = 128;                   % window length
M = 32;                     % frame hop size (a divisor of Nw)
n = 0:N-1;
x = sin(2*pi*0.02*n);       % input signal
Nx = length(x);
g = 0.5*(1-cos(pi/Nw*(2*(0:Nw-1)+1)));  % time window function g[n]
g = M/(0.5*Nw)*g;           % normalize g[n]
y = zeros(1,N);             % output signal vector
Nf = Nw;                    % FFT length = Nw

for m = 1:M:N-Nw+1          % frame processing loop, M = frame hop size

   % analysis stage (apply weighting function and FFT to frame)

   m1 = m:m+Nw-1;           % indices of current frame
   X = fft(x(m1).*g,Nf);

   % synthesis stage (overlap-add of successive IFFT outputs)

   y(m1) = y(m1) + real(ifft(X,Nf));

end
plot(n,x,n,y), xlabel('n'), ylabel('x[n], y[n]'), grid on;
```

The perfect **reconstruction property of the decimated IDWFT** (Fig. 5) can be proved with the same steps as with the overlap-addition filter bank. Since we need a window function in the overlap-add operation (see (9)), requirement (14) is modified to

$$w[n] = \sum_{k=0}^{\frac{N_w}{M}-1} g^2[kM + M - n], \quad n = 1, 2, \ldots, M\,. \tag{18}$$

With the cosine-window (15), we get after some intermediate steps

$$w[n] = \left(a^2 + \frac{b^2}{2}\right)\frac{N_w}{M}. \tag{19}$$

Thus, the IDWFT needs a window normalization factor $\frac{\sqrt{M}}{\sqrt{N_w(a^2+b^2/2)}}$ for each of the two windows of the DWFT/IDWFT filter bank.

3.1 Pitch-scaling using the FFT filter bank

The FFT filter bank as presented in the previous section is an excellent digital signal processing system to efficiently manipulate signal spectra. One obvious application is spectral magnitude modification to equalize e.g. room acoustics or loudspeaker frequency responses. Spectral magnitude modification can also be used to suppress unwanted signals by attenuating noisy subbands. Such a signal enhancement or denoising normally requires a noise estimation subsystem, and an adaptive algorithm to control the spectral amplitudes.[2]

With pitch scaling (also called frequency scaling), we can e.g. shift the tuning of a musical instrument. Furthermore, the frequency spectrum of human voices can be scaled by modifying the harmonic frequencies. It should be noted, however, that pitch shifting of a male voice spectrum to a female one does not necessarily result in a pleasant voice since the formants (resonant frequencies) of the vocal tract are not changed in accordance to the dimensions of a female vocal tract. Pitch scaling is also used to compensate for the frequency alteration due to time-scaling (by playing sound records faster or slower).

From a signal processing point of view pitch scaling is carried out by multiplying all frequencies of the spectrum by a constant factor (typically between 0.5 and 2). When implementing pitch-scaling using the FFT, we must consider the discrete nature of the frequency spectrum. Therefore, scaling by a factor of e.g. 1.3 would result in frequency points not lying on the grid of the FFT used. In order to scale frequencies by arbitrary factors, we use the concept of the **instantaneous frequency** of bandpass signals.

The short-time spectrum $X_m[k]$ in Fig. 6 can be represented with magnitude and phase by

$$X_m[k] = |X_m[k]|\, e^{j\varphi_m[k]}, \tag{20}$$

with frame index m, frequency index k, and **instantaneous phase** $\varphi_m[k]$. There is nothing special with (20) since it simply represents a complex-valued quantity by magnitude and phase.

Frequency scaling requires a relationship between phase and frequency of the spectral components. If we assume that the phases of short-time stationary signals like speech do not change rapidly from frame to frame, then $\varphi_m[k]$ can be linearized with respect to frame index m:

$$\varphi_m[k] = \varphi_{m-1}[k] + \omega_m[k]M + 2\pi l \tag{21}$$

where $\omega_m[k]$ denotes the **instantaneous frequency**, and $2\pi l$ must be added because phase is not unique. How can we interpret the instantaneous frequency in case of the FFT filter bank? Remember that the input signal frames are multiplied by time window $g[n]$. Consequently, the signal spectrum is convolved by the Fourier transform of $g[n]$, and spectral lines are smeared around each FFT frequency point $\theta_k = 2\pi k/N_w$. The bandwidth of this spectral smearing is quite low due to the sharp spectrum of $g[n]$. Therefore, the instantaneous frequency $\omega_m[k]$ will be in close proximity to θ_k.

[2] More information about this application can be found in the course book.

With $\Delta\varphi_m[k] = \varphi_m[k] - \varphi_{m-1}[k]$ and (21), we get

$$|\Delta\varphi_m[k] - 2\pi l - \theta_k M| = \underbrace{|\omega_m[k] - \theta_k| M}_{<\,\pi}. \tag{22}$$

The right hand side of (22) is limited to π since we assume that

$$|\omega_m[k] - \theta_k| < \frac{\pi}{M}. \tag{23}$$

We can now remove the unknown multiple of 2π in (22) by means of a modulo-2π operation:

$$\Delta\bar{\varphi}_m[k] = (\Delta\varphi_m[k] - \theta_k M) \oplus 2\pi \quad \text{with} \quad |\Delta\bar{\varphi}_m[k]| < \pi. \tag{24}$$

The modulo operation in (24) differs from the common definition since it must maintain the sign of the dividend, and it must limit the dividend to $[-\pi, \pi]$ instead to $[-2\pi, 2\pi]$. Thus, you need to write a special MATLAB® function for this purpose.

The complete algorithm to compute the instantaneous frequencies is listed in Table 1.[3] Frequency index k runs up to $\frac{N}{2}$ since we assume real-valued input signals resulting in a Hermitian symmetric (conjugate symmetric) Fourier transform.

$$\begin{aligned}
\varphi_m[k] &= \arctan\left(\frac{\Im m\,\{X_m[k]\}}{\Re e\,\{X_m[k]\}}\right) \qquad k = 0, 1, \ldots, \frac{N}{2} \\
\Delta\varphi_m[k] &= \varphi_m[k] - \varphi_{m-1}[k] \\
\Delta\bar{\varphi}_m[k] &= (\Delta\varphi_m[k] - \theta_k M) \oplus 2\pi, \quad \text{so that } |\Delta\bar{\varphi}_m[k]| < \pi \\
\omega_m[k] &= \theta_k + \frac{\Delta\bar{\varphi}_m[k]}{M}, \qquad \text{with } \theta_k = \frac{2\pi k}{N}
\end{aligned}$$

Table 1: Algorithm to compute the instantaneous frequency $\omega_m[k]$ at each frequency point k, and for each frame index m

We can now easily apply frequency scaling with arbitrary scale factors s_c by changing the instantaneous frequency (which is not restricted to frequencies of the FFT grid). The new frequency indices to be used in the filter bank synthesis stage are computed by

$$k_s = \text{round}\left(\frac{k}{s_c}\right), \quad k = 0, 1, 2, \ldots, k_{max}, \tag{25}$$

with $k_{max} = \min\left(\frac{N}{2}, \left\lfloor s_c \frac{N}{2} \right\rfloor\right)$. The scaled instantaneous frequencies $\tilde{\omega}_m[k]$ are then obtained with k_s as follows:

$$\tilde{\omega}_m[k] = s_c\, \omega_m[k_s]. \tag{26}$$

The new instantaneous phases $\tilde{\varphi}_m[k]$ are found by an equation similar to (21):

$$\tilde{\varphi}_m[k] = (\tilde{\varphi}_{m-1}[k] + M\tilde{\omega}_m[k]) \oplus 2\pi \tag{27}$$

[3] In MATLAB®, we use function `angle()` or `atan2()` which place the angle in the correct quadrant.

where the modulo operation limits phase to $[-2\pi, 2\pi]$.[4]

The modified instantaneous phases lead to the modified spectral components at the synthesis stage of the filter bank in Fig. 6:

$$\tilde{X}_m[k] = |\tilde{X}_m[k]|\, e^{j\tilde{\varphi}_m[k]}. \tag{28}$$

(The modified magnitudes are obtained by $|\tilde{X}_m[k]| = |X_m[k_s]|$).

To gain more insight into the index manipulation, we illustrate the modification of frequencies by an example with $s_c = 3/4$, and 10 Hz FFT frequency grid spacing. For simplicity reasons, we assume that the instantaneous frequencies f_k of the input signal spectrum are lying on the FFT frequency grid. The modified instantaneous frequencies $\tilde{f}_k$ of the given input spectrum frequencies f_k are shown in the last row of Table 2. These frequencies are used to compute the modified instantaneous phases needed in (28).

f_k in Hz	10	20	30	40	50	60	70	...
k	1	2	3	4	5	6	7	...
$k_s = \text{round}\left(\frac{k}{s_c}\right)$	1	3	4	5	7	8	9	...
f_{k_s} in Hz	10	30	40	50	70	80	90	...
$\tilde{f}_k = s_c f_{k_s}$ in Hz	7.5	22.5	30	37.5	52.5	60	67.5	...

Table 2: Frequency scaling example with $s_c = 3/4$

As an example, we use Table 2 to get the modified spectral components of (28) for $k = 2$, and $k = 3$:

$$\tilde{X}_m[2] = |X_m(f = 30)|e^{j\tilde{\varphi}_m(f=22.5)}$$

$$\tilde{X}_m[3] = |X_m(f = 40)|e^{j\tilde{\varphi}_m(f=30)}.$$

The interpretation is as follows: At $k = 2$, we take the spectral magnitude at 30 Hz and combine it with the instantaneous phase at instantaneous frequency 22.5 Hz. By inspecting Table 2, we observe that some spectral components are omitted (e.g. at 20 Hz, or at 60 Hz). This may result in audible distortions, especially when the FFT lengths is to small (resulting in a coarse frequency grid). As an exercise, we recommend to repeat this example with $s_c > 1$.

An important final note must be made regarding a modification of the FFT filter bank overlap addition method. It is required that we perform the operlap-add at the synthesis stage using the window function $g[n]$ when adding the new frame (as shown in Fig. 5). As a result, audible artifacts due to the phase changes of the modified spectra are suppressed.

[4]MATLAB® function `rem()` must be used instead of function `mod()` to get the correct sign if the dividend is negative.

3.2 Time-scaling using the FFT filter bank

With time-scaling, we expand or compress the time axis resulting in a slow-down or speed-up of signal playback. Obviously, time-scaling cannot run in real-time. Only stored signals can be used when changing the time axis. A modification of the playback speed of a signal recording can by done by selecting a sampling frequency different from that of the recording. Changing the sampling rate, however, also changes the frequency axis resulting in unpleasant sounding voices. As a consequence, we must also scale the frequencies when using time-scaling.

Alteration of the playback speed with the FFT filter bank is carried out be selecting a different frame overlapping at the analysis, and at the synthesis stage (see Fig. 6). Thus, we shift the window function $g[n]$ using a frame hop size M_a, and we perform the overlap-add operation at the synthesis stage with frame hop size $M_s \neq M_a$. Typically, the time-scaling factor is chosen within $[0.5, 2]$. The frequency scaling needed to compensate for the unwanted frequency modification is done by the pitch-scaling algorithm presented in Section 3.1.

3.3 The FFT filter bank as a channel vocoder

A channel vocoder (**vo**ice **coder**) is a filter bank system to generate strange sounding voices like robot voices or other special effects. Basically, speech signals are synthesized by changing the excitation of a human vocal tract model. A common speech production model consists of a generator producing near-periodic signals in case of voiced sounds, and noise-like signals in case of unvoiced sounds. This signal is fed to a system of filters which model the influence of the vocal tract. In principle, these filters change the spectral envelope of the excitation signal spectrum to simulate e.g. the formants (resonances) of the vocal tract. As an example, we can use a square wave signal with $50 \ldots 100$ Hz fundamental frequency as excitation signal to transform the original speech to a robotic voice.

When using the FFT filter bank as a channel vocoder, we need two analysis stages: One stage to obtain the short-time spectrum of the original speech signal, and a second stage to compute the short-time spectrum of the excitation signal. The spectral envelope of the speech signal is multiplied by the excitation signal spectrum to yield the spectrum of the new speech signal. However, both envelope and excitation signal spectrum are multiplied by splitting the N_w FFT outputs (bins) in N_c subbands (called vocoder channels). Thus N_w/N_c FFT bins are processed per channel (N_w must be divisible by N_c). The envelope in each channel can be approximated by taking the magnitude and arithmetic mean of the FFT bins in the respective channel. This corresponds to rectifying and lowpass filtering of bandpass signals in an analog channel vocoder. The modified spectrum is transformed back to time domain by a single synthesis stage.

In MATLAB® , a basic channel vocoder can be implemented like the following code segment.[5] Spectral modification is efficiently carried out by using matrix reshaping as discussed in the introductory part of this course.

[5]This code segment is an improved version of MATLAB® program `chanvocoder.m` available at `http://sethares.engr.wisc.edu/vocoders/channelvocoder.html`.

```
x = x(:);                    % original speech signal
p = p(:);                    % new excitation signal
                             % Note: x, p must have same length
Nx = length(x);
g = hanning(N);              % N = window length
g = g(:);
y = zeros(Nx,1);
Nf = N;                      % FFT length = N
Nfh = Nf/2;                  % Note: we assume real-valued signals x, p
Nbins = Nfh/Nc;              % Note: Nfh must be a multiple of Nc
                             % Nc = number of vocoder channels in [0,Fs/2)
Xc = zeros(Nbins,Nc);
Pc = zeros(Nbins,Nc);
for m = 1:M:Nx-N+1

   % analysis filter banks

   m1 = m:m+N-1;
   X = fft(x(m1).*g,Nf);     % cut out a frame of length N,
                             % apply weighting function, and FFT
   P = fft(p(m1).*g,Nf);

   % modify spectrum of new excitation signal by spectral envelope of x
   % in each vocoder channel

   Xc(:) = X(1:Nfh);         % save Nbins FFT bins of each channel as
                             % columns of Xc
   Pc(:) = P(1:Nfh);
   env = mean(abs(Xc));      % full-wave rectifying and averaging
                             % to approximately obtain spectral envelope
                             % of signal x in each channel
   Pc = Pc.*env(ones(Nbins,1),:);   % apply envelope

   % overlap-add of successive IFFT outputs

   y1 = real(ifft(Pc(:),Nf));
   y(m1) = y(m1) + y1;

end
y = 0.9*y/max(abs(y));       % scale output to |y| < 1
```

You may want to experiment with different synthetic and natural excitation signals. One example is a square wave signal obtained with these MATLAB® instructions:

