Robust and Efficient Digital Signal Processing

Gerhard Schmidt (Editor)

Digital Signal Processing and System Theory
Kiel University
Germany

2018

Version: **0.1** (January 2018)

Chapter 1

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:

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. 1.1 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

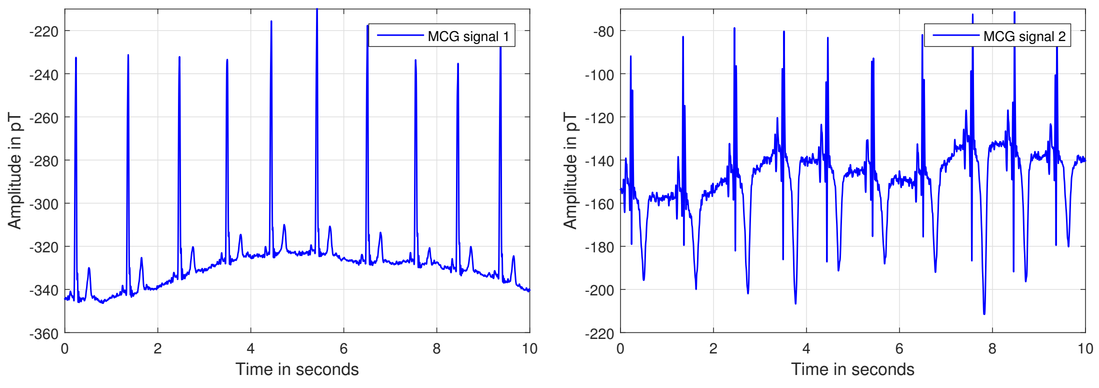

Figure 1.1: MCG (magnetocardiogram) input signals.

means an almost constant shifting from zero.

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:

$$x_{\text{trend,simple}}(n) \quad = \quad x_{\text{a prioi}}. \tag{1.1}$$

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

$$x_{\text{comp,simple}}(n) \quad = \quad x(n) - x_{\text{trend,simple}}(n). \tag{1.2}$$

To obtain a good estimate for $x_{\text{a prioi}}$ we have averaged the two input signals that are depicted in Fig. 1.1 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. (1.2). Fig. 1.2 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.

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:

$$x_{\text{trend,hp}}(n) \quad = \quad \frac{1}{N_{\text{hp}}} \sum_{i=0}^{N_{\text{hp}}-1} x(n-i). \tag{1.3}$$

