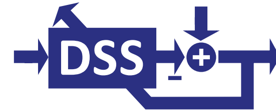
Robust and Efficient Digital Signal Processing



Gerhard Schmidt (Editor)

Digital Signal Processing and System Theory
Kiel University
Germany

2018

Version: **0.1** (January 2018)

Contents

I Basics 5

1 Fast Convolution 7

1.1 Problem . . . 7
1.2 References . . . 7
1.3 Code Examples . . . 7
1.4 Authors of this Chapter . . . 10

2 Recursive Norm Computation 11

2.1 Problem . . . 11
2.2 Recursive Computation . . . 12
2.3 Mixed Recursive/Iterative Computation . . . 13
2.4 References . . . 14
2.5 Code Examples . . . 15
2.6 Authors of this Chapter . . . 18

3 Levinson-Durbin Recursion 19

3.1 Problem . . . 19
3.2 References . . . 19
3.3 Code Examples . . . 19
3.4 Authors of this Chapter . . . 22

4 Prediction-based Filter Design 23

4.1 Basics . . . 23
4.2 Application Examples . . . 23
4.3 References . . . 25
4.4 Authors of this Chapter . . . 26

5 Complex Magnitude Approximations 27

5.1 Problem 27

5.2 A Very Simple Approximation 27

5.3 A Better Approximation 27

5.4 References 27

5.5 Authors of this Chapter 28

6 Removal of Signal Trends 29

6.1 Problem 29

6.2 A Simple Method to Remove a Signal Offset 30

6.3 Removal of Signal Trends by Highpass Filtering 30

6.4 A Non-linear and Time-variant Approach 33

6.5 Comparison of the Three Methods 36

6.6 References 36

6.7 Code Examples 37

6.8 Authors of this Chapter 42

Part I

Basics

Chapter 1

Fast Convolution Without Additional Delay

written by Anton Namenas, Seedo Eldho Paul, and Gerhard Schmidt

This chapter is about numerically robust ways for recursive norm computation. In contrast to iterative norm computations, which are numerically very accurate and robust, recursive approaches offer a large reduction in computational complexity. However, after several thousand iterations error accumulation appear. To avoid this a mixed iterative and recursive approach is proposed that is "cheap" in complexity and robust with respect to error accumulation.

Contents:

1.1	Problem	7
1.3	Code Examples	7
1.4	Authors	10

1.1 Problem

In several signal processing applications

1.2 References

[1] E. Hänsler, G. Schmidt: *Acoustic Echo and Noise Control*, Wiley, 2004.

1.3 Code Examples

Remark:

The following code example can be downloaded via the RED website.

```
%***************************************************************************
% Basic parameters
%***************************************************************************
N     = 588;                % Filter lenght
r     =  64;                % Frameshift
N_FFT = 128;                % FFT size

%***************************************************************************
% Input signal (white Gaussian noise)
%***************************************************************************
x = randn(5000,1);          % Input signal

%***************************************************************************
% Impulse response (white Gaussian noise)
%***************************************************************************
h = randn(N,1);             % Impulse response

%***************************************************************************
% Pure time-domain convolution
%***************************************************************************
y = conv(x,h);              % output signal

%***************************************************************************
% Mixed-domain convolution
%***************************************************************************

    %***********************************************************************
    % Initialization
    %***********************************************************************
    x_td_buffer     = zeros(N_FFT,1);
    h_td            = h(1:N_FFT);
    y_td            = zeros(size(x));
    y_fd_res_buffer = zeros(size(x));
    y_td_curr       = zeros(N_FFT,1);
    k_fd            = 0;
    M               = ceil((N-2*r)/r);
    X_fd_buffer     = zeros(N_FFT/2+1,M);
    H_fd_buffer     = zeros(N_FFT/2+1,M);

    %***********************************************************************
    % Fill the frequency-domain filter coefficients
    %***********************************************************************
    h_ind = N_FFT;

    % Loop over all frames
    for m = 1:M

        %*******************************************************************
        % Reset impulse response vector
        %*******************************************************************
        h_curr = zeros(r,1);

        %*******************************************************************
        % Fill the vector with the corresponding parts of the impulse res.
        %*******************************************************************
        for n = 1:r
            h_ind = h_ind + 1;
            if (h_ind <= N)
                h_curr(n) = h(h_ind);
            end;
        end;
```

```
    %***************************************************************
    % Compute FFT
    %***************************************************************
    H_curr          = fft(h_curr,N_FFT);
    H_fd_buffer(:,m) = H_curr(1:N_FFT/2+1);
end;

%*******************************************************************
% Main loop
%*******************************************************************
for k = 1:length(x)

    %***************************************************************
    % Update counters
    %***************************************************************
    k_fd = k_fd + 1;
    if (k_fd > r)
        k_fd = 1;
    end;

    %***************************************************************
    % Update the buffer
    %***************************************************************
    x_td_buffer(1:end-1) = x_td_buffer(2:end);
    x_td_buffer(end)     = x(k);

    %***************************************************************
    % Generate time-domain based part of the output
    %***************************************************************
    y_td(k) = x_td_buffer(end:-1:1)' * h_td + y_fd_res_buffer(k_fd);

    %***************************************************************
    % Start subsampled processing
    %***************************************************************
    if (k_fd == r)

       %************************************************************
       % Save the result of the previous background processing
       %************************************************************
       y_fd_res_buffer = y_td_curr(r+1:end);

       %************************************************************
       % Compute FFT on the input, if one full frame is available
       %************************************************************
       X_curr = fft(x_td_buffer,N_FFT);
       X_curr = X_curr(1:N_FFT/2+1);

       %************************************************************
       % Update the FFT buffer
       %************************************************************
       X_fd_buffer(:,2:M) = X_fd_buffer(:,1:M-1);
       X_fd_buffer(:,1)   = X_curr;

       %************************************************************
       % Compute the frequency-domain convolution output
       %************************************************************
       Y_fd = zeros(N_FFT/2+1,1);
       for m = 1:M
           Y_fd = Y_fd + X_fd_buffer(:,m) .* H_fd_buffer(:,m);
       end;
```

```
            %*****************************************************************
            % Compute inverse FFT of the output
            %*****************************************************************
            Y_fd_curr       = [Y_fd; conj(Y_fd(end-1:-1:2))];
            y_td_curr       = ifft(Y_fd_curr);

        end;
    end;

%****************************************************************************
% Show the result of both convolutions
%****************************************************************************
offset = 2;
lw     = 1;

figure(1);
plot(y(1:500),'b','LineWidth',lw);
hold on
plot(y_td(1:500)+offset,'r','LineWidth',lw);
grid on
hold off
legend('Output (time domain)', ...
       ['Output (mixed domain) + ',num2str(offset)]);
xlabel('Samples')
```

1.4 Authors of this Chapter

Anton Namenas received the B.Sc. and M.Sc. degrees from Kiel University, Germany, in 2015 and 2016, respectively. Since his B.Sc. graduation he works as a research assistant in the Digital Signal Processing and System Theory group at Kiel University. His research focus is on real-time simulation of acoustic environments and automatic evaluation of speech communication systems.

Seedo Eldho Paul obtained his Bachelor's degree in 2012 in Electronics and Communication Engineering from Mahatma Gandhi University, India. He worked for Wipro Ltd. from 2012 to 2015 as a medical embedded system developer. He is currently doing his Master in Digital Communications at Kiel University.

Gerhard Schmidt received the Dipl.-Ing. and Dr.-Ing. degrees from the Darmstadt University of Technology, Darmstadt, Germany, in 1996 and 2001, respectively. After the Dr.-Ing. degree, he worked in the research groups of the Acoustic Signal Processing Department, Harman/Becker Automotive Systems and at SVOX, Ulm, Germany. Parallel to his time at SVOX, he was a part-time Professor with the Darmstadt University of Technology. Since 2010, he has been a Full Professor with Kiel University, Germany. His main research interests include adaptive methods for speech, audio, underwater, and medical signal processing.

Chapter 2

Recursive Computation of Signal Vector Norms

written by Katharina Rebbe, Gerhard Schmidt, and Owe Wisch

This chapter is about numerically robust ways for recursive norm computation. In contrast to iterative norm computations, which are numerically very accurate and robust, recursive approaches offer a large reduction in computational complexity. However, after several thousand iterations error accumulation appear. To avoid this a mixed iterative and recursive approach is proposed that is "cheap" in complexity and robust with respect to error accumulation.

Contents:

2.1	Problem	11
2.2	Recursive Computation	12
2.3	Mixed Computation	13
2.4	References	14
2.5	Code Examples	15
2.6	Authors	18

2.1 Problem

In several signal processing applications signal vectors that contain the last N sample are utilized. Those vectors are usually defined as

$$\boldsymbol{x}(n) = \Big[x(n),\, x(n-1),\, x(n-2),\, ...,\, x(n-N+1)\Big]^{\mathrm{T}}. \tag{2.1}$$

Furthermore, some applications require to compute the squared norm of such vectors. A direct computation according to

$$\big\|\boldsymbol{x}(n)\big\|^2 = \sum_{i=0}^{N-1} x^2(n-i) \tag{2.2}$$

is numerically quite robust, but requires also N multiplications and $N-1$ additions every sample. In order to show the numerical robustness, we generated a signal that contains white Gaussian noise. Every 1000 samples

we varied the power by 20 dB (up and down in an alternating fashing). From that signal we extracted signal vectors of length $N = 128$ and computed the squared norm according to Eq. (2.2) in floating point precision, once with 64 bits and once with 32 bits and depict the results in a logarithmic fashion in Fig. 2.1. Additionally, the difference between the two versions is shown in the lowest diagram.