```
fsin = 100;                  % fundamental frequency of square wave in Hz
Fs = 16000:                  % sampling frequency in Hz
p = sign(sin(2*pi*fsin/Fs*(0:Nx-1)));
```

4 Wavelet transform

Signal analysis and synthesis with the FFT filter bank in Section 3 uses the same time-frequency resolution in each subband. Many natural signals, however, can be better analyzed with different time-frequency resolutions in subbands. As an example, speech signals require a high frequency resolution at lower frequencies in order to resolve pitch and formant frequencies. In the high frequency range, however, we need a high time resolution to detect transients, plosives, and short speech segments. Such a signal analysis can be obtained by decomposing signals with wavelets. Wavelet analysis and synthesis can be efficiently implemented with filter banks. Since fewer subbands as compared to the FFT filter bank are used, wavelet filter banks require a lower computational complexity. In this section, we give only a brief introduction to wavelets. A more detailed presentation can be found in the course book, and in Mallat's book[6]. Selected applications will be presented in the lecture part of the course, and can also be studied by some of the projects.

4.1 Continuous wavelet transform (CWT)

In order to show the principles of signal representations with wavelets, we consider time-continuous signals first. We obtain a local analysis of finite energy signals $x(t) \in \mathbf{L^2}(\mathbb{R})$ by a signal decomposition with a set of dilated and translated wavelets

$$\Psi_{u,s}(t) = \frac{1}{\sqrt{s}}\,\Psi\left(\frac{t-u}{s}\right), \quad u \in \mathbb{R}, s \in \mathbb{R}^+. \tag{29}$$

The wavelet $\Psi(t) \in \mathbf{L^2}(\mathbb{R})$ has zero average $\int_{-\infty}^{\infty} \Psi(t)\,dt = 0$, it is normalized ($\|\Psi\| = 1$), and centered around $t = 0$. At large scales s, the wavelets in (29) are expanded and cover a large support. On the other hand, small scales s yield compressed wavelets which are used to unveil finer signal details.

The **continuous wavelet transform CWT** at time shift u and scale s of $x(t)$ is

$$X_{\mathrm{CWT}}(u,s) = \int_{-\infty}^{\infty} x(t)\,\frac{1}{\sqrt{s}}\,\Psi^*\left(\frac{t-u}{s}\right)\,dt\,. \tag{30}$$

(* denotes conjugate complex). Equation (30) can be interpreted as cross-correlation operation on signal $x(t)$, and dilated and translated wavelets. It can be written as convolution

$$X_{\mathrm{CWT}}(u,s) = (f * \bar{\Psi}_s)(u) \tag{31}$$

with

$$\bar{\Psi}_s(t) = \frac{1}{\sqrt{s}}\,\Psi^*\left(\frac{-t}{s}\right)\,. \tag{32}$$

As an example, we show the CWT of a piecewise regular signal in Fig. 8.

[6]Stéphane Mallat, "A wavelet tour of signal processing, the sparse way, third edition," Elsevier Inc., 2009.

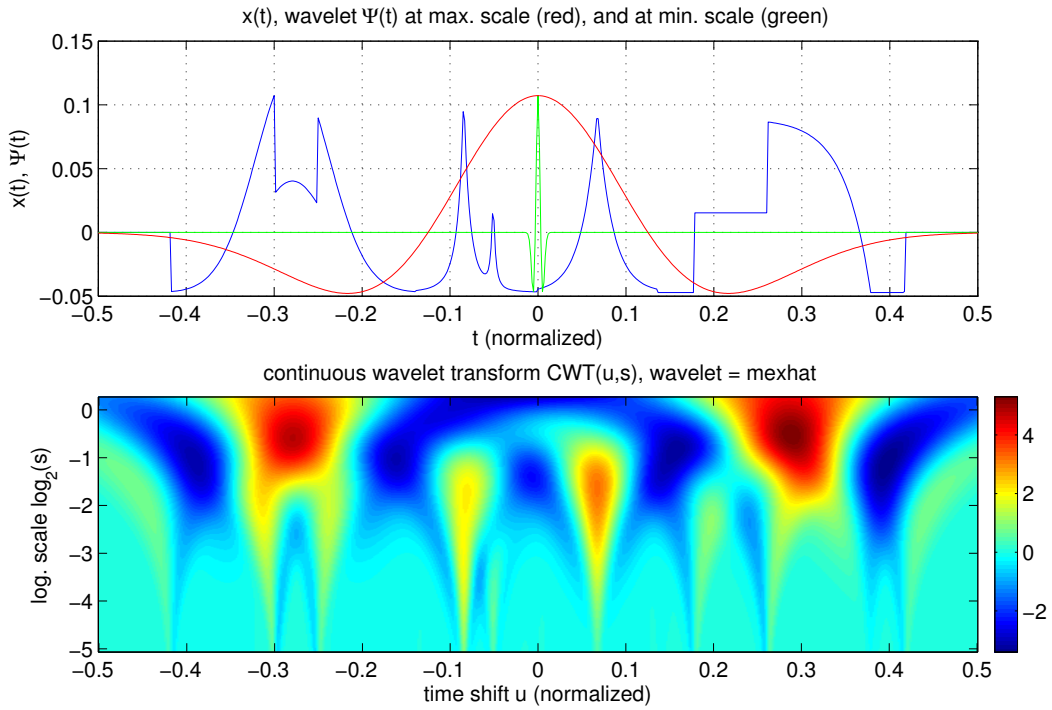


Figure 8: CWT of a piecewise regular signal (blue in upper diagram), logarithmic (base 2) scale axis with $s \in [0.03, 1.2]$. Upper diagram also shows wavelet Ψ_s at $s = 1.2$ (red), and at $s = 0.03$ (green).

The wavelet used in the CWT of Fig. 8 is the second derivative of a Gaussian function:

$$\Psi(t) = \frac{d^2}{dt^2}\, e^{-\frac{t^2}{2}} = \left(1 - t^2\right) e^{-\frac{t^2}{2}}\,. \tag{33}$$

($\Psi(t)$ not normalized to $\|\Psi\| = 1$). At large scales, the CWT shows a coarse signal representation dominated by the low-frequency signal components. The fine details (locations of the signal discontinuities) are clearly visible at small scales. In addition, most of the CWT-values are zero at the regular (smooth) signal segments. This property of the wavelet transform can be used to obtain a compressed signal representation where tiny CWT-values are set to zero. The CWT image in Fig. 8 can easily be interpreted as the result of correlations of signal $x(t)$ with wavelets at different scales. More examples of CWTs are demonstrated in the lecture part of the course using the author's MATLAB® functions `cwt_demo.m`, `cwavtrans.m`, and `icwavtrans.m`.

The finite energy signal $x(t)$ can be reconstructed from it's CWT by means of the **inverse continuous wavelet transform ICWT**

$$x(t) = \frac{1}{C_\Psi} \int_0^\infty \int_{-\infty}^\infty X_{\mathrm{CWT}}(u,s)\, \frac{1}{\sqrt{s}}\, \Psi\left(\frac{t-u}{s}\right) du\, \frac{ds}{s^2}\,, \tag{34}$$

provided that $C_\Psi = \int_0^\infty \frac{|\hat{\Psi}(\omega)|^2}{\omega}\, d\omega < \infty$, and real-valued wavelets are used ($\hat{\Psi}(\omega)$ is the Fourier transform of $\Psi(t)$). Note that $C_\Psi < \infty$ requires that $\Psi(t)$ has zero average as

stated at the beginning of this section. To prove the relationship of the ICWT, we follow the steps given in Mallat's book on page 105. First, we note that (34) can be rewritten by recognizing the second (inner) integral as a convolution operation:

$$x(t) = \frac{1}{C_\Psi} \int_0^\infty (X_{\mathrm{CWT}} * \Psi_s)(t) \, \frac{ds}{s^2} \tag{35}$$

with $\Psi_s(t) = \frac{1}{\sqrt{s}} \Psi\left(\frac{-t}{s}\right)$. The convolution in (35) is carried out with respect to time shift u. Taking the Fourier transform on both sides of (35), and the Fourier transforms applied to (31), and (32) we get (with substitution $\xi = s\omega$)

$$X(j\omega) = \frac{1}{C_\Psi} \int_0^\infty X(j\omega)\sqrt{s}\hat{\Psi}^*(s\omega)\sqrt{s}\hat{\Psi}(s\omega) \, \frac{ds}{s^2} = X(j\omega) \underbrace{\frac{1}{C_\Psi} \int_0^\infty \frac{|\hat{\Psi}(\xi)|^2}{\xi} \, d\xi}_{1} \, . \tag{36}$$

Thus, $C_\Psi = \int_0^\infty \frac{|\hat{\Psi}(\omega)|^2}{\omega} \, d\omega$ ensures that the Fourier transform of $x(t)$ is equal to the Fourier transform of the right hand side of (35) which in turn proves (35).

We illustrate the signal reconstruction with the ICWT in Fig. 9. The integral in (35) is approximated by a finite sum over a set of scales $s \in [s_{\min}, s_{\max}]$. The reconstruction is not perfect because we cannot extend the scale to $s_{\min} = 0$, and to $s_{\max} = \infty$.

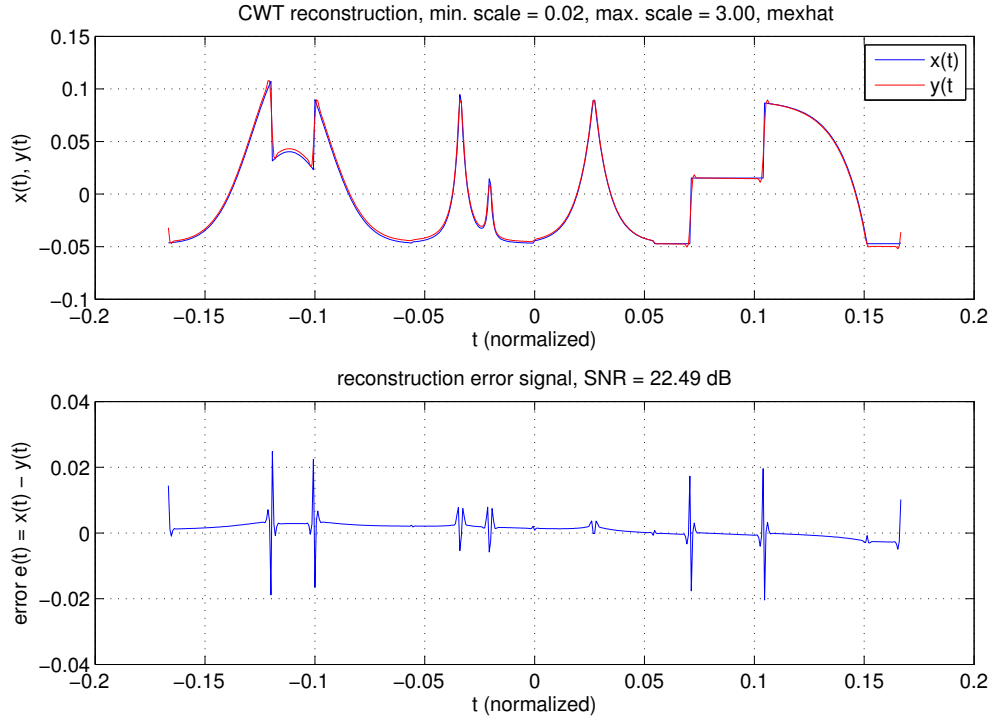


Figure 9: Reconstruction of the piecewise regular signal of Fig. 8 using an approximate ICWT with $s \in [0.02, 3]$. The reconstruction error signal is shown in the lower diagram.

Due to the limited range of scales, both the discontinuities and the low-frequency signal parts are not fully reconstructed. We will see that the **discrete-time wavelet transform DTWT** enables a perfect reconstruction of discrete-time signals. In addition, the DTWT is computationally much more efficient than a discrete approximation of (35). As far as signal analysis is concerned, however, the CWT gives a much finer image of signal details. Thus, the main applications of the CWT are analysis of signals with time-varying components, and detection of signal irregularities like discontinuities.

4.2 Discrete wavelet transform (DWT)

Before introducing the DTWT, we briefly discuss the discrete wavelet transform (DWT) which is still applied to time-continuous signals $x(t)$. In addition, the wavelets $\Psi(t)$ are again time-continuous functions. Time shift u and scale s, however, are discrete variables, and will be represented by indices n and k, respectively.

The (DWT) can be obtained by a filter bank framework derived from the CWT. As opposed to the CWT, a discrete set of scales $s = a^{-k}, a > 1, k \in \mathbb{N}$ is used. In addition, the continuous-time filter bank signals are sampled in accordance to their bandwidths.

Filter banks with uniform subbands consist of a bandpass filter set with equal bandwidths. These bandpass filters can be obtained by modulation of a single lowpass filter with equi-distant modulation frequencies. On the contrary, filter banks of the DWT are created by frequency-scaling (and not by shifting) of a prototype filter transfer function as follows:

$$H_k(j\omega) = a^{\frac{k}{2}} H\left(ja^k\omega\right) \quad \Leftrightarrow \quad h_k(t) = a^{-\frac{k}{2}} h\left(a^{-k}t\right), \quad a > 1,\ k \in \mathbb{N}. \tag{37}$$

$H(j\omega)$ is the prototype filter transfer function, and $H_k(j\omega)$ are the subband filter transfer functions. In time domain, we get time-scaled impulse responses $h_k(t)$. Factor $a^{-\frac{k}{2}}$ is needed to obtain an equal signal energy $\int_{-\infty}^{\infty} |h_k(t)|^2 dt$ for all subbands k.

The result of frequency-scaling applied to a wavelet prototype filter is illustrated in Fig. 10 for scaling parameter $a = 2$. The prototype filter $H_0(j\omega) = H(j\omega)$ shows the

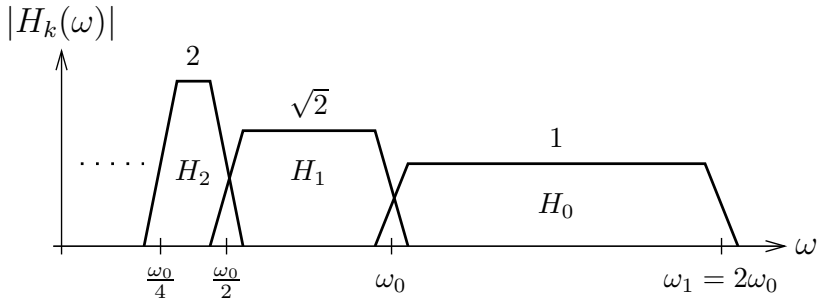


Figure 10: Schematic subband transfer functions of a wavelet analysis filter bank (prototype filter H_0, scaling factor $a = 2$)

largest bandwidth with cutoff frequencies ω_0, and $\omega_1 = 2\omega_0$. A dyadic set of bandwidths similar to an octave band analysis in acoustics is obtained with $a = 2$.

According to the sampling theorem of bandpass signals, wavelet subband signals can be sampled with smaller sampling frequencies in the low frequency range, and with higher sampling frequencies in the upper frequency bands. A block diagram of this sampling process is shown in Fig. 11. We get a set of discrete-time subband signals which as a whole form the **discrete wavelet transform (DWT)**.

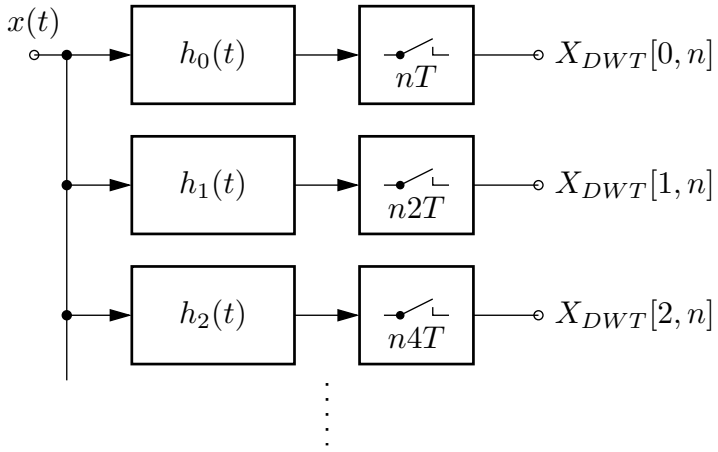


Figure 11: DWT analysis filter bank (scaling factor $a = 2$, subband filter impulse responses $h_k(t)$, sampling interval T)

With Fig. 11, the DWT is obtained by convolving signal $x(t)$ with subband filters $h_k(t)$, followed by sampling of the filter output signals at time points $n2^kT$:

$$\begin{aligned} X_{\mathrm{DWT}}[k,n] &= \int_{-\infty}^{\infty} x(\tau) h_k\left(n2^kT - \tau\right) d\tau \\ &= 2^{-\frac{k}{2}} \int_{-\infty}^{\infty} x(\tau) h\left(nT - 2^{-k}\tau\right) d\tau, \end{aligned} \tag{38}$$

with $k \in \mathbb{N}$, $n \in \mathbb{Z}$. $X_{\mathrm{DWT}}[k,n]$ of (38) assigns a two-dimensional, discrete decomposition to one-dimensional signal $x(t)$. We can get a significant data compression, if the signal can be decomposed with a few coefficients $X_{\mathrm{DWT}}[k,n]$ only.

The **inverse DWT (IDWT)** reconstructs $x(t)$ as follows:

$$x(t) = \sum_k \sum_n X_{\mathrm{DWT}}[k,n]\, \psi_{kn}(t). \tag{39}$$

The two-dimensional signal representation (39) is much more general than e.g. a Fourier series expansion which is a single sum with periodic basis functions. Various basis functions of the IDWT exist which can be adapted to a great variety of signals. An important class of basis functions $\psi_{kn}(t)$ are orthonormal wavelets with property

$$\int_{-\infty}^{\infty} \psi_{kn}^*(\tau)\psi_{lm}(\tau)\, d\tau = \delta[k-l]\delta[n-m] \tag{40}$$