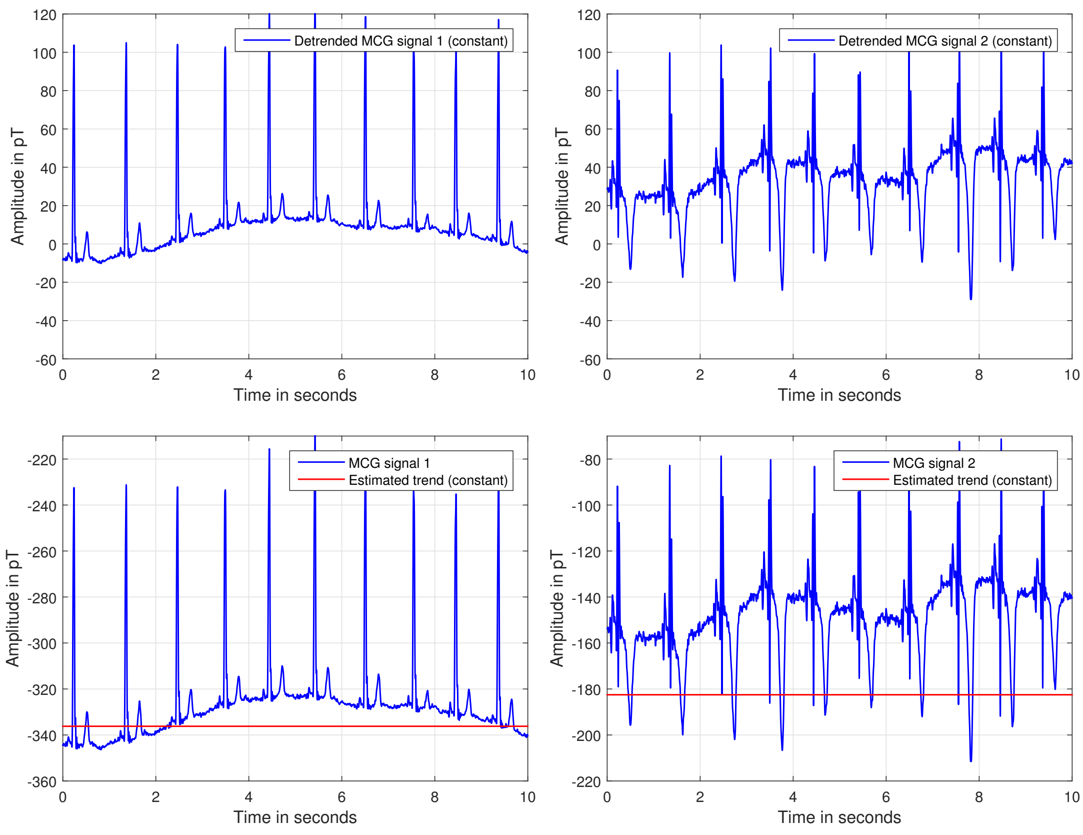

Figure 1.2: Input signals, estimated trends, and trend compensated signals using the method of constant subtraction.

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

$$x_{\text{comp,hp}}(n) \quad = \quad x(n) - x_{\text{trend,hp}}(n), \tag{1.4}$$

resulting in the desired trend removal. While Eq. (1.3) describes a lowpass filter, the subtraction in Eq. (1.4) results in a highpass filter, which gives this section also its name. Both filters are related (in the frequency domain) as:

$$H_{\text{hp}}\big(e^{-j\Omega}\big) \quad = \quad 1 - H_{\text{lp}}\big(e^{-j\Omega}\big). \tag{1.5}$$

The frequency response of the lowpass filter can be obtained by first having a closer look on Eq. (1.3) in the time domain. The equation can be interpreted as a convolution of the input signal with the lowpass impulse response $h_{\text{lp},n}$:

Remark:

The approach presented here uses only causal memory. If file-based processing is performed, the filter operation of Eq. (1.3) could also start at $i = -N_{\text{hp}}/2 - 1$ and end at $i = N_{\text{hp}}/2$.

$$x_{\text{trend,hp}}(n) \quad = \quad \sum_{i=\infty}^{\infty} h_{\text{lp},i}\, x(n-i). \tag{1.6}$$

Comparing Eqs. (1.3) and (1.6) leads to the FIR (finite impulse response, [1, 3, 4]) system

$$h_{\text{lp},i} = \begin{cases} \dfrac{1}{N_{\text{hp}}}, & \text{if } 0 \leq i < N_{\text{hp}}, \\ 0, & \text{else}. \end{cases} \tag{1.7}$$

Taking also the subtraction of Eq. (1.4), which actually detrends the signal, into account leads to the impulse response of the high pass filter (again an FIR filter):

$$h_{\text{hp},i} = \begin{cases} 1 - \dfrac{1}{N_{\text{hp}}}, & \text{if } i = 0, \\ -\dfrac{1}{N_{\text{hp}}}, & \text{if } 1 \leq i < N_{\text{hp}}, \\ 0, & \text{else}. \end{cases} \tag{1.8}$$

In Fig. 1.3 the magnitude responses of resulting lowpass (left side) and highpass filter (right side) are depicted. For a sample rate of $f_s = 1000\,\text{Hz}$ a filter order $N_{\text{hp}} = 2000$ was chosen, leading to an average based on the last two seconds of the signal.