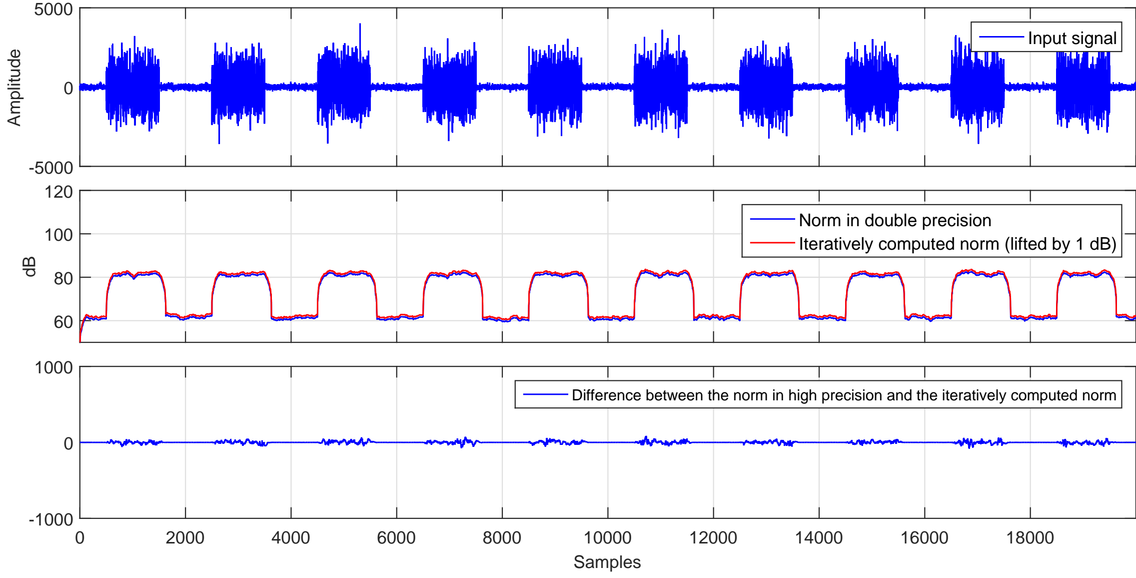


Figure 2.1: Input signal and iterative norm computations.

2.2 Recursive Computation

Remark:

The following ideas (and the solutions) can also be used for other recursive computations such as mean estimations $y(n) = \frac{1}{N} \sum_{i=0}^{N-1} x(n-i)$.

If the norm of the signal vector has to be computed every sample, often recursive computations are favoured since this lead to a significant reduction of computational complexity – especially for signal vectors with a large amount of elements [1]. The recursive variant starts with an initialization. Is is assumed that the signal vector contains zeros at time index $n = 0$ and thus also the squared norm is initialized with zero:

$$\boldsymbol{x}(0) = [0,\, 0,\, 0,\, ...,\, 0]^{\mathrm{T}}, \tag{2.3}$$

$$\|\boldsymbol{x}(0)\|^2 = 0. \tag{2.4}$$

Since with every sample only one new sample value is added to the signal vector and one sample value (the oldest) is leaving the vector, the norm can be computed recursively according to

$$\|\boldsymbol{x}(n)\|^2 = \|\boldsymbol{x}(n-1)\|^2 + x^2(n) - x^2(n-N). \tag{2.5}$$

Using this "trick" only two multiplications and two additions are required to update the norm. However, from a numerical point-of-view, this computation is not as robust as the direct approach according to Eq. (2.2).

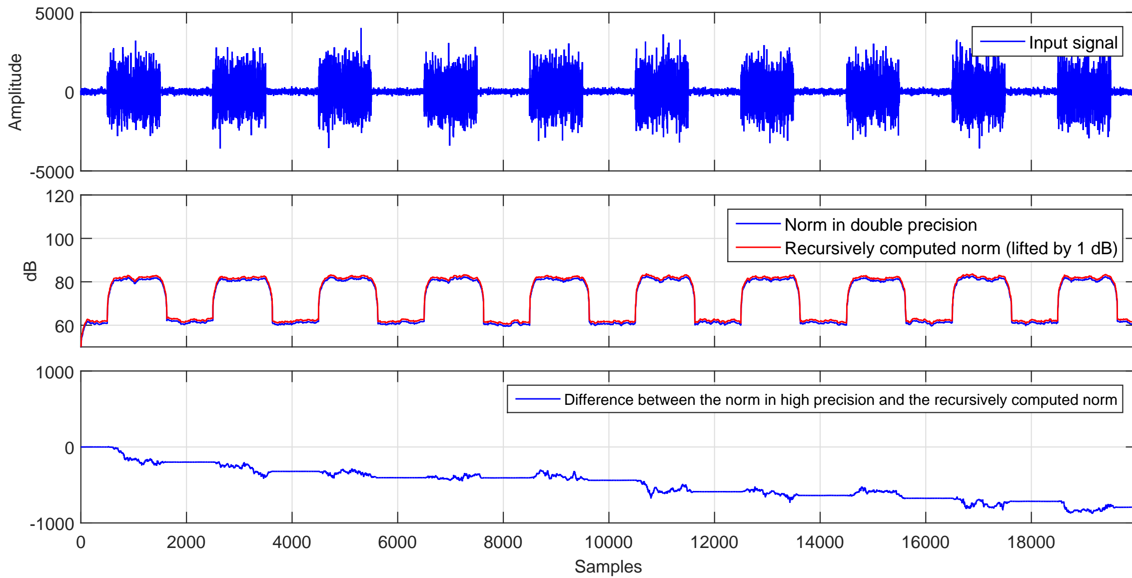


Figure 2.2: Input signal and recursive as well as iterative norm computations.

The problem with the recursive computation according to Eq. (2.5) is that a squared signal value $x^2(n)$ is used twice in the update rule:

- once when it enters the signal vector and
- once when it leaves the vector.

If the computation is done in floating-point arithmetic first the mantissa of the values that should be added or subtracted are adjusted (shifted) such that the exponents of both values are equal. This can be interpreted as some sort of quantization. If the norm value has changed between the "entering" and the "leaving" event, a small error occurs. Unfortunately, this error is biased and thus error accumulation appears.

Usually, this is not critical, because the error is really small, but if the signal has large power variations and the recursion is performed several thousand times, the small error might become rather large.

Due to that problem the norm might get negative, which leads – in some cases – to severe problems. E.g. several gradient based optimizations perform a division by the norm of the excitation vector. If the norm gets negative it means that the direction of the gradient is switched and divergence might be the result. This could be avoided by limiting the result of the recursive computation by the value 0:

$$\big\|\boldsymbol{x}(n)\big\|^2 = \max\Big\{0,\ \big\|\boldsymbol{x}(n-1)\big\|^2 + x^2(n) - x^2(n-N)\Big\}. \tag{2.6}$$

This improves robustness, but does not help against error accumulation as depicted in Fig. 2.2.

2.3 Mixed Recursive/Iterative Computation

A solution to this error accumulation problem is the extent the recursive computation according to Eq. (2.6) by an iterative approach that "refreshes" the recursive update from time to time. This can be realized by

adding in a separate variable $N_{\text{rec}}(n)$ all squared input samples:

$$N_{\text{rec}}(n) = \begin{cases} x^2(n), & \text{if} \quad \text{mod}\,(n,N) \equiv 0, \\ N_{\text{rec}}(n-1) + x^2(n), & \text{else.} \end{cases} \tag{2.7}$$

If N samples are added this variable is replacing to the recursively computed norm and the original sum $N_{\text{rec}}(n)$ is reinitialized with 0:

$$\|\boldsymbol{x}(n)\|^2 = \begin{cases} N_{\text{rec}}(n), & \text{if} \quad \text{mod}\,(n,N) \equiv N-1, \\ \max\left\{0,\, \|\boldsymbol{x}(n-1)\|^2 + x^2(n) - x^2(n-N)\right\}, & \text{else.} \end{cases} \tag{2.8}$$

The additional mechanism adds only a few additions, but helps a lot against error accumulation as indicated in the last example of this section depicted in Fig. 2.3. Thus, if you face problems with norms of signal vectors you might think about using this mixed method.

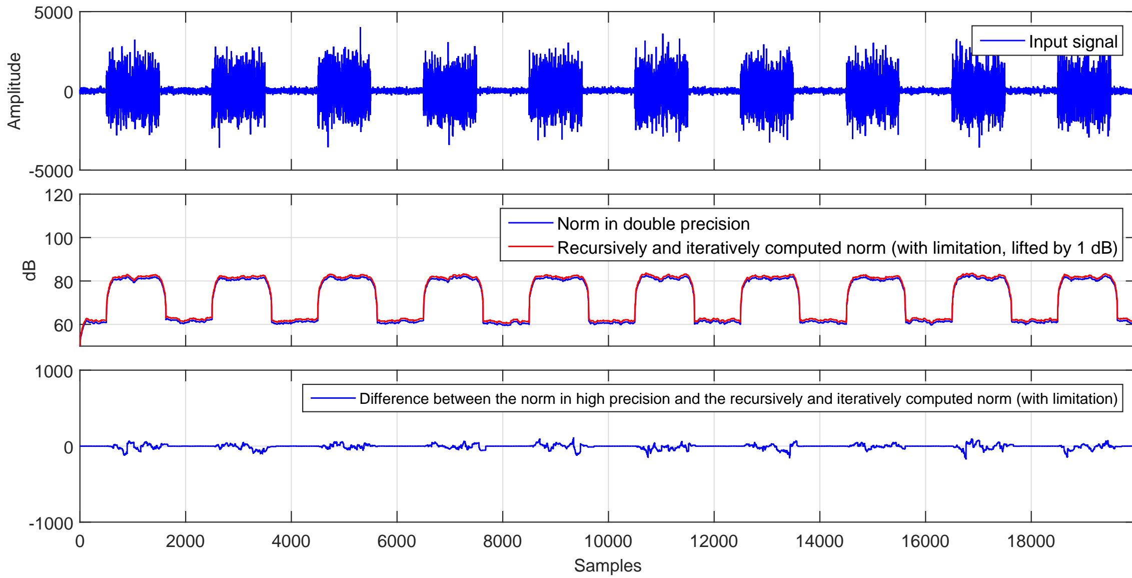


Figure 2.3: Input signal and mixed recursive/iterative as well as purely iterative norm computations.

2.4 References

[1] E. Hänsler, G. Schmidt: *Acoustic Echo and Noise Control*, Wiley, 2004.