($\delta[n]$ is the unit impulse). Integrating both sides of (39) and inserting (40) in the result gives

$$X_{\mathrm{DWT}}[k,n] = \int_{-\infty}^{\infty} x(\tau)\psi_{kn}^*(\tau)\,d\tau. \tag{41}$$

Comparison of (41) and (38) yields

$$\psi_{kn}(t) = h_k^*\left(n2^kT - t\right) = 2^{-\frac{k}{2}}\,h^*\left(nT - 2^{-k}t\right). \tag{42}$$

Thus, orthonormal wavelets can be derived by scaling and shifting of the prototype filter impulse response $h(t)$. The signal expansion with the DWT is then given by

$$\begin{aligned} x(t) &= \sum_k \sum_n X_{\mathrm{DWT}}[k,n]\,h_k^*\left(n2^kT - t\right) \\ &= 2^{-\frac{k}{2}} \sum_k \sum_n X_{\mathrm{DWT}}[k,n]\,h^*\left(nT - 2^{-k}t\right). \end{aligned} \tag{43}$$

A single filter impulse response $h(t)$ is sufficient to design a complete orthonormal wavelet analysis/synthesis filter bank.

4.3 Discrete-time wavelet transform (DTWT)

The DWT as presented in the previous section is applied to continuous-time signals. Discrete-time signals need discrete-time versions of wavelets which lead to the DTWT. The dyadic frequency partitioning of the DTWT is sketched in Fig. 12 with 4 subbands up to half the sampling frequency $f_N = f_s/2$ (the different gains in each subband are omitted). We will show that the DTWTs of length N signals require a computational complexity of $\mathcal{O}(N)$ approaching the smallest count of arithmetic operations needed to process N samples. An FFT spectral analysis typically needs $\mathcal{O}(N \log_2 N)$ arithmetic operations.

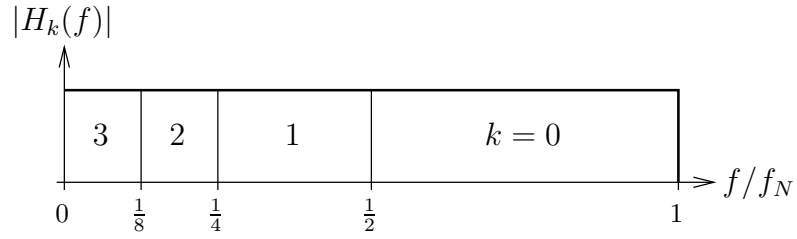


Figure 12: Schematic frequency partitioning of the DTWT with $M = 4$ subbands ($f_N = f_s/2$, with sampling frequency f_s)

The dyadic subband scheme can be easily obtained with a tree-like structure of two-channel filter banks. As shown in Fig. 13, the two-channel filter bank splits the input spectrum in a lowpass, and in a highpass part. After subsampling by a factor 2, the output spectra are stretching from 0 up to $f_N' = f_s'/2 = f_s/4$. The DTWT is obtained by combining two-channel filter banks to the tree structure in Fig. 14. The block diagram in

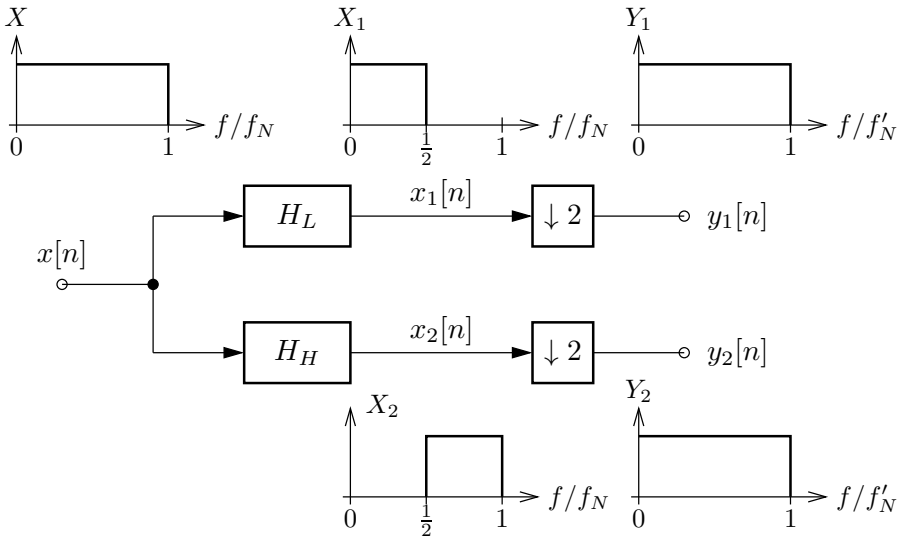


Figure 13: Schematic spectra of a two-channel filter bank with subsampling of subband signals ($f_N = f_s/2$, $f'_N = f'_s/2 = f_s/4$)

Fig. 14 corresponds to the DWT block diagram in Fig. 11 where subsampled subbands are used too. The sampling rate reduction in subbands ensures that the total number of output samples in Fig. 14 is not changed (besides extra samples due to filter operations). As a consequence, the DTWT has approximately the same number of samples as the input signal. This is similar to the FFT. However, less arithmetic operations are needed when using efficient realizations of two-channel filter banks.

Inspecting Fig. 14 and taking into account that subsampling by 2 takes every other signal sample, we can derive the following DTWT equations (compare with (38)):

$$\begin{aligned} X_{\mathrm{DTWT}}[k,n] &= \sum_{m=-\infty}^{\infty} x[m]\, h_k\left[2^{k+1}n - m\right], \quad k = 0, 1, \ldots, M-2 \\ X_{\mathrm{DTWT}}[M-1,n] &= \sum_{m=-\infty}^{\infty} x[m]\, h_k\left[2^{M-1}n - m\right] \end{aligned} \tag{44}$$

where M is the number of subbands. In principle, we can use (44) to compute the DTWT. However, the highly modular tree structure in Fig. 14 is much more efficient. In addition, we only need to store a single set of filter coefficients.

According to Fig. 14, and (44), respectively, the DTWT can be interpreted as a set of maximally decimated nonuniform subband signals. In addition to subsampling, the DTWT subbands normally show a large overlapping. Thus, the DTWT is not a bandpass filter bank with highly selective and disjunctive filters. Nevertheless, the input signal of the DTWT can be perfectly reconstructed by means of the inverse DTWT (IDTWT), also called DTWT synthesis.

Before discussing the IDTWT, we illustrate the filter bank behavior of the DTWT by an example. The DTWT tree structure in Fig. 14 can be remodeled to a common

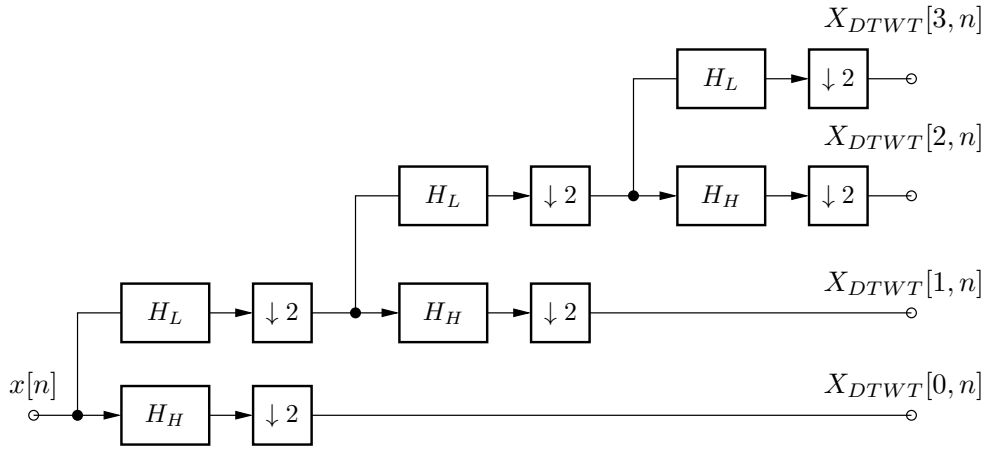


Figure 14: $M = 4$ DTWT as a tree of subsampled two-channel filter banks (analysis stage)

filter bank structure by combining the filters and subsampling operations of each branch to a single filter, and a single subsampling block. The total subsampling is obtained by shifting all factor 2 subsampling operations to the individual DTWT outputs. We can use a property of multirate systems, where subsampling by factor M at the input of a filter with transfer function $H(z)$ can be shifted to the filter output, if we replace $H(z)$ by $H(z^M)$.[7] With this replacement, we obtain the equivalent filter bank structure shown in Fig. 15.

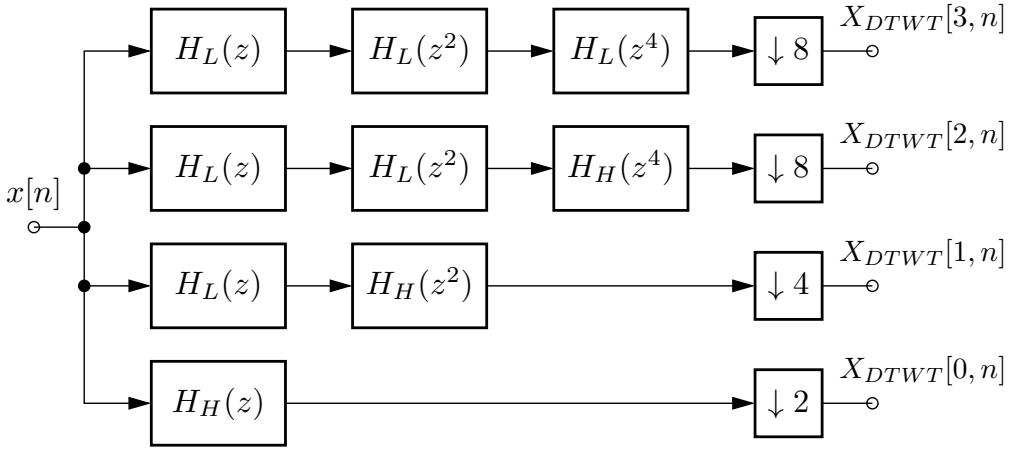


Figure 15: Subband filters of the $M = 4$ DTWT filter bank, derived from the DTWT tree structure in Fig. 14

An example of subband filter transfer functions without subsampling at filter bank outputs is plotted in Fig. 16. The subband filters are derived from two-channel FIR filters

[7] More details can be found in the course book.

of length $L = 30$. This figure shows that the design of wavelet filters is not dominated by the goal of finding the best selective bandpass filters. Typically, wavelet filters are designed to achieve a good signal compression by a DTWT with concentrated coefficients.

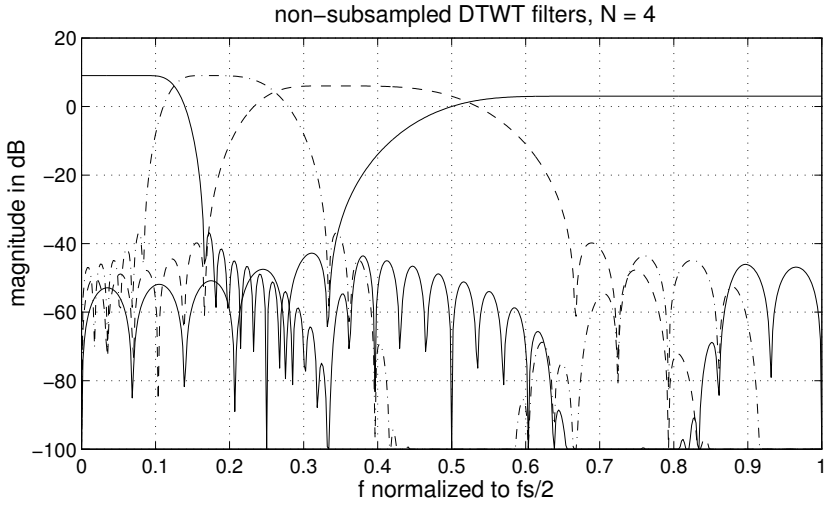


Figure 16: $M = 4$ DTWT filter bank transfer functions (without subsampling), derived from $H_L(z)$, and $H_H(z)$ FIR filters of length $L = 30$

Signal reconstruction from a DTWT is done by the **inverse DTWT (IDTWT)** which can be implemented as a tree of two-channel filter banks too (see Fig. 17). The reconstruction is perfect provided that the two-channel filter banks exhibit the perfect-reconstruction property.

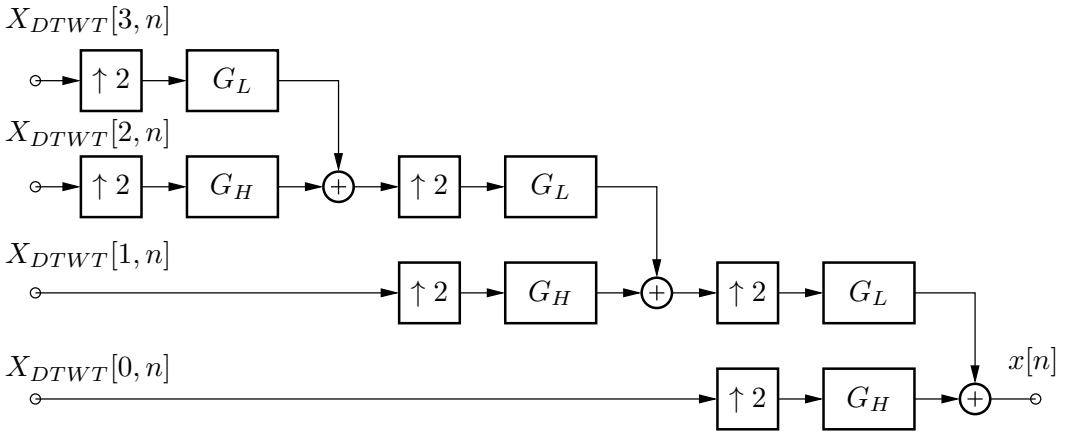


Figure 17: Tree structure of the $M = 4$ IDTWT with upsampled two-channel filter banks

We can verify the reconstruction property of the DTWT+IDTWT combination by replacing each two-channel analysis/synthesis filter bank from top to bottom by a direct link (feed through). From a practical point of view, however, we must take care of the

different signal delays (caused by the filter delays) in each DTWT subband. Additional delays are thus needed to obtain the same delay for all subband signals.

As an example of wavelet analysis with the DTWT tree, we show the magnitudes of the DTWT coefficients in Fig. 18 using a piecewise regular signal. Comparing the DTWT

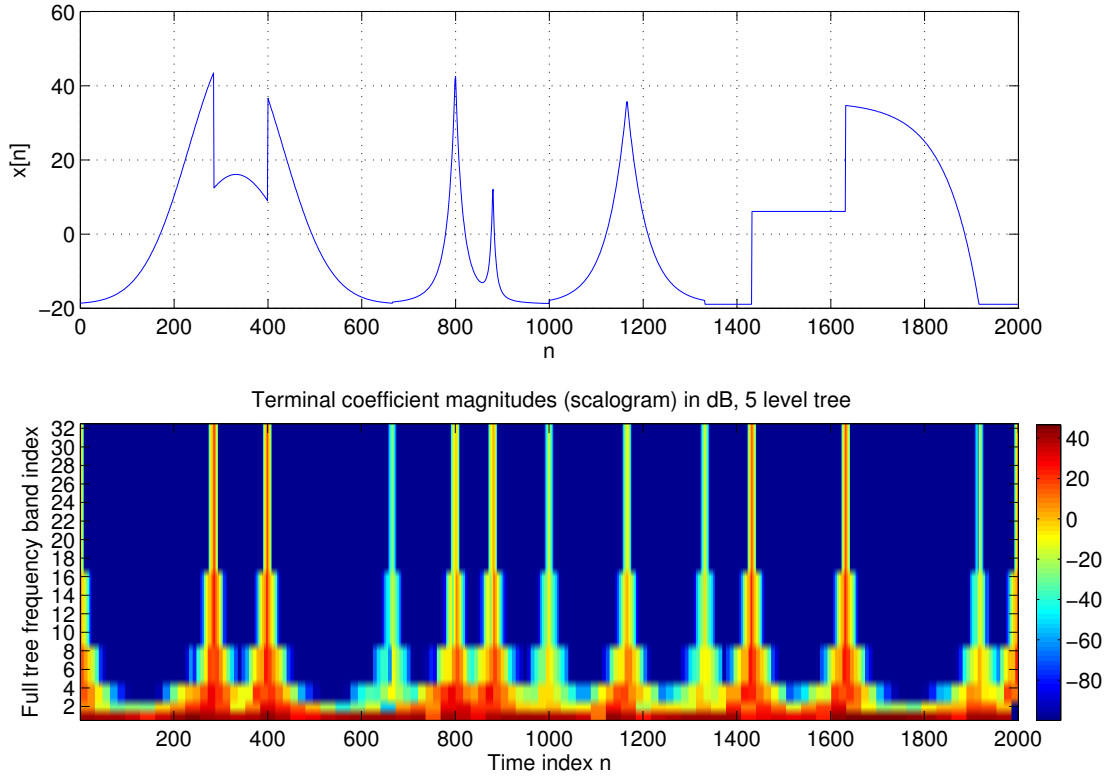


Figure 18: DTWT $|X_{\mathrm{DTWT}}[k,n]|$ of a piecewise regular signal (magnitude in dB, $M = 5$ subbands (tree levels).

with the CWT of Fig. 8, we observe a similar image. However, the CWT looks much sharper since it uses much more coefficients. The $M = 5$ subbands of the DTWT give rise to blocky picture details of Fig. 18. However, the signal discontinuities are clearly resolved, and most of the coefficients are zero in the regular signal segments.

The signal reconstruction property of the IDTWT tree is illustrated in Fig. 19 in case of an $M = 5$ level tree. The filter H_L, and H_H of the wavelet tree have length $L = 20$. In contrast to the ICWT of Fig. 9, we get a perfect signal reconstruction. In addition, the DTWT/IDTWT signal analysis/synthesis requires by far less coefficients, and computational complexity. We will discuss even more efficient realizations in the next two sections.

Another advantage of the modular structure of the DTWT (and IDTWT) is the reduction of the filter bank design to the design of a two-channel analysis/synthesis filter bank.

There are two MATLAB® programs available in archive `mbook.zip` at the author's home page: Program `qmf_2()` can be used to design filter banks with linear phase FIR filters; program `qmf_2hb()` is provided to design orthonormal wavelet filters. Filters based on common wavelets can be designed with function `wfilters.m` of the MATLAB® Wavelet Toolbox.

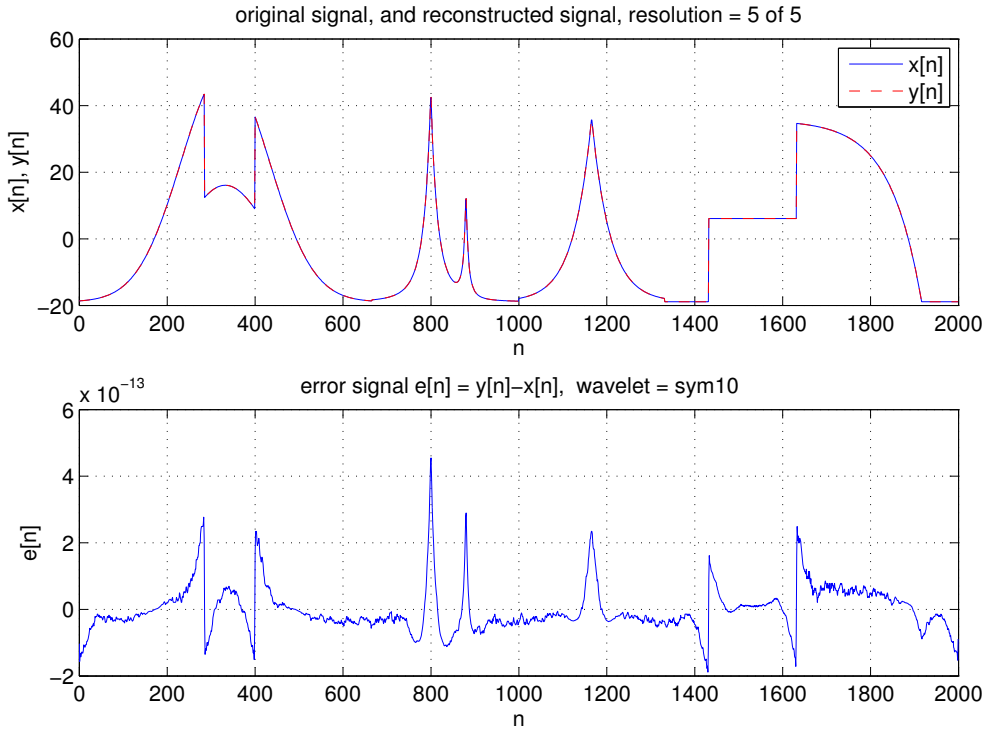


Figure 19: Reconstruction of the piecewise regular signal of Fig. 18 using the IDTWT with $M = 5$ subbands (error signal in lower diagram shows perfect reconstruction within machine precision).

4.4 DTWT/IDTWT realization with polyphase decomposition

We first discuss an efficient **DTWT/IDTWT realization using linear-phase FIR filters** which is based on the **polyphase decomposition**. The transfer function $H(z)$ of a digital filter with impulse response $h[n]$ can be split in two polyphase components as follows:

$$\begin{aligned} H(z) &= \sum_{n=-\infty}^{\infty} h[n] z^{-n} = \sum_{n=-\infty}^{\infty} \underbrace{h[2n]}_{h_g[n]} z^{-2n} + \sum_{n=-\infty}^{\infty} \underbrace{h[2n+1]}_{h_u[n]} z^{-2n} z^{-1} \\ &= H_g\left(z^2\right) + z^{-1} H_u\left(z^2\right). \end{aligned} \tag{45}$$

The two-channel filter bank with lowpass filter H_L and highpass filter H_H can be simplified if the impulse responses of these filters obey the relationship $h_H[n] = (-1)^n h_L[n]$. Note

that this relationship does not hold for all filter bank designs. It is fulfilled when using program `qmf_2()` but not for a design with program `qmf_2hb()`.

The polyphase decomposition (45) combined with $h_H[n] = (-1)^n h_L[n]$ leads to

$$\begin{aligned} H_H(z) &= \sum_{n=-\infty}^{\infty} (-1)^n h[n] z^{-n} \\ &= \sum_{n=-\infty}^{\infty} (-1)^{2n} h[2n] z^{-2n} + \sum_{n=-\infty}^{\infty} (-1)^{2n+1} h[2n+1] z^{-2n} z^{-1} \\ &= H_g\left(z^2\right) - z^{-1} H_u\left(z^2\right). \end{aligned} \tag{46}$$

Thus, we get the block diagram of the lowpass/highpass filters in Fig. 20.

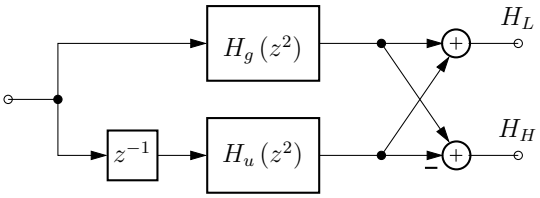


Figure 20: Polyphase decomposition of lowpass/highpass filters (H_L, H_H) of a two-channel filter bank (analysis stage without subsampling)

Only two polyphase filters are needed in Fig. 20 to obtain both subband filters (H_L, H_H). An additional advantage is possible when shifting the subsampling operation of the DTWT, and the upsampling operation of the IDTWT to input, and output, respectively (see Fig. 21). The $N = 2$ DTWT/IDTWT implementation in Fig. 21 is highly efficient

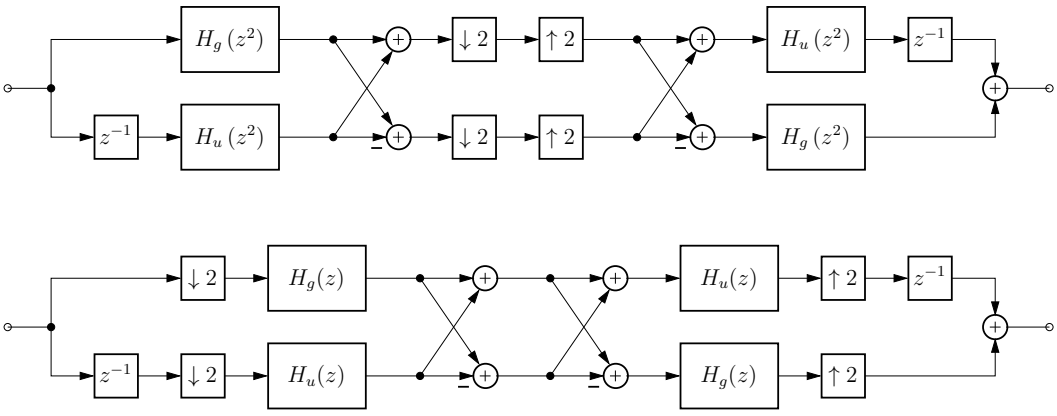


Figure 21: $N = 2$ DTWT + IDTWT polyphase filter bank with subsampling of subband signals (below: shifting of sampling rate conversion to input and output)

since all arithmetic operations run at the lower sampling rate. A mild disadvantage is that a perfect reconstruction of the input signal is not possible with linear phase filters H_L,

H_H. However, the reconstruction error is in the range $10^{-5} \ldots 10^{-3}$ which is acceptable in most applications.

A MATLAB® implementation of Fig. 21 is straight forward:

```
hg = h(1:2:end);              % polyphase filters of low pass prototyp
                              % designed e.g. with h = qmf_2(50,0.65,0.1);
hu = h(2:2:end);
x = x(:);                     % input signal

% analysis filter bank

u1 = filter(hg,1,x(1:2:end));
u2 = filter(hu,1,[0 ; x(2:2:end-1)]);
y1 = u1 + u2;                 % subband signals y1, y2
y2 = u1 - u2;

% synthesis filter bank

v1 = y1 - y2;
v2 = y1 + y2;
y = zeros(length(v1)+length(v2),1);
y(1:2:end) = filter(hg,1,v1);
y(2:2:end) = filter(hu,1,v2);
```

The two-channel polyphase analysis/synthesis filter banks in Fig. 14 (Fig. 17) can be used in the DTWT/IDTWT tree structure. Since the individual subbands signals have different delays, we must compensate the delays in the synthesis stage. For this reason, the following MATLAB® code is a little bit tricky.

```
function y = DTWT_IDTWT(N,h,x)
% function y = DTWT_IDTWT(N,h,x)
% compute combination of DTWT and IDWT to show reconstruction property
%
% N    number of DTWT coefficients = number of filter bank channels
% h    prototype low pass filter impulse response (linear phase filter)
% x,y  input, and output signal vectors
%
% G. Doblinger, TU-Wien, 02-2001, 09-2012

Nx = length(x);
L = length(h);
if Nx < 2^N + L
   error('length of x must be greater than 2^N + length(h)');
end

p1 = h(1:2:end);            % polyphase filters of low pass prototyp
p2 = h(2:2:end);
```

```
DTWT = zeros(Nx+L,N);       % matrix of N DTWT coefficients
yl = x(:);

% compute DTWT (anaylsis filter bank)

for n = 1:N-1
   [yl,yh] = polyafib2(p1,p2,yl);
   DTWT(1:length(yh),n) = yh;
end
DTWT(1:length(yl),N) = yl;

% compute IDTWT = output y of synthesis filter bank

y = DTWT(1:length(yl),N);
for n = N-1:-1:1
   y = polysfib2(p1,p2,y,DTWT(1:length(y),n));
end
y = y(1:Nx);

% ------------------------------------------------------

function [yl,yh] = polyafib2(p1,p2,x)

% compute analysis filter bank (polyphase filters)

u1 = conv(p1,x(1:2:end));
u2 = conv(p2,[0 ; x(2:2:end-1)]);
yl = u1 + u2;
yh = u1 - u2;

% ------------------------------------------------------

function y = polysfib2(p1,p2,yl,yh)

% compute synthesis filter bank (polyphase filters)

L = 2*length(p1);
Ly = 2*length(yl)+L-2;
v1 = yl - yh;
v2 = yl + yh;
y = zeros(Ly,1);
y(1:2:Ly) = conv(p1,v1);
y(2:2:Ly) = conv(p2,v2);
y = y(L:Ly+3-L);   % compensate delay of 2 channel filter bank
```

4.5 DTWT/IDTWT with FIR lattice filters

The polyphase filter banks with linear phase filters as discussed before are not suited for the DTWT and IDTWT with orthonormal wavelets. As indicated in (43), the synthesis stage prototype filter must be equal to the time-reversed (and conjugate complex) analysis stage filter. However, **DTWT/IDTWT based on orthonormal wavelets** can be efficiently be implemented with **FIR lattice filters**. Besides a computational efficiency, lattice filters have an inherent perfect reconstruction property which is maintained even with filter coefficient quantization. Unfortunately, we need nonlinear optimization programs to design these filters. The required MATLAB® programs (`fminu()`, and `fminuc()`) are not included in the student edition of MATLAB® but are required by the author's design program `qmf_2hb()`. Therefore, a set of precomputed filter coefficients can be obtained with `QMF_Lat_Coef.m` available at the author's home page.

A two-channel FIR lattice filter bank is sketched in Fig. 22. Similar to polyphase filter banks, all sampling rate conversion is done at the input, and output, respectively.

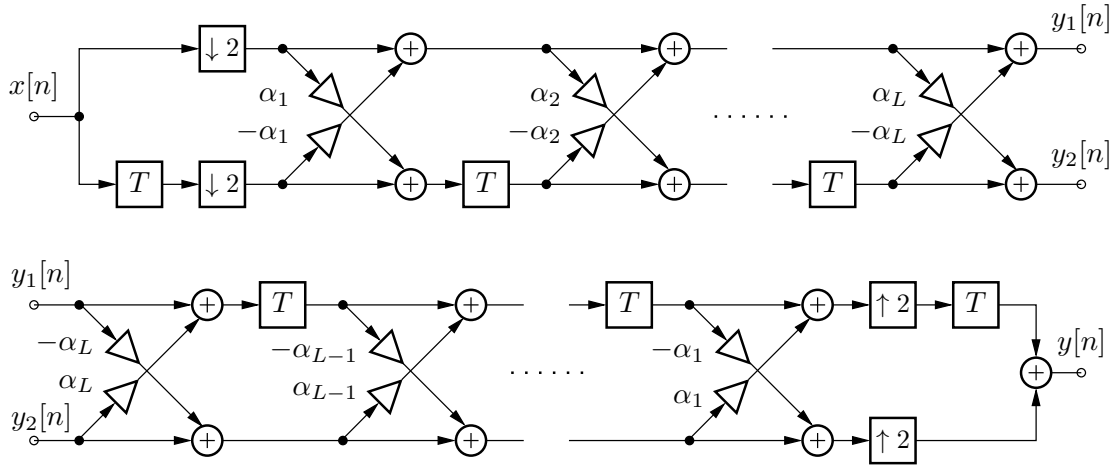


Figure 22: Analysis stage (above) and synthesis stage (below) of a two-channel maximally decimated FIR lattice filter bank

A MATLAB® program of the analysis stage, and of the synthesis stage is listed below. An additional scaling factor $1/\sqrt{\prod_{k=1}^{L}(1+\alpha_k^2)}$ at the input is needed to obtain an overall gain equal to one. The code proposals can by used for higher order DTWTs/IDTWTs. However, delay compensation is tricky too.

```
function [y_lp,y_hp] = latafib2(x,alpha)
%function [y_lp,y_hp] = latafib2(x,alpha)
%
% 2-channel critically sampled lattice analysis filter bank
%
% x             input signal vector
% alpha         vector of lattice coefficients created with QMF_Lat_Coef()
```

```
%              or qmf_2hb()
% y_lp, y_hp   output signal vectors of low pass and high pass filters
%
% G. Doblinger, TU-Wien, 6-1998

Nx = length(x);

% compute lattice input signals p0 (upper path), and q0 (lower path)

p0 = x(:)/sqrt(prod(1+alpha.^2));     % apply gain factor
q0 = [0 ; p0(1:Nx-1)];                % delay q-signal
p0 = p0(2:2:Nx);                      % down-sampling at input
q0 = q0(2:2:Nx);
Nq = length(q0)-1;

% compute lattice stages output signals

p = p0 - alpha(1)*q0;
q = q0 + alpha(1)*p0;
for n = 2:length(alpha)               % loop of remaining stages
   q = [0 ; q(1:Nq)];                 % delayed q-vector
   pt = p;
   p = p - alpha(n)*q;
   q = q + alpha(n)*pt;
end
y_lp = p;
y_hp = q;

% ------------------------------------------------------------------

function y = latsfib2(y_lp,y_hp,alpha)
%function y = latsfib2(y_lp,y_hp,alpha)
%
% 2-channel critically sampled lattice synthesis filter bank
%
% y_lp, y_hp   input signal vectors
% alpha        vector of lattice coefficients
% y            output signal vector
%
% G. Doblinger, TU-Wien, 6-1998

Nx = length(y_lp);
Nq = Nx - 1;
alpha = alpha(end:-1:1);             % use reversed coefficients

% compute first stage output signals

y_lp = y_lp(:);
```

```
y_hp = y_hp(:);
p = y_lp + alpha(1)*y_hp;
q = y_hp - alpha(1)*y_lp;

for n = 2:length(alpha)              % loop of remaining stages
   p = [0 ; p(1:Nq)];                % delayed p-vector
   pt = p;
   p = p + alpha(n)*q;
   q = q - alpha(n)*pt;
end

% up-sampling at output

Nx = 2*Nx;
p_int = zeros(Nx,1);
p_int(2:2:Nx) = p;
q_int = zeros(Nx,1);
q_int(2:2:Nx) = q;
p_int = [0 ; p_int(1:Nx-1)];
y = (p_int + q_int)/sqrt(prod(1+alpha.^2));
```

5 MATLAB® problems and experiments

5.1 Discrete-time signals

Problem 5.1

Plot the following signals in time domain using a time index interval $n \in [-20, 20]$.

$$
\begin{aligned}
x_1[n] &= \delta[n] \quad \text{(unit impulse)} \\
x_2[n] &= \sigma[n] \quad \text{(unit step function)} \\
x_3[n] &= (-1)^n \\
x_4[n] &= \begin{cases} 0 & n \leq 0 \\ n & n > 0 \end{cases} \\
x_5[n] &= \begin{cases} -5 & n \leq -5 \\ n & -5 < n < 5 \\ 5 & n \geq 5 \end{cases}
\end{aligned}
$$

Try to use different MATLAB® instructions for each signal generation.

Problem 5.2

A discrete-time cosine signal with frequency θ is represented by

$$x[n] = \cos \theta n \quad \text{with } n \in [-20, 20].$$

Plot this signal for a set of values θ and comment on the observed signal properties (period, aliasing, unique representation).
Investigate the connection of $x[n]$ to sampling of the continuous-time signal $x_a(t) = \cos \omega t$ with $x[n] = x_a(nT)$. Sampling frequency is $f_s = 1/T$.

Problem 5.3

Write a MATLAB® script to efficiently create and plot the following signals using $n \in [-20, 20]$ and $\alpha \in [-1, 1]$.

$$
\begin{aligned}
x_1[n] &= \alpha^n \sigma[n] \\
x_2[n] &= (n+1)\alpha^n \sigma[n] \\
x_3[n] &= \alpha^{|n|} \\
x_4[n] &= \begin{cases} 0 & n < 0 \\ 1 - \alpha^n & 0 \le n < 10 \\ 1 - \alpha^{10} & n \ge 10 \end{cases} \\
x_5[n] &= \sigma[n] \sum_{k=0}^{n} \alpha^k
\end{aligned}
$$

Compute $x_5[n]$ with and without using the sum operation.

Problem 5.4

Write a MATLAB® script to efficiently create and plot the following signals using $n \in [-20, 20]$. Remember to use dot-operators like `.*` or `./` or `.^` in case of elementwise operations in expressions with vector data.

$$
\begin{aligned}
x_1[n] &= \sigma[n+4] - \sigma[n-4] \\
x_2[n] &= e^{-0.1(n+10)} \sigma[n+10] \\
x_{3_\pm}[n] &= \frac{1}{2}\Big(x_2[n] \pm x_2[-n]\Big) \\
x_4[n] &= x_2[-n-10] \\
x_5[n] &= x_1[(n-4) \oplus 14]
\end{aligned}
$$

($\oplus$ is the modulo operation.)

Problem 5.5

Develop a MATLAB® function to convolve 2 signals of different lengths by creating a file with contents

```
function y = conv_lin(h,x)

Nx = length(x);
Nh = length(h);
Ny = ...

% compute convolution matrix H
...
% compute result vector
y = ...
```

Test your program with the following signals:

a) $h[n] = 1/10,\ 0 \le n \le 9,\ x[n] = \sin\frac{\pi}{5}n,\ 0 \le n < 50$

b) $\mathbf{h} = (1, -1, 1)^T,\ x[n] = \sin\frac{\pi}{3}n,\ 0 \le n < 30$

c) $h[n] = \delta[n] - \delta[n-1],\ x[n] = \sigma[n]$.

Check your results with MATLAB® function `conv()`. Is there a difference in precision and speed when using lengthy signals (i.e. $0 \le n \le 1000$). You may want to use `tic` and `tac` as a stop watch.

Problem 5.6

Use these cryptic instructions to create an m-file `conv_cyc.m` to implement cyclic convolution:

```
h = h(:);
h1 = [h(2:N) ; h];
ic = [0:N-1]';
ir = N:-1:1;
Hc = ic(:,ones(N,1)) + ir(ones(N,1),:);
Hc(:) = h1(Hc);
```

Do you understand these commands? Test your code with the following signals:

a) $N = 10,\ h[n] = (-1)^n,\ 0 \le n \le N-1,\ x[n] = 1,\ 0 \le n \le N-1$

b) repeat a) with $N = 11$

c) $N = 10$, $h[n] = \delta[n] - \delta[n-1]$, $0 \leq n \leq N-1$, $x[n] = n$, $0 \leq n \leq N-1$. Compare cyclic and linear convolution in that case.

Problem 5.7

We create a noisy sinusoid with the following instructions:

```
Nx = 200;
SNR = -5;                       % SNR in dB
                                % (SNR = 20*log20(R))
w0 = 2*pi/Nx*5;                 % frequency of sinusoid
x = sin(w0*[0:Nx-1]);
r = randn(size(x));             % create gaussian noise
R = norm(x)/norm(r);            % compute actual SNR
gain = R * 10^(-SNR/20);        % gain factor needed for
                                % given SNR
xn = x + gain*r;                % add noise to create
                                % noisy measurement data
```

Use an autocorrelation function to measure period (or frequency) of the noisy sinusoid at different values of signal-to-noise ratio SNR. Calculation of period length can be carried out by measuring intervals between zero crossings.

a) Determine the SNR value at which measurement errors increase dramatically.

b) Investigate measurement accuracy as a function of signal length `Nx`.

c) Compare this method by measuring period length directly (i.e. without using a correlation operation).

Hint: A simple method to compute zero crossings is to convert the autocorrelation samples to a rectangular signal and take the difference between adjacent samples. Thus, impulses at zero crossings are obtained and their positions can be determined with the `find` instruction.

5.2 Discrete-time systems

Problem 5.8

Use MATLAB® to analyze the following systems described by input/output relationships. Determine system properties like linearity, causality, stability, and time-invariance. The "unknown" systems are given by means of MATLAB® instructions:

a) **System 1:** `y = 2*x + 0.5;`

b) **System 2:** `y = [x(1) ; x(2:N)-x(1:N-1)];`

c) **System 3:**
`y = conv([1 1 1 1 1],x); y = min(y,1); y = max(y,-1);`

d) **System 4:** `y(1:2:N) = x(1:2:N); y(2:2:N) = -x(2:2:N);`

e) **System 5:** `y = [0:N-1]' .* x(:);`

f) **System 6:** `y = filter(1,[1 -0.9],x,0.5);`

Write a MATLAB® function `blackbox.m` where each system can be selected by function argument `systype`.

```
function y = blackbox(systype,x)
x = x(:);
N = length(x);
if     systype == 1,   y = 2*x + 0.5;
elseif systype == 2,   y = [x(1) ; x(2:N)-x(1:N-1)];
elseif ...
end
```

Use different input signals of length N. Do not always use a unit impulse to test the systems!

Problem 5.9

Find by theory and by a MATLAB® experiment whether the systems with impulse responses $h_1[n]$, $h_2[n]$ can be interchanged in a cascade connection, i.e.

$$\sum_{k=-\infty}^{\infty} h_1[k]\, h_2[n-k] = \sum_{k=-\infty}^{\infty} h_1[n-k]\, h_2[k], \quad \forall n.$$

a) $h_1[n] = \sigma[n]$ (step function) and $h_2[n] = \delta[n] - \delta[n-1]$
Use $x[n] = \sigma[n]$ to test the total system with MATLAB®.

b)

$$h_1[n] = \begin{cases} \frac{1}{N} & 0 \le n \le N-1 \\ 0 & \text{else} \end{cases} \qquad h_2[n] = \sin \frac{2\pi}{N} n \, \sigma[n].$$

Apply $x[n] = \cos \frac{2\pi}{N} n$ as test signal in this case.

Problem 5.10

Determine with MATLAB® the impulse response $h_i[n]$ of an inverse system. The given system has impulse response $h[n]$. If the inverse system is connected in cascade with the given system, then the overall impulse response is a delayed unit impulse:

$$\sum_{k=-\infty}^{\infty} h[n-k] \, h_i[k] = \delta[n - N_0].$$

This system of linear equations can be solved to obtain $h_i[n]$ (see the \ operator in MATLAB®). Note that only finite length impulse responses $h[n]$ and $h_i[n]$ can be used in MATLAB® . An alternative solution is obtained with

```
N = length(h);
Nd = N + Ni - 1;
d = [1 ; zeros(Nd-1,1)];
hi = filter(1,h,d);
```

Why does this piece of code work?

a) $h[n] = (0,8)^n \sigma[n]$

b) $h[n] = (0,8)^n \cos \frac{3\pi}{10} n \, \sigma[n]$

c)

$$h[n] = \begin{cases} 0,5 \left(1 + \cos \frac{2\pi}{N+1}(n+1)\right) & 0 \le n \le N-1 \\ 0 & \text{else} \end{cases}$$

Are all of the given systems invertible? What is the influence of increasing length N of $h[n]$ and N_i of $h_i[n]$? Notice that in general N is not equal to N_i. How should we select N_0? Check the quality of both inversion methods with $x[n] = \sigma[n]$ as input to the cascade connection of $h[n]$ and $h_i[n]$.

Problem 5.11

Use MATLAB® to find output signals $y[n]$ of the following systems using an input signal $x[n] = \cos\theta_0 n$ with different θ_0 values.

a) $h[n] = (0,9)^n \cos\frac{2\pi}{32} n\,\sigma[n]$

b) $h[n] = (0.9)^{|n|}$

c) $H\left(e^{j\theta}\right) = \dfrac{1}{1 - e^{j\theta}}$

Problem 5.12

Solve the given difference equations with MATLAB® functions `filter()` and `filtic()`:

a) $y[n] = 0.5y[n-1] + x[n] - 0.5x[n-1]$
for $x[n] = \sigma[n]$ and initial condition $y[-1] = 1$.

b) $y[n] - 0.6y[n-1] - 0.16y[n-2] = 5x[n]$
for $x[n] = \delta[n]$ and initial conditions $y[-2] = \frac{25}{4}$, $y[-1] = 0$.

Show that the system response is composed of

$$y[n] = y_{zs}[n] + y_{zi}[n]$$

where $y_{zs}[n]$ is the zero-state response and $y_{zi}[n]$ denotes the zero-input response.

Problem 5.13

Write a MATLAB® function `[H,h,p0,z0] = eval_sys(b,a)` to plot causal impulse response, frequency response (magnitude and phase), and poles and zeros of a discrete-time system. Numerator and denominator polynomial coefficients of the given transfer functions are stored in vectors `b`, and `a`, respectively.

a) $H(z) = \dfrac{\alpha z^2 + (1+\alpha)\beta z + 1}{z^2 + (1+\alpha)\beta z + \alpha}$ with $\alpha = 0,8$ and $\beta = 0,5$.

b) $H(z) = \dfrac{2z(3z+17)}{(z-1)(z^2 - 6z + 25)}$

c) $H(z) = \dfrac{1 - z^{-6}}{1 - z^{-1}}$

d) $y[n] + 2y[n-1] + \frac{3}{2}y[n-2] + \frac{1}{2}y(n-3) = x[n] + x[n-1]$

Problem 5.14

In this problem, you will compare direct form with cascade form implementations of IIR filters in regard to sensitivity to filter coefficient tolerances. Quantization of a filter coefficient matrix `C` to a word length of `Nbits` may be carried out with

```
scale = 2^(Nbits-1);
Cq = 1/scale*round(scale*C);
```

Use this quantization rule to write a function `compare_filt(Nbits)` to plot frequency responses of digital IIR filters with accurate, and with quantized coefficients, respectively. Test the system behavior with word lengths between 10 bit and 32 bit. As an example, the following elliptic low pass filter may be used:

```
N = 7;                              % N = filter order
[z0,p0,k] = ellip(N,0.1,70,1/8);    % get poles and zeros
[b,a] = ellip(N,0.1,70,1/8);        % or numerator and denominator
                                    % coefficients
```

Hint: Coefficients of a cascade form can be obtained with function `tf2sos()`.

Problem 5.15

Extend your function `compare_filt(Nbits)` of problem 5.14 to investigate the parallel form of digital filters as well. The required partial fraction expansion is obtained with function `residuez()`. Compare direct, cascade, and parallel form by calculating the error measure

$$\varepsilon^2 = \frac{\sum_{n=0}^{\infty}(h[n] - h_q[n])^2}{\sum_{n=0}^{\infty} h^2[n]}\,.$$

Use a large upper limit to approximately sum up the infinite length impulse response values. Determine the best performing filter structure, i.e. that one showing the least error ε as a function of `Nbits`. Hint: Function `norm()` can be applied to compute a vector norm like the square root of a sum of squares.

Problem 5.16

The theoretical background of this problem can be found in the course book. Therefor, we summarize only the following equations of the block state-space representation of digital filters:

$$\mathbf{x}[n+M] = \mathbf{A}_M \mathbf{x}[n] + \mathbf{B}_M \mathbf{u}[n],$$

with $N \times N$ matrix

$$\mathbf{A}_M = \mathbf{A}^M$$

and $N \times M$ matrix

$$\mathbf{B}_M = \left(\mathbf{A}^{M-1}\mathbf{B}, \mathbf{A}^{M-2}\mathbf{B}, \ldots, \mathbf{AB}, \mathbf{B}\right).$$

$$\mathbf{y}[n] = \mathbf{C}_M \mathbf{x}[n] + \mathbf{D}_M \mathbf{u}[n]\,,$$

with $M \times N$ matrix

$$\mathbf{C}_M = \begin{pmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{M-1} \end{pmatrix}$$

and $M \times M$ lower triangular matrix

$$\mathbf{D}_M = \begin{pmatrix} \mathbf{D} & 0 & 0 & \ldots & 0 \\ \mathbf{CB} & \mathbf{D} & 0 & \ldots & 0 \\ \mathbf{CAB} & \mathbf{CB} & \mathbf{D} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{CA}^{M-2}\mathbf{B} & \mathbf{CA}^{M-3}\mathbf{B} & \mathbf{CA}^{M-4}\mathbf{B} & \ldots & \mathbf{D} \end{pmatrix}.$$

Creating the required matrices is a very good MATLAB® exercise. Write a function `tf2blockss()` to build matrices $\mathbf{A}_M$, $\mathbf{B}_M$, $\mathbf{C}_M$, and $\mathbf{D}_M$ as a function of filter coefficient vectors `a`, `b`, and block length M. Test the block processing with the following MATLAB® code example:

```
M = 40;                              % block length
Nu = 5*M;                            % signal length is a multiple of M
u = ones(Nu,1);                      % input signal
[b,a] = ellip(5,0.1,70,1/8);         % design an elliptic low pass filter
[Am,Bm,Cm,Dm] = tf2blockss(b,a,M);   % block state-space matrices
Ns = max(length(a),length(b))-1;     % state vector length
x = zeros(Ns,1);                     % initial state vector
y = zeros(size(u));

for n = 1:M:Nu                       % block filter operation
   nb = n:n+M-1;                     % block time indices
   y(nb) = Cm*x + Dm*u(nb);
   x = Am*x + Bm*u(nb);
end
```

Problem 5.17

Given is the following piece of MATLAB® code:

```
[b,a] = ellip(7,0.1,100,1/8); % elliptic filter design
[A1,B1,C1,D1] = tf2ss(b,a);   % compute state-space decomposition
[V,L] = eig(A1);              % compute eigenvalue decomposition
A1 = diag(L);                 % A1 = vector of diagonal elements of L
B1 = inv(V)*B1;
C1 = C1*V;

% compute block state-space matrices Am, Bm, Cm, Dm

N = length(A1);               % state-space dimension
M = round(sqrt(2*N));         % optimum block length
Bm = zeros(N,M);
Cm = zeros(M,N);
Am = 1;
for n = M-1:-1:1
   Am = Am .* A1;
   Bm(:,n) = Am .* B1;
   Cm(M-n+1,:) = C1 .* (Am.');
end
Am = Am .* A1;
Bm(:,M) = B1;
Cm(1,:) = C1;
Dm = toeplitz([D1 ; real(Cm(1:M-1,:)*B1)],[D1 zeros(1,M-1)]);
```

Use this example code to develop a function `y = blockss_filt(b,a,x)` implementing signal filtering with a block state-space representation. Note that the matrices involved are complex-valued, However, filter coefficients, input signal, and output signal are real-valued. Take that into account when computing the output signal. Compute the number of arithmetic operations per sampling interval and compare this number with that one of a cascade form.

5.3 Discrete Fourier transform

Problem 5.18

We would like to approximate a periodic signal $\tilde{x}[n]$ with period N by a truncated Fourier series having less than N coefficients:

$$\tilde{x}_M[n] = \frac{1}{N} \sum_{k=0}^{M-1} c_k e^{j\frac{2\pi}{N}nk},$$

with $M \leq N$. Calculate the error

$$\varepsilon(M) = \sum_{k=0}^{N-1} |\tilde{x}[n] - \tilde{x}_M[n]|^2$$

as a function of M using various periodic signals with period N. In addition, check Parseval's equation

$$\sum_{n=0}^{N-1} |\tilde{x}[n]|^2 = \sum_{k=0}^{N-1} |c_k|^2.$$

Hint: Use `c = fft(x,M)` to compute Fourier series coefficients efficiently.

Problem 5.19

In this problem, you will compare execution times of DFT and FFT implementations. First, write a function to compute the DFT and its inverse (IDFT) by directly evaluating their defining equations. In order to avoid trigonometric function evaluation, a table lookup should be used. Second, include

your MATLAB® code in a function `dft_compare(N)` which creates a complex-valued random input signal of length N and compares execution times of your DFT + IDFT implementation with the built-in FFT + IFFT of MATLAB® (DFT + IDFT means DFT followed by IDFT, i.e. output should be equal to input). In addition, measure the RMS error between the input signal and the IDFT/IFFT output signal.
Hint: Use `tic` and `tac` as a stop watch.

Problem 5.20

Frequency domain interpolation can be performed with the following relationship:

$$\begin{aligned} X\left(e^{j\theta}\right) &= \sum_{n=0}^{N-1} x[n] e^{-j\theta n} \\ &= \sum_{k=0}^{N-1} X[k] \underbrace{\frac{1}{N} \sum_{n=0}^{N-1} e^{-j\left(\theta - \frac{2\pi}{N}k\right)n}}_{G_N\left(e^{j\left(\theta-\frac{2\pi}{N}k\right)}\right)} = \sum_{k=0}^{N-1} X[k]\, G_N\left(e^{j\left(\theta-\frac{2\pi}{N}k\right)}\right), \end{aligned}$$

where $x[n]$ is a length N signal with DFT $X[k]$. Thus, we need to know only the N-point DFT $X[k]$ to compute $X\left(e^{j\theta}\right)$ at all desired frequencies θ.
Plot interpolation function $G_N\left(e^{j\theta}\right)$ for various values of N. Determine the signal $g_N[n]$ that is the inverse Fourier transform of $G_N\left(e^{j\theta}\right)$.
Finally, compute the FFTs of length $L = 32$ of the given length $N = 10$ signals using the above interpolation formula. As an alternative, compute the FFTs by zero-padding the given signals. Do you observe any difference?

a) $x[n] = \delta[n], \quad n = 0, 1, \ldots, 9$

b) $x[n] = \delta[n-1], \quad n = 0, 1, \ldots, 9$

c) $x[n] = e^{-0{,}1n^2}, \quad n = 0, 1, \ldots, 9$

Problem 5.21

A decimated DFT may be implemented in MATLAB® as follows:

```
x = x(:);                   % signal stored as column vector
Nx = length(x);
M = ceil(Nx/L);             % L = number of frequency points
N = M*L;                    % N is a multiple of L
x = [x ; zeros(N-Nx,1)];    % append zeros, if necessary
Xb = zeros(L,M);            % matrix for storage of M signal blocks
                            % of length L
Xb(:) = x;                  % columns of Xb are subsequent signal
                            % blocks of length L
xb = sum(Xb.');             % compute row sums of Xb
X = fft(xb);                % decimated FFT of length L
```

Use signals like a short portion of speech to investigate how decimation changes the signal spectrum.
Hint: Function `wavread` can be employed to read audio WAV-files.

Problem 5.22

Given a superposition of 2 sinusoids

$$x[n] = \sin \theta_1 n + \sin \theta_2 n,$$

observe the spectrum of $x_w[n] = w[n]x[n]$ as a function of $\Delta\theta = \frac{\theta_2 - \theta_1}{2}$, and of window function $w[n]$. Determine which window offers the best separation of frequency components, and which yields the most accurate measurements of the spectral amplitudes. Choose $\theta_1 = \theta_0 - \Delta\theta$ and $\theta_2 = \theta_0 + \Delta\theta$ where θ_0 is some fixed frequency and $\Delta\theta$ varies in some interval.

Problem 5.23

Write a function `[yl,yc] = convolve(x,h)` to compute linear and cyclic convolutions by means of FFT and IFFT. Compare both convolution methods using the following signals:

a)

$$x[n] = 1,\ n = 0, \ldots, 9 \quad h[n] = 1,\ n = 0, \ldots, 24$$

b)

$$x[n] = \begin{cases} 1 & 0 \le n \le 9 \\ 0 & 10 \le n \le 15 \end{cases} \qquad h[n] = \begin{cases} 1 & 0 \le n \le 24 \\ 0 & 25 \le n \le 31 \end{cases}$$

c)

$$x[n] = \begin{cases} 1 & 0 \le n \le 9 \\ 0 & 10 \le n \le 15 \end{cases} \qquad h[n] = \delta[n] - \delta[n-10],\ 0 \le n \le 15$$

Explain any observed difference between linear and cyclic convolution.

Problem 5.24

A MATLAB® program of overlap-add convolution looks like this:

```
x = x(:);
h = h(:);
Nx = length(x);
Nh = length(h);
M = L - Nh + 1;                    % block shift (L = FFT length)
Nx1 = M*ceil(Nx/M);                % make length of x a multiple of M
x = [x ; zeros(Nx1-Nx,1)];         % pad with zeros
H = fft(h,L);                      % filter transfer function
x = [zeros(Nh-1,1) ; x];           % add leading zeros to input signal
Nx1 = Nx + Nh - 1;                 % new input signal length
y = zeros(Nx1,1);                  % init. output signal vector

for n = 1:M:Nx1-L+1                % block processing loop
   y1 = ifft(fft(x(n:n+L-1),L).*H);  % FFT - multiply by H - IFFT
   y(n:n+M-1) = y1(Nh:L);          % discard first Nh samples
end

y = y(1:Nx);
if ~any(imag(x)) & ~any(imag(h))   % real-valued signals x, h ?
   y = real(y);                    % avoid tiny imaginary parts
end
```

Modify this sample program to implement the overlap-save method. Test your program with different impulse responses, like those of FIR low pass filters (e.g. `h = fir1(64,1/8)`). Compare your results with `y = filter(h,1,x)`.

Problem 5.25

The chirp DFT is presented in the course book. It can be used to efficiently zoom into a signal spectrum. In MATLAB® , function `czt()` computes the chirp z-transform (and the chirp DFT too). Compare execution times τ_e of chirp DFT and FFT when applied to signal spectra with a resolution Δf in frequency intervals $[f_1, f_2]$ as given in the table. Sampling frequency is fixed to 16 kHz in all cases.

f_1 (Hz)	f_2 (Hz)	Δf (Hz)	τ_e Chirp-DFT	τ_e FFT
0	8000	100		
0	8000	10		
0	8000	1		
1000	2000	100		
1000	2000	10		
1000	2000	1		

5.4 Digital filter design

Problem 5.26

Modify the given MATLAB® code to design FIR low pass filters with impulse responses of even and odd lengths, and of even and odd symmetry.

```
N1 = 40;                    % filter length minus 1
n = 1:N1/2;
fc = 1/10;                  % cutoff frequency normalized to Fs/2
h = sin(pi*fc*n)./(pi*n);   % one-sided discrete-time sinc-function
h = [h(N1/2:-1:1), fc, h];  % include symmetric part and midpoint
plot(abs(fft(h)));
zplane(h,1);                % plot poles and zeros
```

Use the following function prototype to include all 4 cases. Impulse response length is `N`, and `fc` is the cutoff frequency normalized to half the sampling frequency. Parameter `even_odd` is used to specify the required symmetry.

```
function h = firsym(N,fc,even_odd)
N2 = fix(N/2);
```

```
n = 1:N2;
if mod(N,2) == 1
   h = sin(pi*fc*n)./(pi*n);
   if strcmpi(upper(even_odd),'e')
      h = ...
   else
      h = ...
   end
else
   ...
end
```

Problem 5.27

A basic MATLAB® function to implement the frequency sampling method of an FIR filter design is given. In case of a low pass filter, we sample an ideal (rectangular) desired magnitude response and include a linear phase term. Transforming the sampled ideal transfer function to the time domain yields the FIR filter impulse response.

```
function [h,Hid] = firfs(N,fc)

N2 = fix((N-1)/2);
Np = fix(fc*N/2);
Hmag = zeros(1,N);
Hmag(1:Np+1) = ones(1,Np+1);      % magnitude response from 0 to pi
                                  % (ideal low pass filter)
Hmag(N-N2+1:N) = Hmag(N2+1:-1:2); % magnitude resp. from pi to 2*pi,
                                  % using (even) symmetry property
% create linear phase response

phi = zeros(1,N);
phi(1:N2+1) = -pi*(N-1)*(0:N2)/N; % phase response from 0 to pi
phi(N-N2+1:N) = -phi(N2+1:-1:2);  % phase response from pi to 2*pi,
                                  % using (odd) symmetry property

% compute ideal frequency response and FIR filter impulse response

Hid = Hmag .* exp(j*phi);
h = real(ifft(Hid));        % real() avoids a very small imaginary
                            % part caused by rounding errors
```

By running this program, you will notice that only a low stop band attenuation can be achieved. One idea to improve the approximation performance is to

place additional frequency points within transition regions to obtain ramp-like transition bands.
Modify `firfs()` to include up to 3 additional transition band frequency points, and try to optimize their positions in regard to a maximum stop band attenuation.

Problem 5.28

Develop a MATLAB® function `h_min = lin2min(h_lin)` to transform a linear phase FIR filter to a minimum phase filter. Start with impulse response `h_lin` and find the transfer function zeros with `roots()`. Invert all zeros outside the unit circle in the z-plane to obtain a minimum phase filter with all its zeros inside (or on) the unit circle. Use `poly()` to obtain the filter coefficients of the minimum phase FIR filter, and correct the filter gain.
Do you notice any difference when plotting frequency responses (magnitude and phase) of both filters? Plot the impulse responses as well and observe the different shapes. Finally, determine the maximum filter length up to which this methods works.

Problem 5.29

The following MATLAB® example can be used to design FIR filters by means of a least-squares method:

```
L = length(f);                          % number of frequency points
if mod(N,2) == 0                        % even filter length N ?
   M = N/2;
   S = 2*cos(pi*f(:)*(M - 0.5 - [0:M-1])); % system matrix
else
   M = (N-1)/2;
   S = [2*cos(pi*f(:)*(M - [0:M-1])) , ones(L,1)];
end
h = S \ m(:);                           % solve LS-problem
h = [h ; h(M:-1:1)];                    % append symmetric part
```

The magnitude of the desired frequency response must be specified in vector `m` at frequency points given in vector `f`. Modify the given example to get FIR

filters with odd-symmetrical impulse responses. Design a Hilbert transformer with the desired frequency response

$$H\left(e^{j\theta}\right) = e^{-j\theta\frac{N-1}{2}} e^{-j\frac{\pi}{2}} \operatorname{sign}\theta$$

(sign x is the sign function). What deviations from the ideal response do you notice? Is it better to use an even or an odd filter length? Plot poles and zeros in the complex z-plane. Is it possible to design a minimum phase Hilbert transformer, e.g. with your solution of problem 5.28?

Problem 5.30

Use your program of problem 5.29 to design the following filters:

a) FIR filter with a Gaussian magnitude response,

b) FIR differentiator, i.e. a filter with a magnitude response proportional to frequency,

c) FIR filter to convert white noise to $1/f$ noise (power spectral density inversely proportional to frequency, down to a lower cutoff frequency f_c),

d) FIR filter to approximate the mean threshold of hearing according to the magnitude response (in dB)

$$T_q(f) = 3.64\left(\frac{f}{1000}\right)^{-0.8} - 6.5\, e^{-0.6\left(\frac{f}{1000}-3.3\right)^2} + 0.001\left(\frac{f}{1000}\right)^4$$

with frequency f in Hertz.

Problem 5.31

Write a MATLAB® function `h = firkw(fc,fr,ar)` to design low pass FIR filters with a Kaiser window function. Properties of the Kaiser window function are presented in the course book. Arguments `fc`, `fr` are pass band edge, and stop band edge, respectively (both normalized to half the sampling frequency). Stop band attenuation is given by `ar`. Note that a Kaiser window applied to an ideal impulse response yields a 6 dB attenuation at f_c. To obtain a smaller attenuation, you can modify the pass band edge with $f_c \to f_c + (f_r - f_c)/2$. Check your design with results obtained by function `fir1()`.

Problem 5.32

Extend your MATLAB® function `firkw()` to design several common filter types. Include an additional function argument (`typ = 'lp'` or `'hp'` or `'bp'` or `'bs'` or `'di'` or `'hi'`) to select the filter characteristic.

Problem 5.33

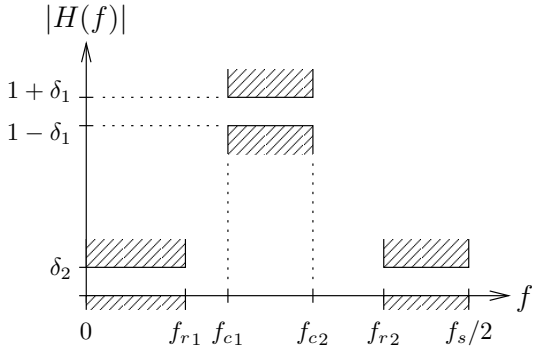
In this problem, you will compare an equi-ripple low pass FIR filter design with the Kaiser window method (function `firkw()` of problem 5.31). Write a function `h = firrem(fc,fr,ar,rp,N)` with pass band edge `fc`, stop band edge `fr`, stop band attenuation `ar` (in dB), and pass band ripple `rp` (in dB). Filter length `N` is optional, i.e. it is estimated by the program if not supplied (see `firpmord()`). Use appropriate weighting vectors in function `firpm()` to give preference to stop band attenuation. Compare your design results with those of function `firkw()`. If you are lucky, you may observe a convergence failure of the Remez algorithm in certain cases (see the next problem).

Problem 5.34

This problem shows a convergence failure of the Remez algorithm in case of a band pass filter. First, design a symmetrical band bass filter with the following specifications:

$$f_{r1} = 0.5, f_{c1} = 0.6, f_{c2} = 0.8, f_{r2} = 0.9, f_s = 10, a_r = 60, r_p = 0.2$$

with frequencies in kHz, a_r, r_p in dB, and $a_r = -20 \log_{10} \delta_2$, $r_p = 20 \log_{10} \frac{1+\delta_1}{1-\delta_1}$.



Second, change upper stop band edge to $f_{r2} = 1$ kHz and notice what happens with the frequency response. Try to interpret your observation.

Problem 5.35

The following example is a least-squares design of a low-delay FIR low pass filter (see section 4.1.5 of the course book):