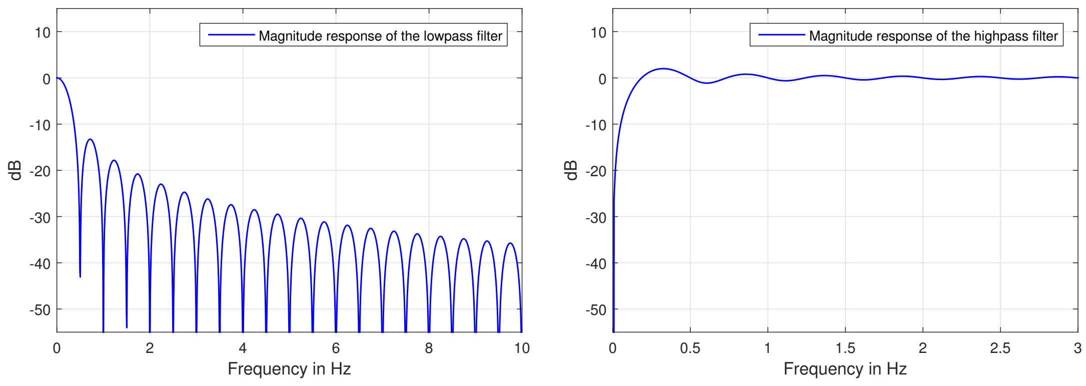

Figure 1.3: Magnitude responses of the lowpass and highpass filter for a filter order of $N_{\text{hp}} = 2000$ and a sample rate of $f_s = 1000\,\text{Hz}$.

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

$$x_{\text{trend,hp}}(n) = x_{\text{trend,hp}}(n-1) + \frac{1}{N_{\text{hp}}} \Big[x(n) - x(n - N_{\text{hp}}) \Big]. \tag{1.9}$$

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. (1.9) 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. (1.9):

$$x_{\text{trend,hp}}(n) = \begin{cases} \dfrac{x_{\text{trend,reset}}(n)}{N_{\text{hp}}}, & \text{if} \quad \bmod\,(n, N_{\text{hp}}) \equiv N_{\text{hp}} - 1, \\ x_{\text{trend,hp}}(n-1) + \frac{1}{N_{\text{hp}}}\Big[x(n) - x(n - N_{\text{hp}})\Big], & \text{else.} \end{cases} \tag{1.10}$$

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

$$x_{\text{trend,reset}}(n) = \begin{cases} x(n), & \text{if} \quad \bmod\,(n, N_{\text{hp}}) \equiv 0, \\ x_{\text{trend,reset}}(n-1) + x(n), & \text{else.} \end{cases} \tag{1.11}$$

To show the results of this second method, the simulation of Fig. 1.2 has been repeated, but now with the highpass method. Fig. 1.4 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.