2.5 Code Examples

Remark:

The following code example can be downloaded via the RED website.

```
%**************************************************************************
% Parameters
%**************************************************************************
N            =   128;
Sig_duration = 10000;

%**************************************************************************
% Generate input signal
%**************************************************************************

% Generate white Gaussian noise
sig = single(randn(Sig_duration,1));

% Boost every second 1000 signal values by 60 dB
for k = 1001:2000:Sig_duration;
    sig(k-1000:k) = sig(k-1000:k) * 1000;
end;

%**************************************************************************
% Compute norms
%**************************************************************************
x_vec                = single(zeros(N,1));
Norm_rec_curr        = single(0);
Norm_rec_curr_lim    = single(0);
N_rec                = single(0);
C_rec                = single(0);
Norm_mixed_curr      = single(0);
N_rec_lim            = single(0);
C_rec_lim            = single(0);
Norm_mixed_lim_curr = single(0);

Norm_double_prec     = zeros(Sig_duration,1);
Norm_iterative       = single(zeros(Sig_duration,1));
Norm_recursive       = single(zeros(Sig_duration,1));
Norm_recursive_lim   = single(zeros(Sig_duration,1));
Norm_mixed           = single(zeros(Sig_duration,1));
Norm_mixed_lim       = single(zeros(Sig_duration,1));

for k = 1:Sig_duration

    %**********************************************************************
    % Update signal vector (not very efficient, but o.k. for here
    %**********************************************************************
    x_new      = sig(k);
    x_old      = x_vec(N);
    x_vec(2:N) = x_vec(1:N-1);
    x_vec(1)   = x_new;

    %**********************************************************************
    % Norm in double precicion
    %**********************************************************************
    Norm_double_prec(k) = double(x_vec)' * double(x_vec);

    %**********************************************************************
    % Iterative norm (of course, there exixt optimized Matlab functions
    % for this purpose, but that's another "story")
    %**********************************************************************
    for n = 1:N
        Norm_iterative(k) = Norm_iterative(k) + x_vec(n)*x_vec(n);
```

```
    end;

    %**********************************************************************
    % Recursive norm computation
    %**********************************************************************
    Norm_rec_curr     = Norm_rec_curr + x_new^2 - x_old^2;
    Norm_recursive(k) = Norm_rec_curr;

    %**********************************************************************
    % Recursive norm computation
    %**********************************************************************
    Norm_rec_curr_lim     = Norm_rec_curr_lim + x_new^2 - x_old^2;
    Norm_rec_curr_lim     = max(0, Norm_rec_curr_lim);
    Norm_recursive_lim(k) = Norm_rec_curr_lim;

    %**********************************************************************
    % Mixed compuation of the norm
    %**********************************************************************
    Norm_mixed_curr = Norm_mixed_curr + x_new^2 - x_old^2;

    C_rec = C_rec + 1;
    if (C_rec == N)
        C_rec = 0;
    end;

    if (C_rec == 0)
        N_rec = 0;
    end;
    N_rec = N_rec + + x_new^2;

    if (C_rec == N-1)
        Norm_mixed_curr = N_rec;
    end;

    Norm_mixed(k) = Norm_mixed_curr;

    %**********************************************************************
    % Mixed compuation of the norm with limiation
    %**********************************************************************
    Norm_mixed_lim_curr = Norm_mixed_lim_curr + x_new^2 - x_old^2;
    Norm_mixed_lim_curr = max(0,Norm_mixed_lim_curr);

    C_rec_lim = C_rec_lim + 1;
    if (C_rec_lim == N)
        C_rec_lim = 0;
    end;

    if (C_rec_lim == 0)
        N_rec_lim = 0;
    end;
    N_rec_lim = N_rec_lim + + x_new^2;

    if (C_rec_lim == N-1)
        Norm_mixed_lim_curr = N_rec_lim;
    end;

    Norm_mixed_lim(k) = Norm_mixed_lim_curr;

end;

%**************************************************************************
```

```
% Show results
%***************************************************************************
fig = figure(1);
set(fig,'Units','Normalized');
set(fig,'Position',[0.1 0.1 0.8 0.8]);

t = 0:Sig_duration-1;

subplot('Position',[0.07 0.8 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_iterative))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Iteratively computed norm (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.62 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_recursive))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Recursively computed computed norm (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.44 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_recursive_lim))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Recursively computed computed norm with limitation (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.26 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_mixed))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Mixed recursively/iteratively computed norm (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.08 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_mixed_lim))+1,'r');
grid on
xlabel('Samples');
ylabel('dB')
legend('Norm in double precision', ...
       'Mixed recursively/iteratively computed norm with limitation (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);
```

2.6 Authors of this Chapter

Katharina Rebbe received the B.Sc. and M.Sc. degrees from Kiel University, Germany, in 2016 and 2017, respectively. Since her M.Sc. graduation she works as a development engineer.

Gerhard Schmidt received the Dipl.-Ing. and Dr.-Ing. degrees from the Darmstadt University of Technology, Darmstadt, Germany, in 1996 and 2001, respectively. After the Dr.-Ing. degree, he worked in the research groups of the Acoustic Signal Processing Department, Harman/Becker Automotive Systems and at SVOX, Ulm, Germany. Parallel to his time at SVOX, he was a part-time Professor with the Darmstadt University of Technology. Since 2010, he has been a Full Professor with Kiel University, Germany. His main research interests include adaptive methods for speech, audio, underwater, and medical signal processing.

Tim Owe Wisch received the B.Sc. and M.Sc. degrees from Kiel University, Germany, in 2015 and 2017, respectively. Since his M.Sc. graduation he works as a research assistant in the Digital Signal Processing and System Theory group at Kiel University. His research focus is on underwater communication and SONAR signal processing.

Chapter 3

Levinson-Durbin Recursion

written by Gerhard Schmidt

This chapter is about ...

Contents:

3.1	Problem	19
3.2	References	19
3.3	Code Examples	19
3.4	Authors	22

3.1 Problem

In several signal processing ...

3.2 References

[1] E. Hänsler, G. Schmidt: *Acoustic Echo and Noise Control*, Wiley, 2004.

3.3 Code Examples

Remark:

The following code example can be downloaded via the RED website.

```
%**************************************************************************
% Parameters
%**************************************************************************
N            =   128;
Sig_duration = 10000;

%**************************************************************************
% Generate input signal
%**************************************************************************

% Generate white Gaussian noise
sig = single(randn(Sig_duration,1));
```

```
% Boost every second 1000 signal values by 60 dB
for k = 1001:2000:Sig_duration;
    sig(k-1000:k) = sig(k-1000:k) * 1000;
end;

%***************************************************************************
% Compute norms
%***************************************************************************
x_vec              = single(zeros(N,1));
Norm_rec_curr      = single(0);
Norm_rec_curr_lim  = single(0);
N_rec              = single(0);
C_rec              = single(0);
Norm_mixed_curr    = single(0);
N_rec_lim          = single(0);
C_rec_lim          = single(0);
Norm_mixed_lim_curr = single(0);

Norm_double_prec    = zeros(Sig_duration,1);
Norm_iterative      = single(zeros(Sig_duration,1));
Norm_recursive      = single(zeros(Sig_duration,1));
Norm_recursive_lim  = single(zeros(Sig_duration,1));
Norm_mixed          = single(zeros(Sig_duration,1));
Norm_mixed_lim      = single(zeros(Sig_duration,1));

for k = 1:Sig_duration

    %***********************************************************************
    % Update signal vector (not very efficient, but o.k. for here
    %***********************************************************************
    x_new      = sig(k);
    x_old      = x_vec(N);
    x_vec(2:N) = x_vec(1:N-1);
    x_vec(1)   = x_new;

    %***********************************************************************
    % Norm in double precicion
    %***********************************************************************
    Norm_double_prec(k) = double(x_vec)' * double(x_vec);

    %***********************************************************************
    % Iterative norm (of course, there exixt optimized Matlab functions
    % for this purpose, but that's another "story")
    %***********************************************************************
    for n = 1:N
        Norm_iterative(k) = Norm_iterative(k) + x_vec(n)*x_vec(n);
    end;

    %***********************************************************************
    % Recursive norm computation
    %***********************************************************************
    Norm_rec_curr     = Norm_rec_curr + x_new^2 - x_old^2;
    Norm_recursive(k) = Norm_rec_curr;

    %***********************************************************************
    % Recursive norm computation
    %***********************************************************************
    Norm_rec_curr_lim     = Norm_rec_curr_lim + x_new^2 - x_old^2;
    Norm_rec_curr_lim     = max(0, Norm_rec_curr_lim);
    Norm_recursive_lim(k) = Norm_rec_curr_lim;
```

```
    %**********************************************************************
    % Mixed compuation of the norm
    %**********************************************************************
    Norm_mixed_curr = Norm_mixed_curr + x_new^2 - x_old^2;

    C_rec = C_rec + 1;
    if (C_rec == N)
        C_rec = 0;
    end;

    if (C_rec == 0)
        N_rec = 0;
    end;
    N_rec = N_rec + + x_new^2;

    if (C_rec == N-1)
        Norm_mixed_curr = N_rec;
    end;

    Norm_mixed(k) = Norm_mixed_curr;

    %**********************************************************************
    % Mixed compuation of the norm with limiation
    %**********************************************************************
    Norm_mixed_lim_curr = Norm_mixed_lim_curr + x_new^2 - x_old^2;
    Norm_mixed_lim_curr = max(0,Norm_mixed_lim_curr);

    C_rec_lim = C_rec_lim + 1;
    if (C_rec_lim == N)
        C_rec_lim = 0;
    end;

    if (C_rec_lim == 0)
        N_rec_lim = 0;
    end;
    N_rec_lim = N_rec_lim + + x_new^2;

    if (C_rec_lim == N-1)
        Norm_mixed_lim_curr = N_rec_lim;
    end;

    Norm_mixed_lim(k) = Norm_mixed_lim_curr;

end;

%**************************************************************************
% Show results
%**************************************************************************
fig = figure(1);
set(fig,'Units','Normalized');
set(fig,'Position',[0.1 0.1 0.8 0.8]);

t = 0:Sig_duration-1;

subplot('Position',[0.07 0.8 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_iterative))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
```

```
legend('Norm in double precision', ...
       'Iteratively computed norm (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.62 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_recursive))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Recursively computed computed norm (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.44 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_recursive_lim))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Recursively computed computed norm with limitation (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.26 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_mixed))+1,'r');
grid on
set(gca,'XTickLabel','');
ylabel('dB')
legend('Norm in double precision', ...
       'Mixed recursively/iteratively computed norm (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);

subplot('Position',[0.07 0.08 0.9 0.17]);
plot(t,10*log10(Norm_double_prec),'b', ...
     t,10*log10(max(0.01, Norm_mixed_lim))+1,'r');
grid on
xlabel('Samples');
ylabel('dB')
legend('Norm in double precision', ...
       'Mixed recursively/iteratively computed norm with limitation (lifted by 1 dB)');
axis([0 Sig_duration-1 0 120]);
```

3.4 Authors of this Chapter

Gerhard Schmidt received the Dipl.-Ing. and Dr.-Ing. degrees from the Darmstadt University of Technology, Darmstadt, Germany, in 1996 and 2001, respectively. After the Dr.-Ing. degree, he worked in the research groups of the Acoustic Signal Processing Department, Harman/Becker Automotive Systems and at SVOX, Ulm, Germany. Parallel to his time at SVOX, he was a part-time Professor with the Darmstadt University of Technology. Since 2010, he has been a Full Professor with Kiel University, Germany. His main research interests include adaptive methods for speech, audio, underwater, and medical signal processing.