```
N = 101;                    % filter length
Ndoff = 20;                 % delay should be smaller by Ndoff samples
                            % than for linear phase filter
Nd = (N-1)/2 - Ndoff;       % desired filter delay
Lpass = 10*N;               % number of frequency point in pass band
Lstop = 10*N;               % number of frequency point in stop band
fc = 2/8;                   % cutoff frequency normalized to Fs/2
fr = 2.5/8;                 % stop band edge normalized to Fs/2
Wpass = 1;                  % pass band weighting factor
Wstop = 50;                 % stop band weighting factor

theta = pi*[linspace(0,fc,Lpass), linspace(fr,1,Lstop)];
W = [Wpass*ones(1,Lpass), Wstop*ones(1,Lstop)];
D = [exp(-j*theta(1:Lpass)*Nd) zeros(1,Lstop)];

h = lslevin(N,theta,D,W);
```

Function `lslevin()` can be found in the program collection `Mbook.zip`. Observe transfer function magnitude, phase, and group delay when varying parameter `Ndoff` in an interval $[-30, 30]$. What are the maximum errors of magnitude and group delay as a function of `Ndoff`? Compare the given low delay filter with a linear phase FIR filter having an approximately equal magnitude response.

Problem 5.36

Use function `lslevin()` of program collection `Mbook.zip` to design an FIR all pass filter having a desired frequency response $D\left(e^{j\theta}\right) = e^{-j\phi(\theta)}$ with the following phase characteristics:

a) a sinusoidal group delay, i.e. a desired phase

$$\phi(\theta) = \frac{N-1}{2}\theta + 2\pi(1-\cos\theta)$$

with $N = 61$.

b) a chirp all pass filter with phase

$$\phi(\theta) = \frac{N-1}{2}\,\theta - \eta\left(\theta - \frac{\pi}{2}\right)^2$$

with $N = 61$ or $N = 101$, and $\eta = \frac{8}{2\pi}$ or $\eta = \frac{16}{2\pi}$.

Try to find applications of these special filters.

Problem 5.37

Use function `lerner_lp()` of program collection `Mbook.zip` to design digital Lerner filters as presented in section 4.2.1 of the course book. Plot frequency response magnitude, phase, and group delay. Compare Lerner filters with FIR filters showing a similar magnitude response. What are the differences in numbers of additions, multiplications, and delay elements? Which filter has a lower signal delay?

Problem 5.38

Extend function `lerner_lp()` to design a digital band pass filter. The basic method is to shift the analog filter poles from low pass locations to band pass locations. Note that the residues of the partial fraction expansion must have correct signs. Compare your results with those of a standard linear phase FIR filter design.

Problem 5.39

MATLAB® function `prony()` can be applied to design recursive digital filters with special impulse responses. Design IIR filters

a) with a triangular impulse response,

b) with a Gaussian impulse response,

c) to approximate a Hilbert transformer impulse response,

d) to approximate a differentiator impulse response.

Select different parameters N, M in function `prony()` and have a look to transfer function poles and zeros. Compare your results with those obtained with FIR filters.

Problem 5.40

In this problem, you will compare standard approximations of IIR low pass filter frequency responses. Determine transfer function poles and zeros with the following specifications:

$$f_c = 0.800 \text{ kHz},\ f_r = 1.2 \text{ kHz},\ f_s = 10 \text{ kHz},\ r_p = 0.5 \text{ dB},\ a_r = 60 \text{ dB}.$$

Select an appropriate filter order for each approximation type to get similar frequency response magnitudes. Determine which filter exhibits the worst group delay behavior.
Hint: Apply functions `butter()`, `cheby1()`, `cheby2()`, and `ellip()` for the design.

Problem 5.41

We would like to investigate the impulse response behavior of different IIR filters as designed in problem 5.39. Proceed according to the following steps:

a) Compute vector `p` of poles, vector `z` of zeros, and gain factor `k` of the individual IIR filters.

b) Transform transfer function poles and zeros with `a = real(poly(p))` and `b = k*real(poly(z))` in the rational transfer function with denominator coefficient vector `a`, and numerator coefficient vector `b`, respectively. Note that `real()` eliminates possibly small imaginary parts which should be zero but may occur due to rounding in function `poly()`.

c) Use `h = filter(b,a,[1 zeros(1,Ntime)])` to compute the filter impulse responses.

Determine which filter is best suited for band width limitation in data communications. Which filters are strictly minimum phase filters showing all zeros within the unit circle in the complex z-plane? Try to estimate the transfer function polynomials with `[b,a] = prony(h,N,M)`, where `h` is the impulse response obtained in step c), and `a`, `b` are the coefficient vectors as mentioned above. Do you get the same coefficients?

Problem 5.42

Design a high pass filter by transforming a low pass filter transfer function with $H_{hp}(z) = H_{lp}(-z)$. In addition, transform poles and zeros by this method as well and observe their locations with function `zplane()`. Find a formula that describes how pass band edges and stop band edges are changed by this transformation.

Problem 5.43

The transformation $z \to \pm z^2$ applied to $H(z)$ can be generalized by $z \to \pm z^L$ where L is an integer. Apply Fourier transform properties to investigate the consequences of such a transform (i.e. $e^{j\theta} \to e^{j\theta L}$) on the frequency response $H(e^{j\theta})$. Design a narrow-band band pass (or low pass) filter and check your theoretical findings with MATLAB® .

Problem 5.44

We would like to design an IIR filter that approximates the frequency response of the mean threshold of hearing. An empirical relationship of a healthy human auditory system is presented on page 50 in problem 5.30. MATLAB® function `yulewalk()` can be employed to design such a filter. Afterwards, design an inverse filter to compensate the given frequency response. Note that compensation is done by a cascade connection of the approximation filter and its inverse filter. Compare your results with those obtained in problem 5.30 where the approximation filter is an FIR filter.

5.5 Multirate filter banks and wavelets

Problem 5.45

The design of a uniform cosine-modulated filter bank can be efficiently done with function `unicmfb()` of program collection `Mbook.zip`.
Design an example filter bank with `[h,g] = unicmfb(8,256,1/32,1e5,256)`. Afterwards, write a function `[y,Y] = unifib(h,g,x)` to compute a complete analysis/synthesis filter bank with various input signals `x`. Store output signal in vector `y` and subband signals in matrix `Y`. Measure filter bank reconstruction errors both in time and in frequency domain.

Problem 5.46

Solving this problem requires a study of course book section 5.1.2 (pp. 126-142). Use functions `unicmfb()` and `nuftrans()` of the program collection to develop a non-uniform cosine-modulated filter bank. Start by designing 2 uniform filter banks with $N_1 = 8$, and $N_2 = 4$ respectively. Choose FIR filter length $L_1 = 96$ (8 channel filter bank) and $L_2 = 40$ (4 channel filter bank). Next, a transition prototype low pass filter between these 2 filters must be designed with `nuftrans()` (filter length $L_t = 96$). The respective bandpass transition filter is obtained with a cosine modulation applied to the prototype filter impulse response.

Problem 5.47

An FFT filter bank as presented in the course and in the course book in section 5.2 is a versatile tool with a great many of applications. Use `fft_fib()` of the program collection to test an FFT analysis/synthesis filter bank with speech signals. Since speech signals usually have a large length, you have to remodel `fft_fib()` to implement block processing. Note that concatenation of subsequent blocks must not result in any transition effects at block boundaries. At which filter bank decimation factor do you notice a filter bank reconstruction error?
Hint: To avoid excessive memory usage, you should read the speech signal in small chunks from a WAV-file. Apply a similar procedure when storing the filter bank output signal.

Problem 5.48

Use your FFT filter bank implementation of problem 5.47 to design a frequency band equalizer. As an example, we may use such a system to compensate loudspeaker frequency responses. Modify the N FFT output bins with real-valued factors $0 \leq g_k \leq 1,\ k = 1, 2, \ldots, N$ according to a desired equalizer characteristic. Don't forget to take spectral symmetry properties of real-valued input signals into account!

Problem 5.49

To solve this problem, you will need the course book (section 5.3.2) or the MATLAB® wavelet toolbox. Write a function `wtfib(N,h_l,h_h)` to plot the subband frequency responses of a length N DTWT (discrete-time wavelet transform). Impulse responses `h_l`, `h_h` can be found with function `unicmfb()` included in the program collection. Alternatively, the wavelet toolbox offers impulse responses of various wavelet functions.

Problem 5.50

The following code example implements a 2-channel polyphase analysis/synthesis filter bank as discussed in the course book (pp. 159-161):

```
hg = h(1:2:end);          % polyphase filters of low pass prototyp
hu = h(2:2:end);
x = x(:);                 % input signal

% analysis filter bank

u1 = filter(hg,1,x(1:2:end));
u2 = filter(hu,1,[0 ; x(2:2:end-1)]);
y1 = u1 + u2;             % subband signals y1, y2
y2 = u1 - u2;

% synthesis filter bank

v1 = y1 - y2;
v2 = y1 + y2;
y = zeros(size(x));
y(1:2:end) = filter(hg,1,v1);
y(2:2:end) = filter(hu,1,v2);
```

At first, check the correct functioning of this program in case of even and odd signal lengths. Secondly, use function `unicmfb()` (see program collection) to design a prototype filter impulse response $h[n]$ of the 2-channel filter bank (e.g. `h = unicmfb(2,40,1/6,1e5,80);`). Finally, compute the reconstruction error using several input signals. Determine the signal delay between input and output signal.

Problem 5.51

The following function computes the DTWT/IDTWT:

```
function y = DTWT_IDTWT(N,h,x)
% function y = DTWT_IDTWT(N,h,x)
% compute DTWT and IDWT to show reconstruction property
%
% N    number of DTWT coefficients = number of filter bank channels
% h    low pass filter impulse response (linear phase filter)
% x,y  input, and output signal vectors
```

```
%
% Note: do not use unicmfb() to design filter because linear phase
%       filters are required, use e.g. h = qmf_2(50,0.65,0.1)
%       (qmf_2() can be obtained from the author's homepage)
%
% G. Doblinger, TU-Wien, 02-2001

Nx2 = ceil(length(x)/2);   % maximum length of DTWT coefficients
L = length(h);
p1 = h(1:2:end);           % polyphase filters of low pass prototyp
p2 = h(2:2:end);

DTWT = zeros(Nx2+L,N);     % matrix of N DTWT coefficients
yl = x;
for n = 1:N-1
   [yl,yh] = polyafib2(p1,p2,yl);
   DTWT(1:length(yh),n) = yh;
end
DTWT(1:length(yl),N) = yl;

% compute IDTWT = output y of synthesis filter bank

y = DTWT(1:length(yl),N);
for n = N-1:-1:1
   y = polysfib2(p1,p2,y,DTWT(1:length(y),n));
end

% ------------------------------------------------------

function [yl,yh] = polyafib2(p1,p2,x)

% compute analysis filter bank (polyphase filters)

u1 = conv(p1,x(1:2:end));
u2 = conv(p2,[0 ; x(2:2:end-1)]);
yl = u1 + u2;
yh = u1 - u2;

% ------------------------------------------------------

function y = polysfib2(p1,p2,yl,yh)

% compute synthesis filter bank (polyphase filters)

L = 2*length(p1);
Ly = 2*length(yl)+L-2;
v1 = yl - yh;
v2 = yl + yh;
```

```
y = zeros(Ly,1);
y(1:2:Ly) = conv(p1,v1);
y(2:2:Ly) = conv(p2,v2);
y = y(L:Ly+3-L);   % compensate delay of 2 channel filter bank
```

Use this program with the following input signals:

a) discrete-time, delayed rectangular impulse,

b) sinusoidal chirp $x[n] = \sin(\alpha n^2)$,

c) modulated Gaussian impulse $x[n] = e^{-\alpha n^2} \cos(\theta_0 n)$.

The prototype filter impulse response of problem 5.50 may be applied. Check the IDTWT reconstruction property of a cascade connection of DTWT and IDTWT. The cascade connection is already present in the given program.

Problem 5.52

If you would like to investigate the denoising property of the wavelet shrinkage method, then function `wt_dn()` of the program collection is well suited for that purpose. Try to suppress white noise (with different amplitude levels) superimposed to the following signals:

a) speech segments (vowels) or singing voices,

b) signals with step-like transitions,

c) low pass and band pass filtered white noise,

d) random walk signal:

$$x[n] = x[n-1] + r[n]\sigma[n], \quad n = 0, 1, 2, \ldots$$

$x[-1] = 0$, $r[n]$ is zero mean Gaussian white noise, and $\sigma[n]$ is the step function.

5.6 Audio signal processing applications

Background information of the problems in this section can be found in chapter 6 of the course book. In addition, a presentation will be given in the lecture part of the course.

Problem 5.53

In the collection of programs, pitch scaling and time scaling of audio signals are implemented by function `pitch_scale()`, and `time_scale()`, respectively. However, a lot of investigations and improvements are feasible. In this problem, you can study an effect resulting in a slightly reverberant speech at the output of the scaling systems. This effect is more pronounced if we increase the time window length of the FFT filter bank. In order to demonstrate this observation, you should measure impulse responses of the systems. Since the systems involved are time-variant, $\delta[n-k]$-impulses with different delays k must be applied to get the 2-dimensional impulse response $h[n,k]$. A second test with sinusoidal signals of different frequencies should show whether the output signal exhibits a slight amplitude modulation (beat). Is it possible to reduces the beat-effect by a proper window length selection?

Problem 5.54

Audio signal enhancement can be accomplished with function `enhance()` of the program collection. It offers a rich field of activity regarding MATLAB® experiments. A major goal is to find optimum parameters which in general are different for speech and for music. As an example, typical time window lengths should be 20 ms for speech, and 40 ms for music, respectively. Use function `enhance()` to investigate the following issues:

a) At which input SNR do you observe a dramatic performance loss of the denoising process?

b) Can you interpret the denoising process by plotting output signal spectrograms with MATLAB® function `spectrogram()`?

c) What happens if you apply a noisy sinusoidal chirp as used to test wavelet denoising algorithms?

Problem 5.55

In this problem, you will carry out 3 experiments to investigate the distortion introduced by signal enhancment algorithms. These experiments are especially useful when restoring noisy music recordings. The main goal is to find optimum parameters N, α, and ρ needed in function `enhance()`.

a) Use noisy sinusoids with frequencies 400 Hz, 1 kHz, and 3 kHz to determine the threshold at which these signals are audible within the noise but are suppressed by the enhancement system. Try to find optimum parameters offering a good compromise between noise reduction and suppression of the desired signal.

b) Investigate the settling behavior of the enhancement system by on/off switching of noisy sinusoids at the system's input. Measure settling time as a function of window length N.

c) Observe the modulation of enhanced sinusoidal signals by residual noise as a function of input SNR.

6 MATLAB® projects

As an alternative to solving a set of problems, you can choose one of the projects listed in this section. Normally, these projects are worked out in a group of two students. The theoretical background of the projects will be presented in the course lectures. However, further reading is required to successfully finish the project. In the following, I will present a brief overview on possible projects. It has been proved in the past that all projects fit in the time schedule of this course.

6.1 Adaptive filters for interference suppression

Adaptive filters can be used to suppress unwanted signals which are superimposed to a desired signal. As an example consider an ECG signal disturbed by power line signal interference (hum), or by measurement noise. You will investigate basic adaptive filtering algorithms in standard test setups, and in applications to interference cancellation.

6.2 Software audiometer using MATLAB®

Audiometers are used to test hearing. Randomly selected tones with different amplitudes and frequencies are presented to a listener to find the individual threshold of hearing. In addition, tones plus noise can be used to evaluate hearing under noisy environments. We will determine relative audiograms only since we do not calibrate the measurements.

6.3 Voice analysis using MATLAB®

Voice analysis can be used to detect impairments of the human vocal tract like a hoarsely voice. Typically, the analysis is carried out in the frequency domain using the spectrogram.

6.4 Wavelet transform applied to signal denoising

Wavelets enable efficient representations of many natural signals because tiny coefficients of a wavelet decomposition can be removed. Thus, it is possible to suppress unwanted signals by maintaining only strong wavelet coefficients.

6.5 FFT filter bank for frequency dependent dynamic range compression

The FFT filter bank is well suited to design a dynamic range compressor (and expander). In contrast to conventional dynamic range modification, the FFT filter bank enables a frequency dependent compression/expansion characteristic. Such a system can be used to simulate the effect of hearing impairment. Therefore, people with normal hearing can get an impression on how a reduced dynamic range in certain frequency bands degrades sound perception.

6.6 Speaker localization using two microphones and an FFT filter bank

Two microphones and digital signal processing can be applied to locate a speaker moving in front of the microphones. A compass arrow in a MATLAB® figure window should points to the target speaker. It is possible to run the system in real-time if speaker movements are not to fast (as they usually are).

6.7 Data compression applied to audio signals

Efficient digital storage and transmission of audio signals is based on coding methods which suppress signal components not audible by human hearing. An example is MPEG-1 audio coding. Other methods (called lossless audio coding) do not alter the signal. Compression is achieved by decorrelating the signal and coding the remaining (residual) signal. In this project, you can investigate the principles of both methods.

6.8 Data compression applied to ECG signals

Similar to audio and image signal compression, biomedical signals like ECG signals can also be compressed in order to reduce the storage demand. The compression algorithm must take into account the special nature of ECG signals.

6.9 Determination of fitness using ECG signal analysis

Physical fitness of an individual can be approximately measured by monitoring heart rate variability in addition to ECG spectral analysis. Heart rate variability computed by ECG signal analysis will be used as input to a decision algorithm to score the fitness.

6.10 Analysis of lung sounds with spectral analysis

Respiratory sounds (lung sounds) can by analyzed to detect wheezing which occurs e.g. with asthma. Typically, spectral peaks in colored noise must be detected. Alternatively, wavelet analysis can be used to find wheezing patterns.

6.11 MATLAB® real-time spectrum analyzer (audio input from soundcards)

Timer and audio recording functions in MATLAB® enable real-time spectral analysis on today's personal computers. In addition, the program should also act as an oscilloscope to plot signals in time domain, and to compute cross-correlation functions.

6.12 FFT filter bank for time- and pitch-scaling

Altering the pitch of human voices, and of musical instruments can be accomplished by means of an FFT filter bank as discussed in this course. The pitch-scaling algorithm can be extended to achieve time-scaling of recorded signals as well.

6.13 Experiments with an FFT filter bank channel vocoder

The channel vocoder as presented in the appendix can be used to create strange sounding voices like a robotic speech. The FFT filter bank is well suited to efficiently implement this task. Basically, the short-time spectral envelope of the original speech signal is estimated and multiplied with the spectrum of a different signal (e.g. a square wave signal, or a noise signal). Such a modification corresponds to a voice generation with exchanged vocal tract excitation.

A Introductory MATLAB® examples of the course

We summarize the examples presented during the introductory part of the course. Most of them can also be found in the first chapter of the course book. All scripts of this section are contained in the following archives available in the download area of the Institute of Telecommunications:

- Sections A.1, A.2, A.3: `Doblinger_lecture_1.zip`
- Section B: `Doblinger_lecture_3.zip`
- Section C: `Doblinger_chapter5.zip`
- Section D: `Doblinger_wavelet_demo.zip`.

Download instructions will be sent out in a "tiss news" at the beginning of the course. All MATLAB® code segments listed in the following can be run within the MATLAB® editor window by selecting a section (click into the area limited by characters %%), and by pressing "CONTROL+RETURN" (or using the "RUN SECTION" icon of the editor menu bar).

A.1 Signal generation and manipulation

```
% examples of Chapter 1

n = -10:10;                    % create a row vector n containing integers
                               % (time points) -10, -9 , ... 9, 10
                               % (using colon-operator :)
theta = 2*pi/20;               % frequency parameter (pi <=> fs/2)
x = 0.8 * sin(theta*n);        % create a sinusoidal signal stored in
                               % column vector x
stem(n,x);                     % discrete-time plot
xlabel('n'), ylabel('x[n]');

%%
n = -10:10;                    % create a row vector n containing integers
                               % (time points) -10, -9 , ... 9, 10
f = 400;                       % frequency in Hz
Fs = 8000;                     % sampling frequency in Hz
x = sin(2*pi*f/Fs*n);
stem(n,x);
xlabel('n'), ylabel('x[n]');

% show aliasing when sampling an continuous-time signal
% (using function dtcos() of the book)

%%
N = 40;                        % number of samples
```

```
m = 7;
f = m/N*Fs;
fprintf(1,'f = %4.2f Hz\n',f);
dtcos(N,m);                    % no aliasing, f < Fs/2

%%
m = 17;
f = m/N*Fs;
fprintf(1,'f = %4.2f Hz\n',f);
dtcos(N,m);                    % no aliasing, f < Fs/2

%%
m = 27;
f = m/N*Fs;
fprintf(1,'f = %4.2f Hz\n',f);
dtcos(N,m);                    % aliasing, f > Fs/2

%%
close all
nx = 20;
x1 = zeros(1,nx);              % create a row vector of 20 zeros
i1 = 1:2:nx;                   % row vector of odd integers (1,3,5,...19)
x1(i1) = 1;                    % set every other element of x1 to one
stem(x1);                      % plot x1
xlabel('n'), ylabel('x[n]');

%%
n = -20:20;
x = sin(0.1*pi*n);
x1 = x;
%i1 = find(x1 < 0);            % find indices of x1 where vector
                               % are negative
%x1(i1) = 0;                   % replace negative signal parts by zeros
x1(x1 < 0) = 0;                % more compact
subplot(2,1,1),stem(n,x);
xlabel('n'), ylabel('x[n]');
subplot(2,1,2),stem(n,x1);
xlabel('n'), ylabel('x_1[n]');

%%
x2 = 2*x;
Aclip = 1.5;
%i2p = find(x2 > Aclip);
%i2n = find(x2 < -Aclip);
%x2(i2p) = Aclip;              % clip positive half waves
%x2(i2n) = -Aclip;             % clip negative half waves
x2(x2 > Aclip) = Aclip;        % more compact
x2(x2 < -Aclip) = -Aclip;
```

```
subplot(2,1,1),plot([n(1) n(end)],[Aclip Aclip],'r--',...
   [n(1) n(end)],[-Aclip -Aclip],'r--');
hold on
subplot(2,1,1),stem(n,2*x);
hold off
xlabel('n'), ylabel('x[n]');
subplot(2,1,2),stem(n,x2);
xlabel('n'), ylabel('x_2[n]');

%%

x = sin(0.1*pi*n);
x1 = (x > 0);                 % x1 is set to 1 where x > 0, and x1 = 0
                              % elsewhere, i.e. a square wave is created
                              % from the original sinusoid
x2 = sign(x);
subplot(3,1,1),stem(n,x);
xlabel('n'), ylabel('x[n]');
subplot(3,1,2),stem(n,x1);
xlabel('n'), ylabel('x_1[n]');
subplot(3,1,3),stem(n,x2);    % Note: x2 = 0 at n = 0, but x2(10) = 1
xlabel('n'), ylabel('x_2[n]');

% Note: use modulo operation or matrix reshaping (see below) to obtain
% an exact square wave!