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 $f_s = 1$ kHz, we get

$$N_{\text{global}} = 1000, \tag{1.12}$$

$$N_{\text{frame}} = 100. \tag{1.13}$$

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

$$N_{\text{global}} = K_{\text{frame}}\, N_{\text{frame}}, \tag{1.14}$$

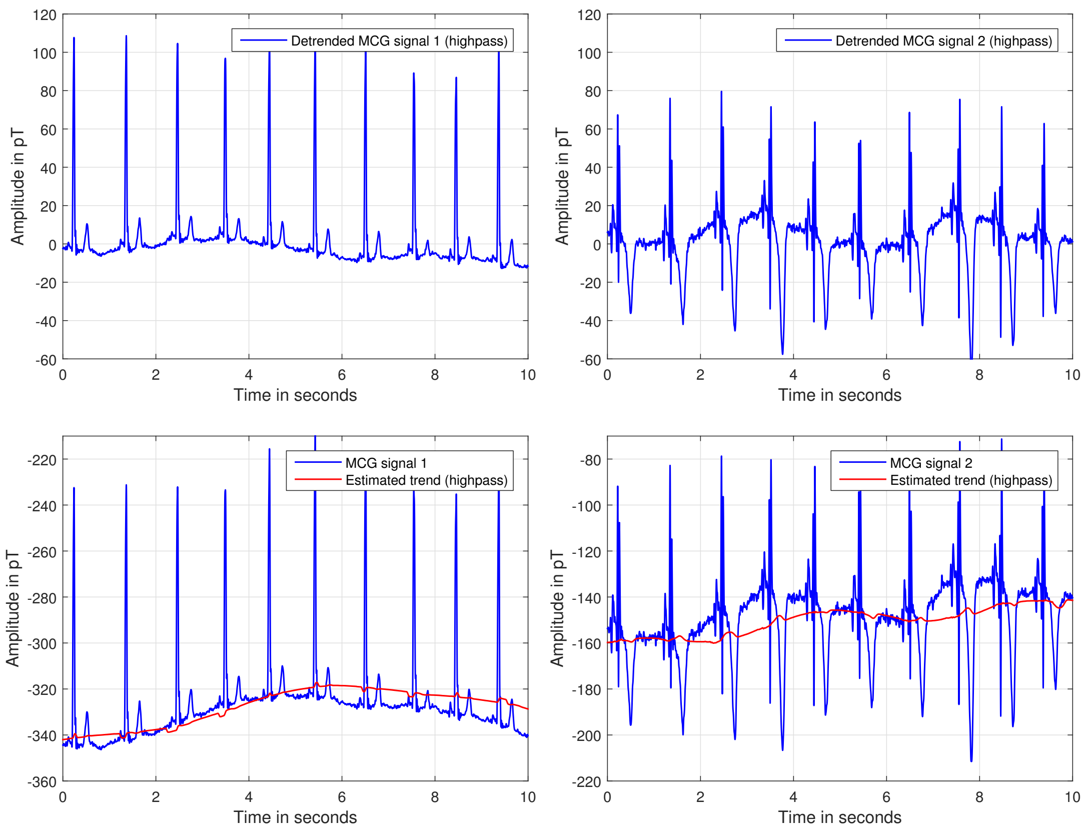

Figure 1.4: Input signals, estimated trends, and trend compensated signals using highpass filtering.

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

$$x_{\text{curr. mean}}(n) = \begin{cases} \frac{x(n)}{N_{\text{frame}}}, & \text{if } \operatorname{mod}(n, N_{\text{frame}}) \equiv 0, \\ x_{\text{curr. mean}}(n-1) + \frac{x(n)}{N_{\text{frame}}}, & \text{else.} \end{cases} \tag{1.15}$$

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

$$n = \lambda N_{\text{frame}} - 1, \tag{1.16}$$

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. (1.15) only one first-order recursive smoothing process and the subtraction of the estimated trend according to

$$x_{\text{comp,nl}}(n) = x(n) - x_{\text{trend,nl}}(n) \tag{1.17}$$

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

$$\text{mod}\big(n, N_{\text{frame}}\big) \quad \equiv \quad N_{\text{frame}} - 1, \tag{1.18}$$

which is an equivalent condition to Eq. (1.16). If we would use the (subsamples) estimated trend directly in Eq. (1.17), sudden signal *jumps* might appear. For that reason, first order IIR smoothing is performed to avoid such artifacts:

$$x_{\text{trend,nl}}(n) = \beta_{\text{sm}}\, x_{\text{trend,nl}}(n-1) + (1-\beta_{\text{sm}})\, \tilde{x}_{\text{trend,nl}}\left(\left\lfloor \frac{n}{N_{\text{frame}}} \right\rfloor\right). \tag{1.19}$$

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

$$0 \ll \beta_{\text{sm}} < 1. \tag{1.20}$$

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 (1.18) the short-term mean is added to a vector that contains the last K_{frame} short-term means

$$\boldsymbol{x}_{\text{mean}}(n) \quad = \quad \begin{cases} \Big[x_{\text{curr. mean}}(n),\ x_{\text{curr. mean}}(n-N_{\text{frame}}),\ \ldots,\ x_{\text{curr. mean}}\big(n-(K_{\text{frame}}-1)N_{\text{frame}}\big)\Big]^{\text{T}}, \\ \qquad \text{if } \text{mod}(n, N_{\text{frame}}) \equiv N_{\text{frame}} - 1, \\ \boldsymbol{x}_{\text{mean}}(n-1), \\ \qquad \text{else.} \end{cases} \tag{1.21}$$

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

$$\boldsymbol{x}_{\text{mean, sorted}}(n) \quad = \quad \Big[x_{\text{mean, sorted, }0}(n),\ x_{\text{mean, sorted, }1}(n),\ \ldots,\ x_{\text{mean, sorted, }K_{\text{frame}}-1}(n)\Big]^{\text{T}}. \tag{1.22}$$

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.