Chapter 4

Prediction-based Filter Design

written by Gerhard Schmidt

In this chapter we will discuss how linear prediction can be used for designing filters with an arbitrary frequency response. The described design schemes can be used to implement real-time filter design applications that can work also on very simple hardware.

Contents:

4.1	Basics	23
4.2	Application Examples	23
4.3	References	25
4.4	Authors	26

4.1 Basics

In this chapter we will discuss how linear prediction can be used for designing filters with an arbitrary frequency response. Since linear predictors are used in a variety of applications (e.g. speech coding) various implementations exist, that solve the so-called normal equations in a robust and efficient manner. These schemes can be reused to implement real-time filter design applications that can work also on very simple hardware.

4.2 Application Examples

Prediction in general means to forecast signal samples that are not yet available (forward prediction) or to reestablish already forgotten samples (backward prediction). With this capability predictors play an important role in signal processing wherever it is desirable, for instance, to reduce the amount of data to be transmitted or stored. Examples for the use of predictors are encoders for speech or video signals.

However, linear prediction can also be used for several other applications:

Remark:

Before we start with the derivation of the filter design itself, the following applications should motivate the design process.

- **Loudspeaker equalization**

  To improve the playback quality of loudspeakers equalization filters might be placed before the DA converters of playback devices (see Fig. 4.1). These filters are designed such that the frequency re-

sponse of the system consisting of the loudspeaker itself and the equalization filter should be close to a predefined curve.

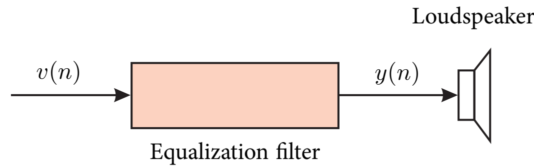


Figure 4.1: Basic structure of loudspeaker equalization schemes.

If more than one loudspeaker should be equalized often additional restrictions such as linear phase behaviour (constant group delay) are desired. Fig. 4.2 shows an example of such a desired frequency response together with a non-equalized loudspeaker and its equalized counterpart.

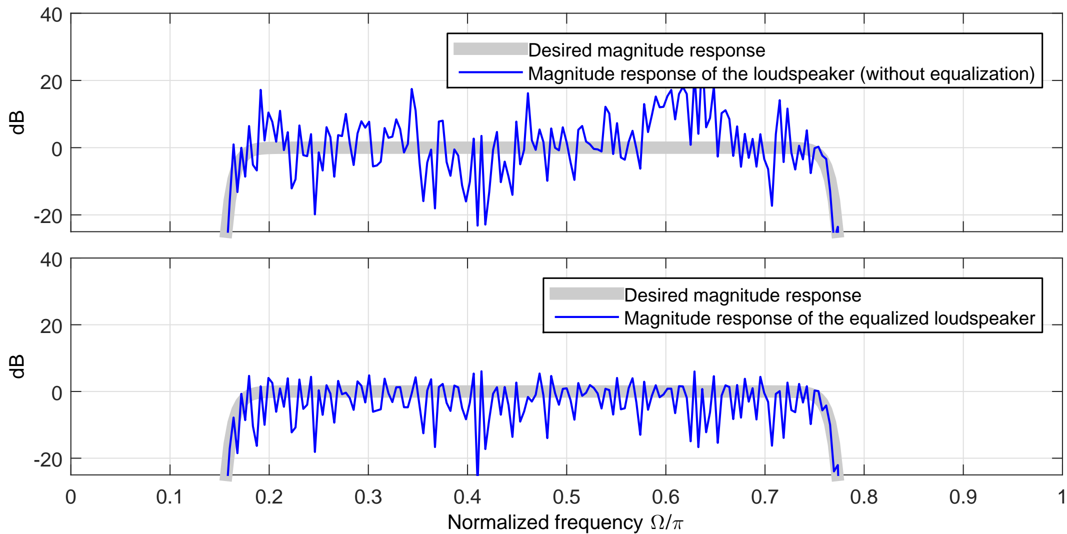


Figure 4.2: Magnitude responses of the non-equalized and the equalized loudspeaker.

- **Low-delay noise suppression**

  Whenever a desired signal is superimposed by noise signal enhancement techniques can be applied (see Fig. 4.3). Usually, statistically optimized, time-variant filters such as so-called *Wiener filters* [1] are utilized here.

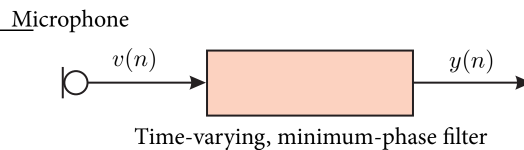


Figure 4.3: Low-delay noise suppression.

Those approaches are usually realized in the short-term Fourier domain. However, if the delay that is

inserted by the Fourier transforms is too large, time-domain approaches with low-order minimum-phase filters might be an alternative solution. The design of these filters can be prediction-based [2, 3]. Fig. 4.4 shows an example of a noisy speech signal (filter input) and the corresponding noise-reduced signal (filter output).

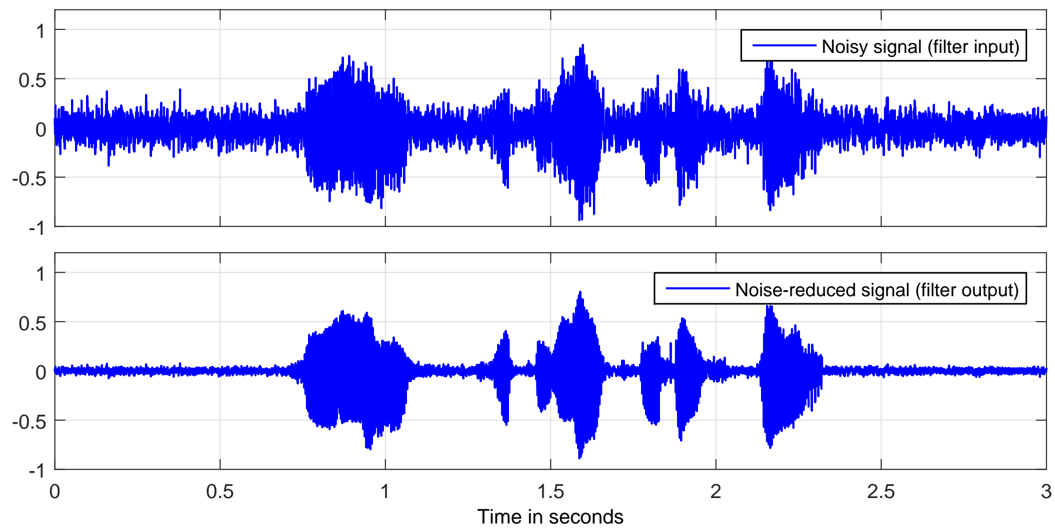


Figure 4.4: Signal before and after noise suppression.

- **Signal generation**

  As a last application so-called *general purpose noise or signal generators* can be mentioned. They are build usually by a white noise generator (either with Gaussian or uniform amplitude distribution) and a succeeding shaping filter for adjusting the power spectral density (PSD) of the output filter (see Fig. 4.5).

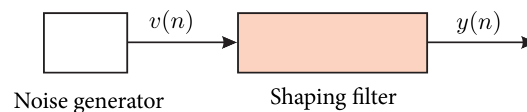


Figure 4.5: Signal generation.

Since the input PSD is constant (white noise) the shaping filter must be designed such that its frequency response (respectively the squared magnitude of it) is the same as the desired PSD. Fig. 4.6 shows an example of in input and output PSDs (in blue) together with the desired PSD (in grey).

4.3 References

[1] E. Hänsler, G. Schmidt: *Acoustic Echo and Noise Control*, Wiley, 2004.

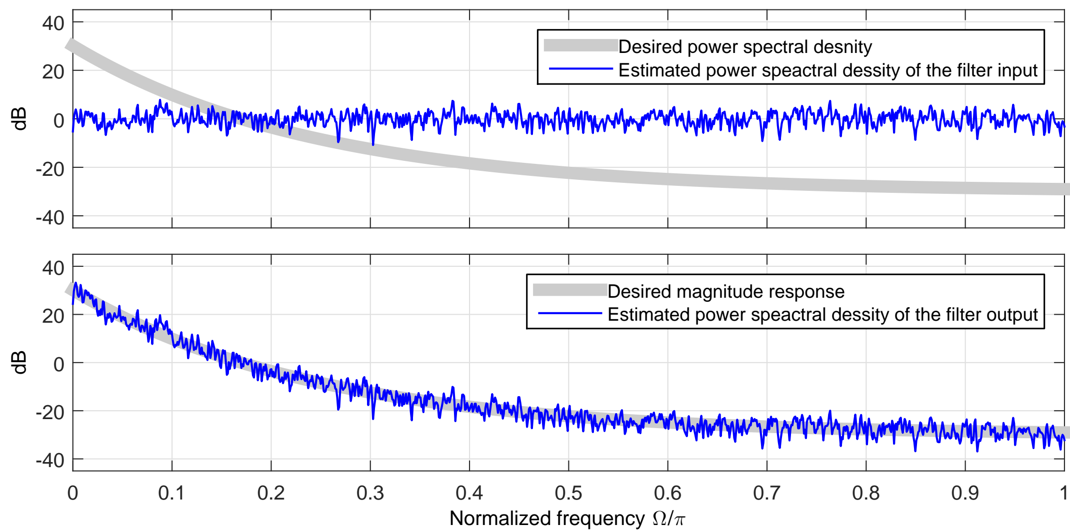


Figure 4.6: Power spectral density before and after the shaping filter.