%%
close all;
n = -20:20;
N = 5;
x = (mod(n,N) == 0);          % x2 = 1 at multiples of N, i.e. a unit
                              % pulse train with period N is generated
                              % (mod() is the modulo operation)
stem(n,x);
xlabel('n'), ylabel('x[n]');

%%
n = 0:20;                     % vector of time points 0, 1, 2 ... 20
alpha = 0.8;
x = alpha .^ n;               % operator .^ performs a componentwise
                              % exponentiation
stem(n,x);
xlabel('n'), ylabel('x[n]');
```

A.2 Special manipulations of vectors and matrices

```
% elementwise operations on vectors (dot operator like .*)

clc
x = [1 ; 2 ; 3 ; 4 ; 5];      % 5 x 1 column vector
disp('x = '), disp(x)
y = x.*x;                     % elementwise product (Hadamard product)
disp('y = x.*x'),disp(y)
y = x'*x;                     % scalar (inner) vector product
disp('y = x''*x'),disp(y)
y = x*x';                     % outer vector product
disp('y = x*x'''),disp(y)

%%
% Note: if x is complex, then ' denotes conjugate transpose!
%       use .' to transpose a vector

x = [1 ; 1+1i*2];             % use 1i instead of j or i to avoid
                              % mistake with i,j used as indices
disp('x = '), disp(x)
y = x';                       % conjugate complex row vector
disp('y = x'), disp(y)
y = x.';                      % row vector
disp('y = x.'), disp(y)

%%
% special case: repeat columns

x = [1 ; 2 ; 3 ; 4 ; 5];      % 5 x 1 column vector
y = x*ones(1,4);              % operation = multiplication by 1
disp('y = x*ones(1,4)'),disp(y)

% compact form avoiding multiplications by 1 (using colon-operator :)

y = x(:,ones(1,4));           % operation = addressing
disp('y = x(:,ones(1,4))'),disp(y)

%%
% special case: repeat rows

x = [1  2  3  4  5];      % 1 x 5 row vector
y = ones(4,1)*x;
disp('y = ones(4,1)*x'),disp(y)

% compact form avoiding multiplications by 1

y = x(ones(4,1),:);
```

```
disp('y = x(ones(4,1),:)'),disp(y)

%%
% elementwise operations on matrices

clc
A = [1 2 ; 3 4];
disp('A = '), disp(A)
B = A^2;                        % matrix multiplication AA
disp('B = A^2 = AA'), disp(B)
C = A.^2;                       % Hadamard product A o A
disp('C = A.^2 = AoA'), disp(C)

%%
% reshaping matrices to vectors and vice versa using the colon operator

A = [1 2 3 ; 4 5 6];
disp('A = '), disp(A)
a = A(:);                       % a = vec(A), stacking all columns to a vector
disp('a = A(:)'), disp(a)
B = zeros(3,2);
B(:) = a;                       % reshaping vector to matrix
disp('B(:) = a'), disp(B)

%%
% example solving a set of linear equations Ax = b

A = randn(3);
disp('A = '), disp(A)
b = randn(3,1);
disp('b = '),disp(b)
x = A\b;                        % solution of Ax = b
disp('x = A\b'), disp(x)
disp('||Ax-b||'), disp(norm(A*x-b))   % display L2 norm of solution error

%%
% does this work if b is a matrix, i.e. the solution is also a matrix?

clc
A = randn(3);
disp('A = '), disp(A)
B = randn(3);
disp('B = '), disp(B)
X = A\B;                        % solution of AX = B
disp('X = A\B'), disp(X)

% solution using vec-operation and Kronecker product
% AX = B  -> A[x1 x2 x3] = [b1 b2 b3] -> Ax1 = b1, Ax2 = b2, Ax3 = b3
```

```
% -> [A 0 0 ; 0 A 0 ; 0 0 A][x1 ; x2 ; x3] = [b1 ; b2 ; b3]
% more compact: kron(I,A)vec(X) = vec(B), I = identity matrix

x = kron(eye(3),A)\B(:);
disp('x = kron(eye(3),A)\B(:)'), disp(x)
X = zeros(3);
X(:) = x;
disp('X(:) = x'), disp(X)
```

A.3 Time shift, time reversal, convolution, and correlation

```
% time shift (positive delay)

N = 20;
n = 0:N-1;
x = 0.8 .^ n;                 % create exponentially decaying signal
Nd = 10;                      % time delay in samples
y = [zeros(1,Nd-1),x(1:N-Nd+1)];   % add Nd-1 leading zeros to delay signal
                                   % limit to length of x
subplot(2,1,1),stem(n,x);
xlabel('n'), ylabel('x[n]');
subplot(2,1,2),stem(n,y);
xlabel('n'), ylabel('y[n]');

%%
% how to implement a negative delay (advance)?

N = 20;
x = 0.9 .^ (0:N);             % create a row vector of exponentially
                              % decaying signal samples
x = [zeros(1,N) x];           % add leading zeros to vector x to delay
                              % signal in order to use negative delays
Nd = 10;
x1 = x(Nd+1:end);             % shift signal to the left
x2 = [zeros(1,Nd-1) x];       % shift singal to the  right
n = 0:length(x)-1;
subplot(3,1,1),stem(n,x);
xlabel('n'), ylabel('x[n]'), title(['Nd = ',int2str(Nd)]);
axis([0 length(x)+Nd -inf inf])
n = 0:length(x1)-1;
subplot(3,1,2),stem(n,x1);
xlabel('n'), ylabel('x[n+Nd]');
axis([0 length(x)+Nd -inf inf])
n = 0:length(x2)-1;
subplot(3,1,3),stem(n,x2);
xlabel('n'), ylabel('x[n-Nd]');
```

```
axis([0 length(x)+Nd -inf inf])

%%
% alternative with negative time indices

N = 20;
Nd1 = 10;
Nd2 = -10;
Nx = N + max(abs([Nd1 Nd2]));
x = 0.9 .^ (0:N);
n = -Nx:Nx;                    % vector of time indices
n0 = Nx+1;                     % index of time origin (n = 0)
x0 = zeros(1,2*Nx+1);
x0(n0:n0+N) = x;
x1 = zeros(1,2*Nx+1);
x1(n0+Nd1:n0+Nd1+N) = x;
x2 = zeros(1,2*Nx+1);
x2(n0+Nd2:n0+Nd2+N) = x;
subplot(3,1,1),stem(n,x0);
ylabel('x[n]');
subplot(3,1,2),stem(n,x1);
ylabel('x[n-Nd1]'), title(['Nd1 = ',int2str(Nd1)]);
subplot(3,1,3),stem(n,x2);
xlabel('n'), ylabel('x[n-Nd2]'), title(['Nd2 = ',int2str(Nd2)]);

%%
% time reversal

x_rev = x(end:-1:1);           % perform time-inversion by reversing
                               % order of vector components
                               % Note: end = index of last x-component
n = 0:length(x)-1;
subplot(2,1,1),stem(n,x);
xlabel('n'), ylabel('x[n]'), title(['N = ',int2str(N)]);
subplot(2,1,2),stem(n,x_rev);
xlabel('n'), ylabel('x[N-n]')

%%
% creating periodic signals by repeating the fundamental signal part

N = 20;                        % period
n = 0:N-1;
x = 0.9 .^ n;                  % finite length signal (fundamental signal part)
np = 0:4*N-1;                  % time indices of periodic signal
xp = x(mod(np,N)+1);           % use modulo operation rem() to periodically
                               % extend signal x
subplot(2,1,1),stem(n,x);
xlabel('n'), ylabel('x[n]')
```

```
axis([0 length(xp) -inf inf])
subplot(2,1,2),stem(np,xp);
xlabel('n'), ylabel('x_p[n]')

%%
% cyclic shift

N = 20;                       % period
Nd = 10;                      % time-shift in samples
n = 0:N-1;
x = 0.9 .^ n;                 % finite length signal
np = 0:4*N-1;
xp = x(mod(np,N)+1);
%xp1 = x(mod(np+Nd,N)+1);     % cyclic time-shift by Nd samples
xp1 = xp(mod(np+Nd,N)+1);
subplot(2,1,1),stem(np,xp);
xlabel('n'), ylabel('x[n]')
subplot(2,1,2),stem(np,xp1);
xlabel('n'), ylabel('x[n-Nd]'), title(['Nd = ',int2str(Nd)]);

%%
% decimation used to plot a long speech signal

x = wavread('male_1ch_16kHz.wav');
Nx = 10000;                   % Nx samples of x will be used
M = 80;                       % decimation factor
Nb = floor(Nx/M);
Nx = M*Nb;                    % Nx is a multiple of M
X = zeros(M,Nb);
X(:) = x(1:Nx);               % columns of X contaim frames of length M
xmax = max(X);                % maxima of all columns of X
xmin = min(X);

close all
t = 0:Nx-1;
subplot(3,1,1), plot(t,x(1:Nx)), grid on;
xlabel('n'), ylabel('x[n]');
tb = 0:Nb-1;
subplot(3,1,2), plot(tb,x(1:M:Nx)), grid on;      % plot every Mth sample
xlabel('n (decimated)'), ylabel('x_d[n]');
title(sprintf('M = %d',M));
subplot(3,1,3),
hold on, area(tb,xmax), area(tb,xmin), hold off;  % plot max/min of frames
colormap([0 0.6 1]), grid on;
xlabel('n (decimated)'), ylabel('x_e[n]');

%%
% matrix representation of convolution
```

```
% (see also functions conv(), and filter())

close all;

Nx = 4;
Nh = 3;
x = ones(Nx,1);              % rectangular signal
h = ones(Nh,1);              % rectangular filter impulse response
Ny = Nx+Nh-1;                % length of convolution result
H = zeros(Ny,Nx);
for n = 1:Nx                 % build matrix
   H(n:n+Nh-1,n) = h;
end
y = H*x;                     % perform convolution
stem((0:Ny-1),y);
xlabel('n'), ylabel('y[n]');

%%
% alternative without for loop (looks very strange)

h1 = [zeros(Nx-1,1,1) ; h ; zeros(Nx-1,1)];  % add zeros to vector h
ic = (0:Ny-1)';
ir = Nx:-1:1;
H = ic(:,ones(Nx,1)) + ir(ones(Ny,1),:);     % matrix of indices where h1
                                             % is to be placed
H(:) = h1(H);

y = H*x;                     % perform convolution
stem((0:Ny-1),y);
xlabel('n'), ylabel('y[n]');

%%
% correlation function of a sinusoidal chirp
% (see also function xcorr())

Nx = 300;
n = 0:Nx-1;
x = sin(pi/3000*n.^2);
y = 1/Nx*conv(x,x(end:-1:1));
subplot(2,1,1),plot(n,x), grid on
xlabel('time index n'), ylabel('x[n]');
subplot(2,1,2),plot(y), grid on
xlabel('lag index k'), ylabel('R_{xx}[k]');

%%
% correlation function of Gaussian white noise

xn = randn(size(x));
```

```
yn = 1/Nx*conv(xn,xn(end:-1:1));
subplot(2,1,1),plot(n,xn), grid on
xlabel('time index n'), ylabel('x[n]');
subplot(2,1,2),plot(yn),grid on
xlabel('lag index k'), ylabel('R_{xx}[k]');
```

B Filter design examples

Depending on your MATLAB® version there are two ways to design digital filters. The first one makes use of the "Filter Design Toolbox" which is not included in the "Student Version of MATLAB". The second one only needs the "Signal Processing Toolbox" which is included in the "Student Version of MATLAB". Some fixed-point functions are used to demonstrate the influence of filter coefficient quantization. These functions may not be included in your MATLAB® version. However, it is easy to program the quantization functions with a few lines of MATLAB® code (see problem 5.14 on page 40).

B.1 Example using the Filter Design Toolbox

```
% bandpass filter specifications (as a spectral mask)
%                 --------
% Ap              --------
%                |        |
%
% As1_______|                |_______As2
%   0       fs1 fp1      fp2 fs2     Fs/2

clear all
clc

Fs = 16000;   % sampling frequency in Hz
fs1 = 1800;   % lower stopband frequency in Hz
fp1 = 2000;   % lower passband frequency in Hz
fp2 = 3000;   % upper passband frequency in Hz
fs2 = 3200;   % upper stopband frequency in Hz
As1 = 80;     % lower stopband attenuation in dB
Ap = 1.0;     % passband ripple in dB
As2 = 80;     % upper stopband attenuation in dB

% call filter design class constructor of bandpass filters

D = fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2',fs1,fp1,fp2,fs2,...
      As1,Ap,As1,Fs);

disp(D);            % show bandpass class properties
```

```
designmethods(D)    % show possible desing methods

%%
% first design an equiripple FIR filter
% Note: plot of impulse response, phase response etc. can be selected
% in plot window of fvtool

Hfir = design(D,'equiripple');

disp(Hfir);

close all;
fvtool(Hfir);       % filter visualization tool (plot magnitude response)

%%
close all;
impz(Hfir);         % plot impulse response

%%
close all;
grpdelay(Hfir);     % plot group delay (phase differentiated by frequency)

fprintf(1,'FIR filter length = %d\n\n', length(Hfir.Numerator));

%%
% repeat with elliptic IIR filter
% structure: cascade of 2nd order sections (direct form I)

Hiir1 = design(D,'ellip','FilterStructure','df1sos');

disp(Hiir1);

close all;
fvtool(Hiir1);

%%
Nimp = 400;         % number of impulse response samples
close all;
impz(Hiir1,Nimp);

%%
close all;
grpdelay(Hiir1);

fprintf(1,'number of 2nd order sections = %d\n\n', size(Hiir1.sosMatrix,1));

%%
% quantize FIR filter coefficients
```

```
Nw = 24;              % word length in bits
Hfirq = Hfir;
Hfirq.Arithmetic = 'fixed';
set(Hfirq, 'CoeffWordLength',Nw);

close all;
fvtool(Hfirq);

%%
% repeat with IIR filter

Nw = 8;               % word length in bits
Hiirq = Hiir1;
Hiirq.Arithmetic = 'fixed';
set(Hiirq, 'CoeffWordLength',Nw);
fvtool(Hiirq);

%%
% use direct form II 2nd order sections

Hiir2 = design(D,'ellip','FilterStructure','df2sos');
Nw = 12;               % word length in bits
Hiirq = Hiir2;
Hiirq.Arithmetic = 'fixed';
set(Hiirq, 'CoeffWordLength',Nw);
fvtool(Hiirq);
```

B.2 Example using the Signal Processing Toolbox

```
% bandpass filter specifications (as a spectral mask)
%                 --------
% Ap              --------
%                |        |
%
% As1_______|                |_______As2
%   0       fs1 fp1      fp2 fs2     Fs/2

clear all
clc

Fs = 16000;   % sampling frequency in Hz
fs1 = 1800;   % lower stopband frequency in Hz
fp1 = 2000;   % lower passband frequency in Hz
fp2 = 3000;   % upper passband frequency in Hz
fs2 = 3200;   % upper stopband frequency in Hz
```

```
As1 = 80;     % lower stopband attenuation in dB
Ap = 1.0;     % passband ripple in dB
As2 = 80;     % upper stopband attenuation in dB

% FIR filter design

% determine FIR filter order of equiripple design

delta_p = (10^(Ap/20)-1)/(10^(Ap/20)+1);  % passband ripple
delta_s1 = 10^(-As1/20);                  % stopband ripples
delta_s2 = 10^(-As2/20);
[N,f,a,w] = firpmord([fs1 fp1 fp2 fs2],[0 1 0],...
   [delta_s1 delta_p delta_s2],Fs);

% FIR design using Remez (Parks-McClellan algorithm)

h = firpm(N,f,a,w);    % FIR filter impulse response

Hfir = dfilt.dffir(h); % FIR filter structure
disp(Hfir);

close all;
fvtool(Hfir);
%fvtool(h,1);
%freqz(h,1);           % old-fashioned style

%%
% repeat with elliptic IIR filter

Fsh = Fs/2;
As = max(As1,As2);
[N,Wp] = ellipord([fp1/Fsh fp2/Fsh],[fs1/Fsh fs2/Fsh],Ap,As);

[b,a] = ellip(N,Ap,As,Wp);

% determine cascade form of 2nd order sections
% direct form I, norm-2 scaling, poles closest to unit circle at the end

[sos,g] = tf2sos(b,a,'up','two');

Hiir = dfilt.df1sos(sos,g);
disp(Hiir);

close all
fvtool(Hiir);

%%
% quantize FIR filter coefficients
```

```
% Note: quantization probably not supported without filter design toolbox

Nw = 12;             % word length in bits
Hfirq = Hfir;
Hfirq.Arithmetic = 'fixed';
set(Hfirq, 'CoeffWordLength',Nw);

close all;
fvtool(Hfirq);

%%
% repeat with IIR filter (cascade of 2nd order sections)

Nw = 12;              % word length in bits
Hiirq = Hiir;
Hiirq.Arithmetic = 'fixed';
set(Hiirq, 'CoeffWordLength',Nw);
fvtool(Hiirq);
```

C Spectrogram examples

Spectrograms are used to perform a localized spectral analysis of short-time stationary signals. A spectrogram can be implemented using the analysis stage of an FFT filter bank (see Section 3). The following MATLAB® program illustrates this kind of spectral analysis with various types of signals. In the lecture, we will present examples of the windowed Fourier transform (WFT) too. The listings of these experiments are not included here.