[2] H. Löllmann, P. Vary: *Low Delay Filter for Adaptive Noise Reduction*, Proc. IWAENC '05, Eindhoven, The Netherlands, pp. 205 - 208, 2005.

[3] H. Löllmann, P. Vary: *A Filter Structure for Low Delay Noise Suppression*, Proc. ITG-Fachtagung '06, Kiel, Germany, 2006.

4.4 Authors of this Chapter

Gerhard Schmidt Text GUS

Chapter 5

Complex Magnitude Approximations

written by Gerhard Schmidt

This chapter starts with a brief introduction in telephony with special emphasis on hands-free systems. Secondly the main outline of this book is described and the notation used in the remaining chapters is explained.

Contents:

5.1	Problem	27
5.2	A Very Simple Approximation	27
5.3	A Better Approximation	27
5.4	References	27
5.5	Authors	28

5.1 Problem

- Need for magnitude instead of magnitude square values
- Complexity of square root computations

5.2 A Very Simple Approximation

Some text ...

5.3 A Better Approximation

Some text ...

5.4 References

[1] The New Bell Telephone, *Sci. Am.* **38**(1), 1(1877).

5.5 Authors of this Chapter

Gerhard Schmidt Text GUS

Chapter 6

Removal of Signal Trends

written by Christin Bald, Julia Kreisel, and Gerhard Schmidt

This chapter is about the removal of the offset – also referred to as *trend* – of a signal. Therefore three different methods are introduced and compared against each other: subtracting a priori knowledge, highpass filtering, and a nonlinear, time-variant method. The presented methods are numerically robust and computationally efficient. The performances of the methods are demonstrated by removing the trend of a magnetically measured heart signal.

Contents:

6.1	Problem	29
6.2	A Simple Method	30
6.3	Highpass Filtering	30
6.4	Non-linear, Time-variant Approach	33
6.5	Comparison	36
6.6	References	36
6.7	Code Examples	37
6.8	Authors	42

6.1 Problem

In several applications signals are recorded that contain an offset. Sometimes the offset carries information – in these cases this signal component should not be removed. However, in a variety of applications one is interested mainly in the temporal variations of the signal (and not in the offset). In these cases a simple (meaning computationally efficient) and robust offset or so-called *trend* removal can be applied. This allows follow-up signal processing to be a bit simpler, e.g. thresholds do not have to be adjusted to the offset.

> ***Remark:***
>
> The MCG signals depicted on the next page and used in the following examples can be downloaded as wav files from the RED website.

Examples for such signals are ECG or MCG signals, where ECG abbreviates electrocardiogram and MCG its magnetic counterpart, magnetocardiogram. Since the authors work with the analysis of both we will use MCG signals as an example. Fig. 6.1 shows two signals that were recorded at the same time but at different positions.

MCG signals show the same cardiac cycle as known from ECG signals. One heart cycle consists of a P wave, a QRS complex, and a T wave [2]. The first wave is the P wave in which the atria contraction is described. The QRS complex is the combination of the Q, R and S wave showing the beginning of the contraction of the ventricle. The start of the T wave describes the beginning of the relaxation. The offset means an almost

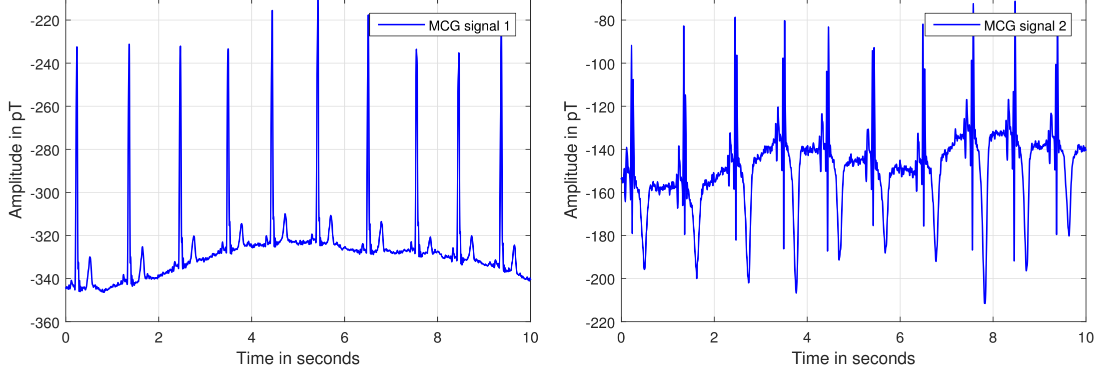


Figure 6.1: MCG (magnetocardiogram) input signals.

constant shifting from zero.

6.2 A Simple Method to Remove a Signal Offset

A very simple method to remove the trend in a signal is to know the trend a priori and to subtract this value. If the application allows for a so-called *pre-measurement*, it is rather simple to obtain an estimate for the mean of the signal (with or without the desired signal component). Assuming that we have this measure, we can obtain a simple trend estimate by just using the a priori knowledge:

$$x_{\text{trend,simple}}(n) \quad = \quad x_{\text{a prioi}}. \tag{6.1}$$

The trend compensated signal can be obtained by subtracting the estimated trend from the measured signal:

$$x_{\text{comp,simple}}(n) \quad = \quad x(n) - x_{\text{trend,simple}}(n). \tag{6.2}$$

To obtain a good estimate for $x_{\text{a prioi}}$ we have averaged the two input signals that are depicted in Fig. 6.1 for the entire length of both signals individually. The resulting values are then used as a priori knowledge and are subtracted from the input signals according to Eq. (6.2). Fig. 6.2 shows the resulting detrended signals (in the upper diagrams) as well as the estimated trends (red color) and the input signals (blue color) in the lower diagrams. Please note that only the first 10 seconds of the signals are depicted. The second signal gets a much smaller offset during the period of 10 to 30 seconds (compared to the first 10 seconds), leading to the depicted average that looks a bit too small at first glance.

6.3 Removal of Signal Trends by Highpass Filtering

A second method for trend removal is to estimate the mean of the signal by means of averaging the signal over the last N_{hp} samples:

$$x_{\text{trend,hp}}(n) \quad = \quad \frac{1}{N_{\text{hp}}} \sum_{i=0}^{N_{\text{hp}}-1} x(n-i). \tag{6.3}$$

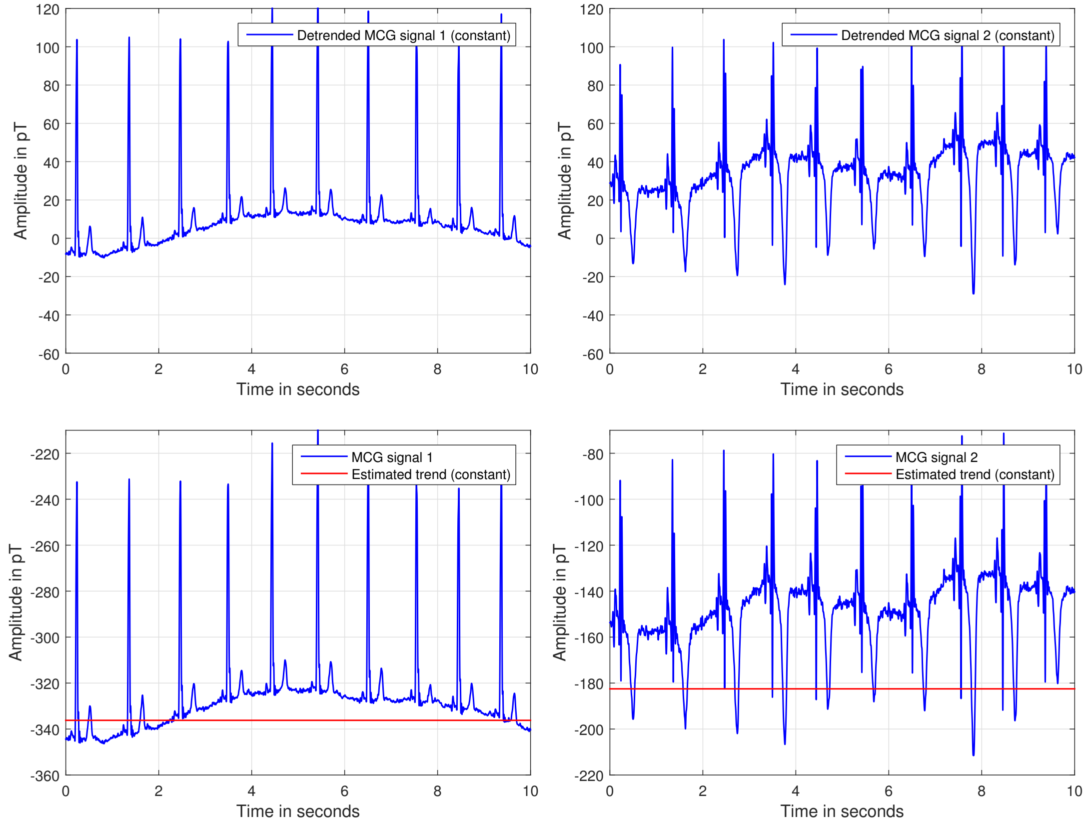


Figure 6.2: Input signals, estimated trends, and trend compensated signals using the method of constant subtraction.

The estimated mean value $x_{\text{trend,hp}}(n)$ is subtracted from the input signal

$$x_{\text{comp,hp}}(n) \quad = \quad x(n) - x_{\text{trend,hp}}(n), \tag{6.4}$$

resulting in the desired trend removal. While Eq. (6.3) describes a lowpass filter, the subtraction in Eq. (6.4) results in a highpass filter, which gives this section also its name. Both filters are related (in the frequency domain) as:

$$H_{\text{hp}}\big(e^{-j\Omega}\big) \quad = \quad 1 - H_{\text{lp}}\big(e^{-j\Omega}\big). \tag{6.5}$$

Remark:

The approach presented here uses only causal memory. If file-based processing is performed, the filter operation of Eq. (6.3) could also start at $i = -N_{\text{hp}}/2 - 1$ and end at $i = N_{\text{hp}}/2$.

The frequency response of the lowpass filter can be obtained by first having a closer look on Eq. (6.3) in the time domain. The equation can be interpreted as a convolution of the input signal with the lowpass impulse response $h_{\text{lp},n}$:

$$x_{\text{trend,hp}}(n) \quad = \quad \sum_{i=\infty}^{\infty} h_{\text{lp},i}\, x(n-i). \tag{6.6}$$

Comparing Eqs. (6.3) and (6.6) leads to the FIR (finite impulse response, [1, 3, 4]) system

$$h_{\mathrm{lp},i} = \begin{cases} \frac{1}{N_{\mathrm{hp}}}, & \text{if } 0 \le i < N_{\mathrm{hp}}, \\ 0, & \text{else.} \end{cases} \tag{6.7}$$

Taking also the subtraction of Eq. (6.4), which actually detrends the signal, into account leads to the impulse response of the high pass filter (again an FIR filter):

$$h_{\mathrm{hp},i} = \begin{cases} 1 - \frac{1}{N_{\mathrm{hp}}}, & \text{if } i = 0, \\ -\frac{1}{N_{\mathrm{hp}}}, & \text{if } 1 \le i < N_{\mathrm{hp}}, \\ 0, & \text{else.} \end{cases} \tag{6.8}$$

In Fig. 6.3 the magnitude responses of resulting lowpass (left side) and highpass filter (right side) are depicted. For a sample rate of $f_s = 1000\,\mathrm{Hz}$ a filter order $N_{\mathrm{hp}} = 2000$ was chosen, leading to an average based on the last two seconds of the signal.

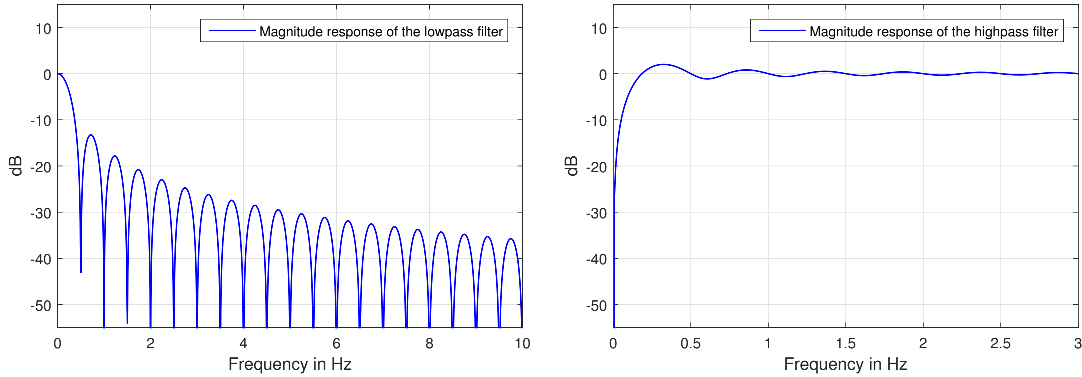


Figure 6.3: Magnitude responses of the lowpass and highpass filter for a filter order of $N_{\mathrm{hp}} = 2000$ and a sample rate of $f_s = 1000\,\mathrm{Hz}$.

The computation of the lowpass part of the trend removal is rather costly, since usually a few hundred or even thousand (as in our example) signals have to be added. Of course, one can speed up the process by performing the entire operation in the spectral domain using fast Fourier transforms. However, this would introduce delay due to the necessary framing. Another way that is able to save an even larger amount of complexity is to exploit the special choice of the filter coefficients (they are all the same) and transform the FIR filter into an IIR structure. This can be achieved by recursively computing the estimated trend according to

$$x_{\mathrm{trend,hp}}(n) = x_{\mathrm{trend,hp}}(n-1) + \frac{1}{N_{\mathrm{hp}}} \Big[x(n) - x(n - N_{\mathrm{hp}}) \Big]. \tag{6.9}$$

This requires only two additions and one multiplication per sample. Please note that the division by N_{hp} can be computed during initialization and the inverse value can be stored. This transforms the division into a

multiplication (at least for the main operation of the filter). In terms of memory nothing has changed with this *trick* – still N_{hp} samples have to be stored in a so-called *ringbuffer*.

Transforming specific FIR filter structures into equivalent IIR counterparts is not new. However, only a few authors mention the numerical problems that appear with this kind of processing.

When computing Eq. (6.9) on a processor with floating point precision, the mantissae of all terms that should be added are shifted until the exponents are all the same. When a sample is entering the memory this shifting operation is not necessarily the same as during the leaving event. As a consequence biased error accumulation appears. If the signal is rather short this is not really an issue. However, if a few thousand samples have to be processed this might lead to numerical problems.

A solution to this problem is rather simple. The recursive processing should be updated from time to time by an iteratively computed estimation. This could be achieved with a small extension to Eq. (6.9):

$$x_{\text{trend,hp}}(n) = \begin{cases} \dfrac{x_{\text{trend,reset}}(n)}{N_{\text{hp}}}, & \text{if} \quad \text{mod}\;(n, N_{\text{hp}}) \equiv N_{\text{hp}} - 1, \\ x_{\text{trend,hp}}(n-1) + \frac{1}{N_{\text{hp}}}\Big[x(n) - x(n - N_{\text{hp}})\Big], & \text{else}. \end{cases} \tag{6.10}$$

The so-called *reset value* can be computed according to

$$x_{\text{trend,reset}}(n) = \begin{cases} x(n), & \text{if} \quad \text{mod}\;(n, N_{\text{hp}}) \equiv 0, \\ x_{\text{trend,reset}}(n-1) + x(n), & \text{else}. \end{cases} \tag{6.11}$$

To show the results of this second method, the simulation of Fig. 6.2 has been repeated, but now with the highpass method. Fig. 6.4 shows in the upper diagrams the detrended results as well as the input signals (blue color) and the estimated trends (red color) in the lower diagrams.

6.4 A Non-linear and Time-variant Approach

As a last method we would like to introduce a non-linear, time-variant method. The method is nearly as simple as the highpass filter, but is usually a bit better, especially if the short-term mean of the signal is not zero. In addition a significant reduction of the required memory (compared to the highpass filter approach) is possible.

As a first step we define the global memory of the signal as N_{global} samples. In the following we will base our analyses on input signals up to that delay. Please note that it is not required to store the input signal for that amount of samples. As a second step we split the global memory into frames and define the framesize N_{frame} for our method. For the MCG example we could use about 1 second of global memory and a frame duration of about 100 ms. Since the data was sampled at $f_s = 1\,\text{kHz}$, we get

$$N_{\text{global}} = 1000, \tag{6.12}$$

$$N_{\text{frame}} = 100. \tag{6.13}$$

Furthermore, we assume that N_{global} is an integer multiple of N_{frame}

$$N_{\text{global}} = K_{\text{frame}}\, N_{\text{frame}}, \tag{6.14}$$

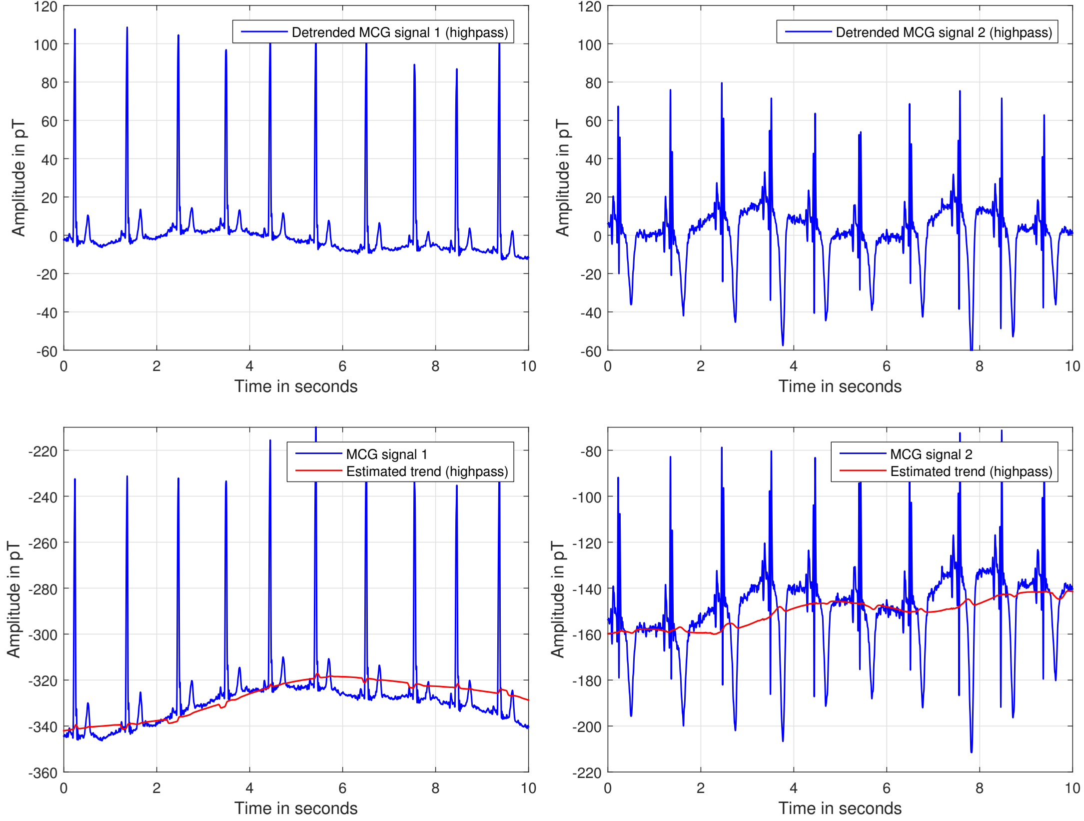


Figure 6.4: Input signals, estimated trends, and trend compensated signals using highpass filtering.

which is true in the example above ($K_{\text{frame}} = 10$). As a next step we compute in an iterative manner the mean for each frame according to

$$x_{\text{curr. mean}}(n) = \begin{cases} \frac{x(n)}{N_{\text{frame}}}, & \text{if } \operatorname{mod}(n, N_{\text{frame}}) \equiv 0, \\ x_{\text{curr. mean}}(n-1) + \frac{x(n)}{N_{\text{frame}}}, & \text{else.} \end{cases} \tag{6.15}$$

Please note that $x_{\text{curr. mean}}(n)$ only results in the correct mean if the condition

$$n = \lambda N_{\text{frame}} - 1, \tag{6.16}$$

is true (with $\lambda \in \mathcal{Z}$). To save computational complexity the division by N_{frame} can also be avoided for the sample-by-sample iteration – it should be performed only when the frame is completely *filled*. Beside the simple update according to Eq. (6.15) only one first-order recursive smoothing process and the subtraction of the estimated trend according to

$$x_{\text{comp,nl}}(n) = x(n) - x_{\text{trend,nl}}(n) \tag{6.17}$$

are computed at the high sample rate. All other computations are computed only once per N_{frame} samples.