```
clc, close all;

% start with spectrogram of a sum of sinusoidal signals
% to show frequency localization

% create a sum of sinusoids
%        Nsin
% x[n] = sum  A_k*sin(2pi*f_k/Fs*n + phi_k)
%        k=1

Fs = 16000;
f = [800 1600 3000]';
A = [1 1 1]';
phi = [0 pi/4 pi/3]';

Nx = 2000;
n = 0:Nx-1;
X = A(:,ones(1,Nx)).*sin(2*pi/Fs*(f*n)+phi(:,ones(1,Nx)));
x = sum(X);
x = x(:);
```

```
Nwin = 160;           % no window effect
%Nwin = 128;          % shows window effect
Mov = Nwin/2;         % overlapping of frames
Nfft = Nwin;          % FFT length
%w = hamming(Nwin);   % removes blocking effects
w = rectwin(Nwin);    % shows blocking effects (if Nwin not a multiple
                      % of period length)

[Xs,f,t] = spectrogram(x,w,Mov,Nfft,Fs);  % compute spectrogram

close all;
pos = [0.2 0.4 0.6 0.6];
figure('Units','normal','Position',pos);

subplot(2,1,1), plot(n/Fs,x), grid on;
xlabel('time in sec.'), ylabel('x[n]');
title('sum of sinusoids');
subplot(2,1,2),imagesc(t,f,abs(Xs)), colormap(jet), colorbar;
set(gca,'YDir','normal');
xlabel('time in sec.'), ylabel('frequency in Hz');
title(...
sprintf('mag. spectrogram (lin. scale), Nwin = %d, Nfft = %d, Mov = %d',...
   Nwin,Nfft,Mov));

%%
%
% spectrogram of 1-impulses to show time localization

Fs = 16000;
Nx = 2000;
n = 0:Nx-1;
ni = [200 400 500];  % position of 1-impulses
x = zeros(1,Nx);
for k = 1:length(ni)
   x = x + double(n == ni(k));
end
Nwin = 16;           % small window for high time resolution
Mov = Nwin/2;
Nfft = 256;
%w = hamming(Nwin);   % influence on spectral amplitudes
w = rectwin(Nwin);   % rectangular window does not modify impulses
[Xs,f,t] = spectrogram(x,w,Mov,Nfft,Fs);

close all;
pos = [0.2 0.4 0.6 0.6];
figure('Units','normal','Position',pos);
```

```
subplot(2,1,1), plot(n/Fs,x), grid on;
xlabel('time in sec.'), ylabel('x[n]');
title('sum of 1-impules');
subplot(2,1,2), imagesc(t,f,abs(Xs)), colormap(jet), colorbar;
set(gca,'YDir','normal');
xlabel('time in sec.'), ylabel('frequency in Hz');
title(...
sprintf('mag. spectrogram (lin. scale), Nwin = %d, Nfft = %d, Mov = %d',...
   Nwin,Nfft,Mov));

%%
%
% spectogram of modulated gaussian impulses (i.e. impulses shifted both
% in time and frequency)

Fs = 16000;
Nx = 2000;
n = 0:Nx-1;
ni = [200 500 1000 1600 1600];     % positions of impulses
fi = [1000 2000 3000 5000 3000];   % frequencies of impulses
si = [30 60 80 100 200];           % width of impulses
x = zeros(1,Nx);
j = 1i;                  % j = sqrt(-1)
for k = 1:length(ni)
   x = x + real(exp(-((n-ni(k))/si(k)).^2+j*2*pi*fi(k)/Fs*n));
end

Nwin = 32;
Mov = Nwin/2;
Nfft = 256;
w = hamming(Nwin);       % avoids side lobes
%w = rectwin(Nwin);

[Xs,f,t] = spectrogram(x,w,Mov,Nfft,Fs);  % compute spectrogram

close all;
pos = [0.2 0.4 0.6 0.6];
figure('Units','normal','Position',pos);

subplot(2,1,1), plot(n/Fs,x), grid on;
xlabel('time in sec.'), ylabel('x[n]');
title('sum of modulated gaussian impulses');
subplot(2,1,2), imagesc(t,f,abs(Xs)), colormap(jet), colorbar;
set(gca,'YDir','normal');
xlabel('time in sec.'), ylabel('frequency in Hz');
title(...
sprintf('mag. spectrogram (lin. scale), Nwin = %d, Nfft = %d, Mov = %d',...
   Nwin,Nfft,Mov));
```

```
%%
%
% 1. spectrogram of a chirp (linearly modulated sinusoidal signal)
% 2. spectrogram of sinusoidal PM/FM
% demonstrate aliasing

Fs = 16000;
Nx = 1000;
n = 0:Nx-1;
mode = 1;
if mode == 1            % chirp
   alpha = 0.5;         % modulation parameter (no aliasing with 0<alpha<=0.5)
   %alpha = 1;           % aliasing
   x = sin(alpha*pi/Nx*n.^2);
   tstr = 'sinusoidal chirp signal (lin. sweep)';
else                    % sinusoidal PM/FM
   f0 = 3000;
   fm = 50;
   dphi = 20;
   x = sin(2*pi*f0/Fs*n+dphi*cos(2*pi*fm/Fs*n));
   tstr = 'sinusoidal phase/frequency modulation';
end

Nwin = 64;
Mov = ceil(3*Nwin/4);
Nfft = 256;
w = hamming(Nwin);      % avoids side lobes
%w = rectwin(Nwin);

[Xs,f,t] = spectrogram(x,w,Mov,Nfft,Fs);  % compute spectrogram

close all;
pos = [0.2 0.4 0.6 0.6];
figure('Units','normal','Position',pos);

subplot(2,1,1), plot(n/Fs,x), grid on;
xlabel('time in sec.'), ylabel('x[n]');
title(tstr);
subplot(2,1,2), imagesc(t,f,abs(Xs)), colormap(jet), colorbar;
set(gca,'YDir','normal');
xlabel('time in sec.'), ylabel('frequency in Hz');
title(...
sprintf('mag. spectrogram (lin. scale), Nwin = %d, Nfft = %d, Mov = %d',...
   Nwin,Nfft,Mov));

%%
%
```

```
% spectrogram of a bird song (blackbird)

[x,Fs] = wavread('blackbird.wav');
Nx = length(x);
n = 0:Nx-1;

Nwin = 1024;            % optimum value regarding time/frequency resolution
Mov = ceil(3*Nwin/4);
Nfft = 2*Nwin;
w = hamming(Nwin);

[Xs,f,t] = spectrogram(x,w,Mov,Nfft,Fs);  % compute spectrogram

fmax = 4000;            % limit frequency scale to [0,fmax]
nmax = round(fmax/Fs*Nfft);
f = f(1:nmax);
Xs = Xs(1:nmax,:);
close all;
pos = [0.2 0.4 0.6 0.6];
figure('Units','normal','Position',pos);

subplot(2,1,1), plot(n/Fs,x), grid on;
xlabel('time in sec.'), ylabel('x[n]');
title('blackbird song');

% Note: use image() instead of imagesc() to avoid scaling to full colormap

subplot(2,1,2), image(t,f,abs(Xs)), colormap(jet), colorbar;
set(gca,'YDir','normal');
xlabel('time in sec.'), ylabel('frequency in Hz');
title(...
sprintf('mag. spectrogram (lin. scale), Nwin = %d, Nfft = %d, Mov = %d',...
   Nwin,Nfft,Mov));

%%
%
% spectrogram of a speech signal

[x,Fs] = wavread('female_1ch_16kHz.wav');
Nx = length(x);
n = 0:Nx-1;

Nwin = 1024;            % optimum value regarding time/frequency resolution
Mov = ceil(3*Nwin/4);
Nfft = 2*Nwin;
w = hamming(Nwin);

[Xs,f,t] = spectrogram(x,w,Mov,Nfft,Fs);  % compute spectrogram
```

```
Xmag = abs(Xs);
Xmag = Xmag/max(max(Xmag));
Xmag = max(20*log10(Xmag),-60);   % scale spectrogram to [0,-60dB]

close all;
pos = [0.2 0.4 0.6 0.6];
figure('Units','normal','Position',pos);

subplot(2,1,1), plot(n/Fs,x), grid on;
xlabel('time in sec.'), ylabel('x[n]');
title('female speech signal');

% Note use image() instead of imagesc() to avoid scaling to full colormap

subplot(2,1,2), imagesc(t,f,Xmag), colormap(jet), colorbar;
set(gca,'YDir','normal');
xlabel('time in sec.'), ylabel('frequency in Hz');
title(sprintf('mag. spectrogram (in dB), Nwin = %d, Nfft = %d, Mov = %d',...
   Nwin,Nfft,Mov));
```

D Wavelet transform examples

There are two MATLAB® examples showing wavelet analysis, and signal denoising using wavelets. Both programs are implemented with the "Wavelet Toolbox of MATLAB". However, the examples also contain code segments using functions written by the author. Examples of the continuous wavelet transform (CWT) will be presented during the lecture. The source code of these experiments is not listed here.

D.1 Signal anaylsis with wavelets

```
% select and show wavelet filters

%wltype = 'db8';
%wltype = 'db45';
%wltype = 'sym8';
%wltype = 'sym12';
wltype = 'coif4';
%wltype = 'dmey';

[h_lp,h_hp] = wfilters(wltype);   % requires MATLAB wavelet toolbox
Nh = length(h_lp);
n = 0:Nh-1;
Fs = 16000;

close all;
figure('name','time domain','Units','normal',...
```

```
   'Position',[0.5,0.5,0.49,0.45]);

subplot(2,1,1), stem(n,h_lp);
xlabel('n'), ylabel('h_{lp}[n]');
title(sprintf('wavelet = %s',wltype));
subplot(2,1,2), stem(n,h_hp,'r');
xlabel('n'), ylabel('h_{hp}[n]');
figure('name','frequency domain','Units','normal',...
   'Position',[0.005,0.5,0.49,0.45]);
Nfft = 16*Nh;
Nfh = Nfft/2+1;
H_lp = fft(h_lp,Nfft);
H_hp = fft(h_hp,Nfft);
f = linspace(0,Fs/2,Nfh);
plot(f,20*log10(abs(H_lp(1:Nfh))),'b',f,20*log10(abs(H_hp(1:Nfh))),'r');
grid on, xlabel('f in Hz'), ylabel('magnitude in dB');

%%
% select type of test signal

sigtype = 'blocks';
%sigtype = 'bumps';
%sigtype = 'doppler';

Nx = 2000;
x = signals(sigtype,Nx);

Nlevels = 3;   % levels of wavelet tree
alg = 'w';    % use wavelets
%alg = 'b';    % use best basis wavelet packets
%alg = 'f';    % use full binary tree

wpspec(x,Nlevels,wltype,alg);

%%
% repeat with functions written by the author
% (MATLAB wavelet toolbox is not required)

% design wavelet low pass and high pass filters
% (do be used istead of wavelet functions)

h = qmf_2(50,0.65,0.1);

%%
wpspec(x,Nlevels,h,alg);
```

D.2 Signal denoising with wavelets

```
SNR = 10;      % SNR in dB of noisy signal

% select type of test signal

sigtype = 'blocks';
%sigtype = 'bumps';
%sigtype = 'doppler';

Nx = 2000;
xc = signals(sigtype,Nx);

% add white Gaussian noise

gain = 10^(-SNR/20);
w = randn(Nx,1);
x = xc + gain/std(w)*w;

% denoise using different threshold rules supported by function wden()

Nlevels = 4;    % number of wavlet tree levels

%th_rule = 'heursure';
%th_rule = 'rigrsure';
th_rule = 'sqtwolog';
%th_rule = 'minimaxi';

hardsoft = 's';
%hardsoft = 'h';

xd = wden(x,th_rule,hardsoft,'one',Nlevels,'sym8');

% show all signals

close all
upperright = [0.55,0.3,0.44,0.65];
figure('name','Denoising with wavelets','Units','normal','Position',upperright);

ymax = max([abs(xc) ; abs(x)]);
subplot(3,1,1), plot(xc), grid on;
axis([0 Nx-1 -ymax ymax]);
xlabel('n'), ylabel('x_c[n]')
title(sprintf('clean signal = %s',sigtype'));
subplot(3,1,2), plot(x), grid on;
axis([0 Nx-1 -ymax ymax]);
xlabel('n'), ylabel('x_n[n]')
title(sprintf('noisy signal, SNR = %3.2f dB', SNR))
```

```
subplot(3,1,3), plot(xd), grid on;
axis([0 Nx-1 -ymax ymax]);
xlabel('n'), ylabel('x_d[n]')
title(sprintf('denoised signal, threshold rule = %s', th_rule));

%%
% repeat with functions written by the author
% (MATLAB wavelet toolbox is not required)

% design wavelet low pass and high pass filters
% (do be used instead of wavelet functions)

h = qmf_2(50,0.65,0.1);

%%
% denoise using global threshold algorithm, and wavelet packet function

alg = 'w';    % use wavelets
%alg = 'b';    % use best basis wavelet packets
%alg = 'f';    % use full binary tree
hardsoft = 's';
%hardsoft = 'h';

wpdn_fir(xc,Nx,0,0,SNR,Nlevels,h,alg,hardsoft);

%%
% show wavelet spectrum

D = wpspec(xc,Nlevels,h,alg);
```

E MPEG-1 audio encoding

In this course, we will also present the principal operation of MPEG-1 audio encoding. An extended version of this coding scheme is used to process MP3 audio files. The slides of this presentation are available in the archive `Doblinger_mpeg_audio.zip` which includes a MATLAB® program to illustrate the most prominent features of MPEG-1 audio encoding.

F How to structure a MATLAB® project?

Most of the problems of this course can be solved by writing a single MATLAB® script or function. However, the more advanced problems and course projects usually need a set of programs stored in different files. Thus, organizing the program development of more demanding signal processing applications is an important issue before writing any lines of MATLAB® code. This section contains a short summary of available methods.

When starting a MATLAB® projects, functions can be organized as follows:

a) All functions in different m-files in the same directory,

b) A main function (also called primary function) in an m-file containing subfunctions,

c) A main function in the parent directory, and other functions in a subdirectory called `private`,

d) Functions organized as above plus class definitions in class directories starting with character `@` like `@myfirstclass`,

e) Use of packages with a package parent directory name starting with character `+` like `+mypackage`.

Method A

We create a directory and put all functions used by the project into this directory. Very simple but function names must not be coincide with other functions. If you do not know that e.g. function `sin` exists, then try `which sin` in the MATLAB® command window. This method should be used for small projects.

Method B

We create an m-File with the function name and put all functions (called subfunctions) in the m-File after the primary function (i.e. the first function in the file).

Example of an m-File named `myfunction.m`:

```
function y = myfunction(a,b,c)
% this is the primary function
% ...
y = mysubfunction1(a);   % call of a subfuntion
% ...
end   % end of myfunction (can be omitted)

function y1 = mysubfunction1(a)
% ...
end   % end of mysubfunction1 (can be omitted)

function y2 = mysubfunction2(a,b)
% ...
end   % end of mysubfunction2 (can be omitted)
```

Method C

We create a directory containing the primary function and other functions used by the project. Additional functions which are needed by all functions of the project are put into a subdirectory called `private`. Files in this private directory are known to project functions in the parent directory only.

Example directory structure:

```
myproject
    |_ mainfunction.m
    |_ function1.m
    |_ private
          |_ function2.m
          |_ function3.m
```

Functions `function2` and `function3` may be used in `mainfunction` and `function1` but not outside of directory `myproject`.

Method D

We apply one of the above methods and additionally use classes of object-oriented programming as discussed in appendix section G.

Example directory structure:

```
myproject
    |_ mainfunction.m
    |_ function1.m
    |_ private
          |_ function2.m
          |_ function3.m
    |_ @class1
          |_ class1.m
    |_ @class2
          |_ class2.m
```

Method E

MATLAB® packages may be used with large projects where we want to encapsulate all files and classes to a self-contained application. Function and class names need to be unique within the package only. Functions and classes defined within a package can also be used outside the package. The (parent) package directory must be included in the MATLAB® path. Package directory names are starting with character `+`.

Example directory structure:

```
+mypackage
    |_ mainfunction.m
    |_ function1.m
    |_ private
          |_ function2.m
          |_ function3.m
    |_ @class1
          |_ class1.m
    |_ @class2
          |_ class2.m
    |_ +subpackage1
          |_ ...
```

Functions used inside and/or outside the package directory must be referenced by the package name like `y = mypackage.mainfunction(x)`. The same holds with classes as in the example `obj = mypackage.class1(a,b,c)`.

As an alternative, we can import package functions and classes:

Example of package class import within a function:

```
function y = function1(x)
   import mypackage.class1

   obj = class1(a,b,c);  % call constructor of imported class
   ...
end
```

Example of package function import within a function:

```
function y = mainfunction(x,a,b,c)
   import mypackage.function1

   y = function1(x);     % call imported function
   ...
end
```

G Object-oriented programming in MATLAB®

In this section we give introductory examples of using object-oriented programming features of MATLAB® . A tutorial on this topic and a comprehensive description can be found in the MATLAB® User's Guide available in the built-in documentation. We will focus here only on defining classes, and on overloading built-in MATLAB® functions to process class objects. As an example, we can extend the + operation to add not only scalars or matrices but also certain elements of a class object. MATLAB® selects the proper operation by inspecting the operand types.

Example:

In this example, we define a class named `sig` of signals and process class objects with overloaded operation +, and overloaded function `plot()`. First, in the working directory we create a subdirectory `@sig` which contains file `sig.m`. This file defines the class and associated methods applied to objects of this class:

```
classdef sig
   properties
      x           % signal data (must be a vector)
      Fs          % sampling frequency in Hz
      name        % signal name
   end
   methods
      function ob = sig(x1,Fs,name)
         % signal class constructor
         % must be called before any use of signal class

         ob.x = x1;
         ob.Fs = Fs;
         ob.name = name;
      end
      function sum = plus(s1,s2)
         % overload + operator for adding signal class objects

         if s1.Fs ~= s2.Fs
            error('signals must have same sampling rate');
         elseif (min(size(s1.x)) || min(size(s2.x))) ~= 1
            error('signals must be vectors');
         end
         Fs_sum = s1.Fs;
         name_sum = 'sum of signals';
         N1 = length(s1.x);
         N2 = length(s2.x);
         Nsum = max(N1,N2);
         if N1 < Nsum
            fprintf(1,'zeropadding %s\n',s1.name);
            y = [s1.x(:) ;zeros(Nsum-N1,1)] + s2.x(:);
         elseif N2 < Nsum
            fprintf(1,'zeropadding %s\n',s2.name);
            y = [s2.x(:) ;zeros(Nsum-N2,1)] + s1.x(:);
         else
            y = s1.x(:) + s2.x(:);
         end
         if size(s1.x,1) == 1
            y = y.';
         end
         sum = sig(y,Fs_sum,name_sum);  % create function return object
```

```
      end
      function plot(s)
         % time domain plot used with signal class

         t = (0:length(s.x)-1)/s.Fs;
         close all;
         pos = [0.01 0.4 0.5 0.5];
         figure('numbertitle','off','name',s.name,'Units',...
            'normal','Position',pos);
         plot(t,s.x), grid on;
         xlabel('sec.'), ylabel('amplitude');
         title(s.name);
      end
      function fftplot(s,Nfft)
         % plot spectrum magnitude of signal
         % Nfft = optional FFT length

         if nargin == 1
            Nfft = length(s.x);
         end
         X = fft(s.x,Nfft);
         Nfh = Nfft/2+1;
         f = linspace(0,s.Fs/2,Nfh);
         N = length(s.x);
         t = 1/s.Fs*(0:N-1);
         close all;
         pos = [0.01 0.4 0.5 0.5];
         figure('numbertitle','off','name',s.name,'Units',...
            'normal','Position',pos);
         subplot(2,1,1), plot(t,s.x), grid on;
         xlabel('sec.'), ylabel('amplitude');
         subplot(2,1,2), plot(f,abs(X(1:Nfh))), grid on;
         xlabel('Hz'), ylabel('spectral magnitude');
         title(sprintf('FFT length = %d',Nfft));
      end
   end
end
```

At the beginning of this example, we define the signal class `sig` consisting of 3 elements (`x`, `Fs`, and `name`). In section `methods`, a class constructor function must be defined first to initialize objects of class `sig`. Next, 2 overloaded functions are defined. First, the + operation is extended in function `plus()` to add signal class objects. It checks sampling rates, signal lengths, and adds the respective signal data. The second function modifies `plot()` to plot signal class objects in a user-defined mode including axis labels, and a plot title. Finally, function `fftplot()` is defined to plot signal objects in frequency domain.

In order to use our signal class we create a script `use_sig.m` in the working directory:

```
clear;                  % needed to remove an already defined signal class
```

```
Fs = 1000;                % sampling frequency in Hz
f1 = 100;                 % frequency of first sinusoid
Nx = 300;                 % number of samples
n = 0:Nx-1;
x1 = sin(2*pi*f1/Fs*n);  % create 2 sinusoidal signals
f2 = 150;
x2 = sin(2*pi*f2/Fs*n);

%%

s1 = sig(x1,Fs,'sin f = 100');   % call signal class constructor
s2 = sig(x2,Fs,'sin f = 150');

s3 = s1 + s2;             % adding signal class objects using overloaded +

close all;

plot(s3);                 % time domain signal plot

%%

fftplot(s3);              % frequency domain signal plot
```

The contents of `use_sig.m` as shown above is self-explaining. The previous basic example and a more advanced one which processes ECG data are available to students of this course. Unpack the files from archive `Doblinger_oop_Matlab.zip` and run script `use_sig.m`, or `use_ecg.m` within the MATLAB® editor.

H Additional MATLAB® scripts and functions

H.1 Using a timer in MATLAB®

A timer can be used to synchronize various events in a MATLAB® program. There are 2 examples provided in archive `Doblinger_use_timer.zip`. The first example performs averaging of a noisy sinusiod controlled by a timer. A periodic timer event is used to call a function showing an updated plot of the averaging process. Running function `sin_avg()` demonstrates the successive improvements in smoothing the noisy sinusoid both in time and in frequency domain.

In the second example, an audio signal is played and at the same time displayed in time and frequency domain. A cursor moves frame by frame over the signal display, and the respective animated spectrum is plotted. Cursor movement is synchronized to the sampling frequency used to play the signal. However, timing precision is limited by the resolution provided by the MATLAB® timer, and by the processing speed of graphical output. Thus, we cannot expect a precise synchronization at high sampling frequencies.