It will lead to a trend estimate $\tilde{x}_{\text{trend,nl}}(n)$ that is available only if

$$\text{mod}\big(n, N_{\text{frame}}\big) \quad \equiv \quad N_{\text{frame}} - 1, \tag{6.18}$$

which is an equivalent condition to Eq. (6.16). If we would use the (subsamples) estimated trend directly in Eq. (6.17), sudden signal *jumps* might appear. For that reason, first order IIR smoothing is performed to avoid such artifacts:

$$x_{\text{trend,nl}}(n) = \beta_{\text{sm}}\, x_{\text{trend,nl}}(n-1) + (1-\beta_{\text{sm}})\, \tilde{x}_{\text{trend,nl}}\left(\left\lfloor \frac{n}{N_{\text{frame}}} \right\rfloor\right). \tag{6.19}$$

The symbols $\lfloor \ldots \rfloor$ should indicate rounding down. The smoothing parameter β_{sm} is chosen from the interval

$$0 \ll \beta_{\text{sm}} < 1. \tag{6.20}$$

Typically β_{sm} is chosen out of the range $[0.9,\, 0.9999]$ – depending on the sample rate f_s and on the frame size N_{frame}. If a frame is completely filled (according to condition (6.18) the short-term mean is added to a vector that contains the last K_{frame} short-term means

$$\boldsymbol{x}_{\text{mean}}(n) \quad = \quad \begin{cases} \Big[x_{\text{curr. mean}}\big(n\big),\, x_{\text{curr. mean}}\big(n - N_{\text{frame}}\big),\, \ldots,\, x_{\text{curr. mean}}\big(n - (K_{\text{frame}} - 1)N_{\text{frame}}\big)\Big]^{\text{T}}, \\ \qquad \text{if } \text{mod}(n, N_{\text{frame}}) \equiv N_{\text{frame}} - 1, \\ \boldsymbol{x}_{\text{mean}}(n-1), \\ \qquad \text{else.} \end{cases} \tag{6.21}$$

This method allows to store the supporting points of the averaged signal in a subsampled manner, meaning that only K_{frame} data words (in our example 10) are required to store information of N_{global} samples (in our example 1000). The basic idea to estimate the trend is now to sort the entries of the vector $\boldsymbol{x}_{\text{mean}}(n)$ and to utilize, e.g., the median of the stored short-term means. Beside the median also other quantiles could be utilized. However, the median usually should be the first choice. The vector containing the sorted mean values is denoted by

$$\boldsymbol{x}_{\text{mean, sorted}}(n) \quad = \quad \Big[x_{\text{mean, sorted, 0}}(n),\, x_{\text{mean, sorted, 1}}(n),\, \ldots,\, x_{\text{mean, sorted, } K_{\text{frame}} - 1}(n)\Big]^{\text{T}}. \tag{6.22}$$

The reason for using a sorting operation here (and not a second averaging stage) is that this allows also for a good trend estimation even if the signal might have a positive or negative bias (as it is the case in the first MCG example signal). Of course the literature is full of efficient sorting algorithms. However, since we can assume that we already have a sorted list available before a new short-term mean is computed we can use a two-stage procedure that updates the sorted list:

- As a first stage we copy the old sorted vector and the new short-term mean into an extended sorted vector. This operation can be performed with $K_{\text{frame}} + 1$ operations.
- In a second stage we copy the extended vector back into the original. Beside this copying operation we check if the element to be copied is equal to the short-term mean that leaves the vector $\boldsymbol{x}_{\text{mean}}(n)$. This element is then not copied resulting in a shortening of the extended vector. For this second part again $K_{\text{frame}} + 1$ operations are required.

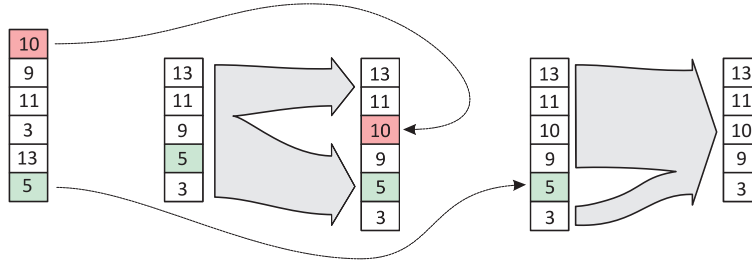


Figure 6.5: Sorting of the frame means.

As a consequence the entire sorting procedure (which is actually only an update of an already sorted list) required only about $2\,K_{\text{frame}}$ operations. Fig. 6.5 shows an overview about the two-stage sorting procedure. Every N_{frame} sample the (non-smoothed) trend estimate is updated according to

$$\tilde{x}_{\text{trend,nl}}(n) \quad = \quad \begin{cases} x_{\text{mean, sorted, }K_{\text{frame}}/2}(n), & \text{if } \text{mod}(n, N_{\text{frame}}) \equiv N_{\text{frame}} - 1, \\ \tilde{x}_{\text{trend,nl}}(n-1), & \text{else.} \end{cases} \tag{6.23}$$

As in the last sections we perform the detrending operation with the two MCG input signals and show the estimated trends together with the input signals as well as the detrended signals in Fig. 6.6.

6.5 Comparison of the Three Methods

In comparison to the other two methods, the first one is the simplest one. As one can see in Fig. 6.7 the method removes the trend (at least partly), but the resulting signal is not really on the desired zero line. However, the required computing time is really low since only a fixed constant is subtracted. Also nearly no memory (one data word for the mean) is required. The method using the highpass approach gives a quite good result but needs the largest memory. In addition, the highpass method requires a little bit more computing time, but still only a few operations are required per sample if computed in recursive manner. The non-linear method fits nearly exactly to the zero line. Furthermore, much less memory is required (compared to the highpass method). As conclusion one can say that the non-linear method is the best for removing offsets – at least for the examples that we tested here.

6.6 References

[1] S. K. Mitra: *Digital Signal Processing – A Computer Based Approach*, 2nd edition, Mc Graw-Hill, 2001.

[2] C. M. Porth: *Essentials of Pathophysiology – Concepts of Altered Health States*, 3rd edition, Wolters Kluwer, 2011.

[3] J. G. Proakis, D. G. Manolakis: *Digital Signal Processing – Principles, Algorithms, and Applications*, 3rd edition, Prentice Hall, 1996.

[4] F. J. Taylor: *Digital Filter Design Handbook*, Marcel Dekker, 1983.

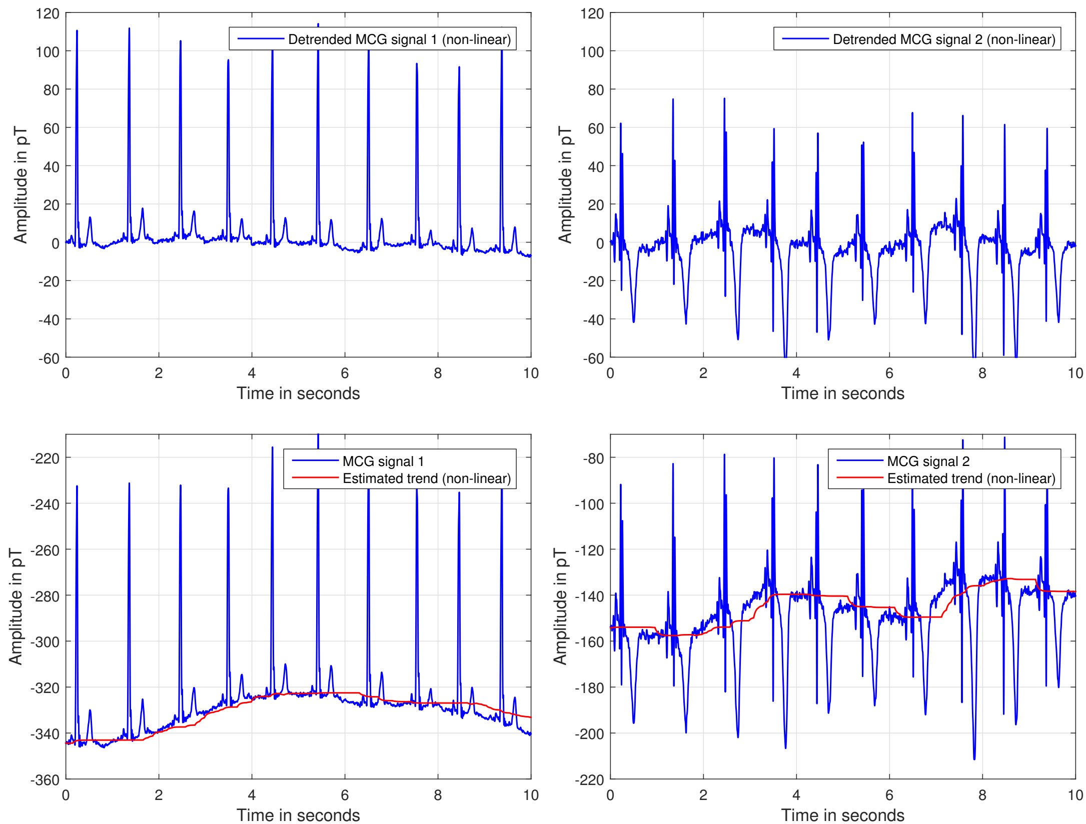


Figure 6.6: Input signals, estimated trends, and trend compensated signals using the non-linear and time-variant approach.

6.7 Code Examples

Remark:

The following code example can be downloaded via the RED website.

```
%**********************************************************************
% Clear and close everything
%**********************************************************************
clc;
clear all;
close all;

%**********************************************************************
% Load input data
%**********************************************************************
[sig, f_s] = audioread('mcg_01.wav');

%**********************************************************************
% Detrending - Method 1: Subtraction of constant (mean)
%**********************************************************************
sig_detr_const = sig - mean(sig);

%**********************************************************************
% Detrending - Method 2: Highpass
```

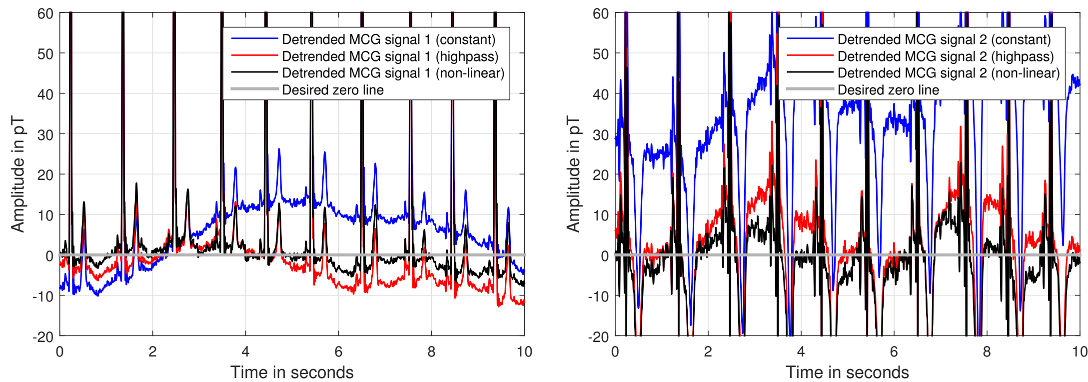


Figure 6.7: Comparison of the three methods.

```
%***************************************************************************

% Parameters ***************************************************************
N_hp     = 2*f_s;

% Initializations **********************************************************
mean_start      = mean(sig);
sig_detr_hp     = zeros(size(sig));
est_mean_sig_hp = zeros(size(sig));
sig_mem         = zeros(N_hp,1) + mean_start;
ptr_sig         = 0;
mean_rec        = mean_start;
mean_iter       = 0;
k_iter          = 0;
N_hp_inv        = 1 / N_hp;

%***************************************************************************
% Main loop
%***************************************************************************
for k = 1:length(sig)

    %***********************************************************************
    % Get new input signal
    %***********************************************************************
    sig_entering = sig(k);

    %***********************************************************************
    % Update ring buffer
    %***********************************************************************
    % Update of the pointer (modulo N_hp) **********************************
    ptr_sig = ptr_sig + 1;
    if (ptr_sig > N_hp)
        ptr_sig = 1;
    end;

    % Store leaving signal in variable *************************************
    sig_leaving      = sig_mem(ptr_sig);

    % Add new input to signal memory ***************************************
    sig_mem(ptr_sig) = sig_entering;
```

```
    %*************************************************************************
    % Update mean recursively
    %*************************************************************************
    mean_rec = mean_rec + (sig_entering - sig_leaving) * N_hp_inv;

    %*************************************************************************
    % Iteratively computed mean and correction of rec. mean
    %*************************************************************************
    k_iter    = k_iter + 1;
    mean_iter = mean_iter + sig_entering;
    if (k_iter == N_hp)
        mean_rec  = mean_iter * N_hp_inv;
        mean_iter = 0;
        k_iter    = 0;
    end

    %*************************************************************************
    % Store estimated mean (for analysis purposes)
    %*************************************************************************
    est_mean_sig_hp(k) = mean_rec;

    %*************************************************************************
    % Output = input - mean
    %*************************************************************************
    sig_detr_hp(k) = sig_entering - mean_rec;

end;

%*****************************************************************************
% Detrending - Method 3: New method (no name found yet)
%*****************************************************************************

% Parameters *****************************************************************
Cell_dur       = round(0.1 * f_s);          % Cell duration
N_mem_des      = round(1.0 * f_s);          % Total memory duration
N_cells        = round(N_mem_des/Cell_dur);  % Number of cells
mean_start     = mean(sig);
beta_sm        = 0.98;

% Initializations ************************************************************
detr_counter              = 0;
Cell_dur_inv              = 1 / Cell_dur;
mean_curr_cell            = 0;
vec_cell_means            = zeros(N_cells,1)   + mean_start;
vec_cell_means_sorted     = zeros(N_cells,1)   + mean_start;
vec_cell_means_sorted_ext = zeros(N_cells+1,1) + mean_start;
pointer_vec_cell_means    = 0;
global_mean_est           = mean_start;
sig_detr_new              = zeros(size(sig));
est_mean_sig_new          = zeros(size(sig));
global_mean_est_sm        = mean_start;

%*****************************************************************************
% Main loop
%*****************************************************************************
for k = 1:length(sig)

    %*************************************************************************
    % Get new input signal
    %*************************************************************************
    sig_entering = sig(k);
```

```
%*********************************************************************
% Increment main counter
%*********************************************************************
detr_counter = detr_counter + 1;

if (detr_counter > Cell_dur)

    % Reset counter ***************************************************
    detr_counter = 0;

    % Finalize estimation of mean of current cell ********************
    mean_curr_cell = mean_curr_cell * Cell_dur_inv;

    % Update vector containing cell means ****************************
    pointer_vec_cell_means = pointer_vec_cell_means + 1;
    if (pointer_vec_cell_means > N_cells)
        pointer_vec_cell_means = 1;
    end;
    vec_cell_means_leaving = vec_cell_means(pointer_vec_cell_means);
    vec_cell_means(pointer_vec_cell_means) = mean_curr_cell;

    % Insert new mean into sorted list ********************************
    index_offset = 0;

    for k_sort = 1:N_cells

        if ( (mean_curr_cell > vec_cell_means_sorted(k_sort)) && ...
             (index_offset == 0) )
            index_offset = 1;
            vec_cell_means_sorted_ext(k_sort)   = mean_curr_cell;
        end

        vec_cell_means_sorted_ext(k_sort+index_offset) = vec_cell_means_sorted(k_sort);

    end;

    if (index_offset == 0)
        vec_cell_means_sorted_ext(N_cells+1) = mean_curr_cell;
    end;

    % Remove leaving mean from sorted list ****************************
    index_offset = 0;

    for k_sort = 1:N_cells

        if ( (vec_cell_means_leaving == vec_cell_means_sorted_ext(k_sort)) && ...
             (index_offset == 0) )
            index_offset = 1;
        end

        vec_cell_means_sorted(k_sort) = vec_cell_means_sorted_ext(k_sort+index_offset);

    end;

    % Update mean by taking from the middle of the sorted list ********
    global_mean_est = vec_cell_means_sorted(round(N_cells/2));

    % Reset current mean estimation ***********************************
    mean_curr_cell = 0;
```

```
    else
        % Update estimation of mean of current cell **********************
        mean_curr_cell = mean_curr_cell + sig_entering;
    end;

    %*********************************************************************
    % Smoothing of the estimated mean
    %*********************************************************************
    global_mean_est_sm = beta_sm * global_mean_est_sm + ...
                         (1 - beta_sm) * global_mean_est;
    est_mean_sig_new(k) = global_mean_est_sm;

    %*********************************************************************
    % Detrend input signal
    %*********************************************************************
    sig_detr_new(k) = sig_entering - global_mean_est;
end

%*************************************************************************
% Analyses
%*************************************************************************
t_h = (0*f_s+1:30*f_s-1);
t   = (t_h-1)/f_s;
lw = 1.5;

% Time-domain analyses ***************************************************
fig = figure;
set(fig,'Units','Normalized');
set(fig,'Position',[0.1 0.1 0.8 0.8]);

sb_td_detr(1) = subplot('Position',[0.08 0.68 0.84 0.28]);
plot(t,sig_detr_const(t_h),'b','LineWidth',lw);
grid on;
set(gca,'XTickLabel','');
legend('Detrended signal (const subtraction)');

sb_td_detr(2) = subplot('Position',[0.08 0.38 0.84 0.28]);
plot(t,sig_detr_hp(t_h),'b','LineWidth',lw);
grid on;
set(gca,'XTickLabel','');
legend('Detrended signal (highpass)');

sb_td_detr(3) = subplot('Position',[0.08 0.08 0.84 0.28]);
plot(t,sig_detr_new(t_h),'b','LineWidth',lw);
grid on;
xlabel('Time in seconds');
legend('Detrended signal (new method)');

linkaxes(sb_td_detr,'xy');


% Time-domain analyses ***************************************************
fig = figure;
set(fig,'Units','Normalized');
set(fig,'Position',[0.1 0.1 0.8 0.8]);

sb_td_detr(1) = subplot('Position',[0.08 0.68 0.84 0.28]);
plot(t,sig(t_h),'b', ...
     t,sig(t_h)*0+mean(sig),'r','LineWidth',lw);
grid on;
set(gca,'XTickLabel','');
```

```
legend('Input signal','Estimated mean (const subtraction)');

sb_td_detr(2) = subplot('Position',[0.08 0.38 0.84 0.28]);
plot(t,sig(t_h),'b', ...
     t,est_mean_sig_hp(t_h),'r','LineWidth',lw);
grid on;
set(gca,'XTickLabel','');
legend('Input signal ','Estimated mean (highpass)');

sb_td_detr(3) = subplot('Position',[0.08 0.08 0.84 0.28]);
plot(t,sig(t_h),'b', ...
     t,est_mean_sig_new(t_h),'r','LineWidth',lw);
grid on;
xlabel('Time in seconds');
legend('Input signal ','Estimated mean (new method)');

linkaxes(sb_td_detr,'xy');
```

6.8 Authors of this Chapter

Christin Bald received the B.Sc. and M.Sc. degrees from Kiel University, Germany, in 2016 and 2017, respectively. Since her M.Sc. graduation she works as a research assistant in the Digital Signal Processing and System Theory group at Kiel University. Her research focus is on multichannel measurements and analysis of magnetoelectric sensor systems.

Julia Kreisel works towards her B.Sc. degree at Kiel University, Germany. During her Bachelor thesis, her aim is to measure the magnetic field of the heart for contact-free bedside diagnostic. For the magnetic measurements an array of optically pumped magnetometers is implemented into a mattress.

Gerhard Schmidt received the Dipl.-Ing. and Dr.-Ing. degrees from the Darmstadt University of Technology, Darmstadt, Germany, in 1996 and 2001, respectively. After the Dr.-Ing. degree, he worked in the research groups of the Acoustic Signal Processing Department, Harman/Becker Automotive Systems and at SVOX, Ulm, Germany. Parallel to his time at SVOX, he was a part-time Professor with the Darmstadt University of Technology. Since 2010, he has been a Full Professor with Kiel University, Germany. His main research interests include adaptive methods for speech, audio, underwater, and medical signal processing.