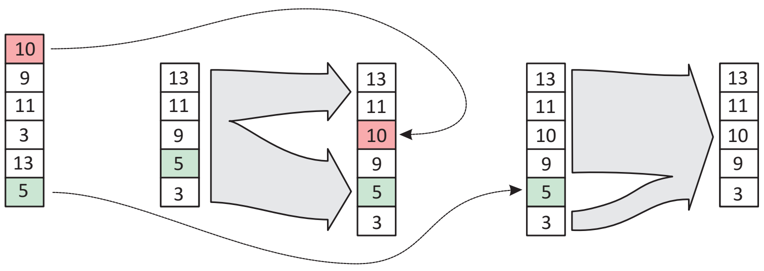

Figure 1.5: Sorting of the frame means.

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. 1.5 shows an overview about the two-stage sorting procedure.

Every N_{frame} sample the (non-smoothed) trend estimate is updated according to

$$\tilde{x}_{\text{trend,nl}}(n) \quad = \quad \begin{cases} x_{\text{mean, sorted, }K_{\text{frame}}/2}(n), & \text{if mod}(n, N_{\text{frame}}) \equiv N_{\text{frame}} - 1, \\ \tilde{x}_{\text{trend,nl}}(n-1), & \text{else.} \end{cases} \tag{1.23}$$

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. 1.6.

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. 1.7 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 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.

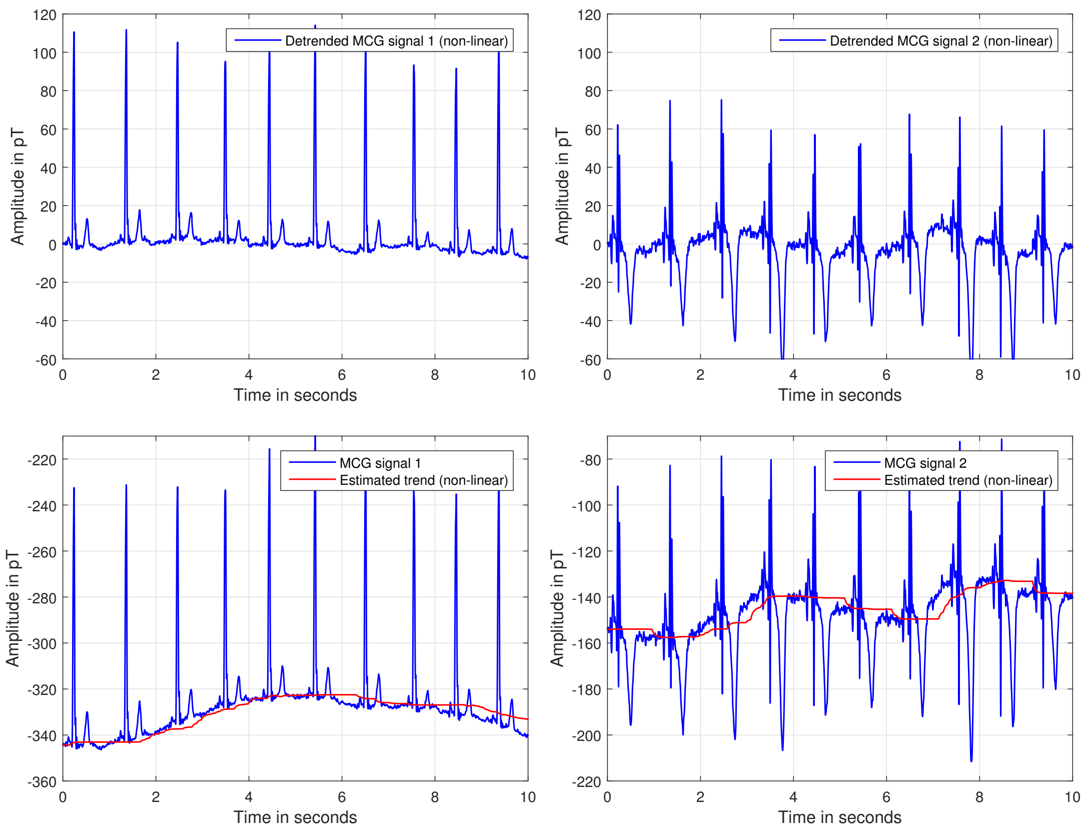

Figure 1.6: Input signals, estimated trends, and trend compensated signals using the non-linear and time-variant approach.

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
```

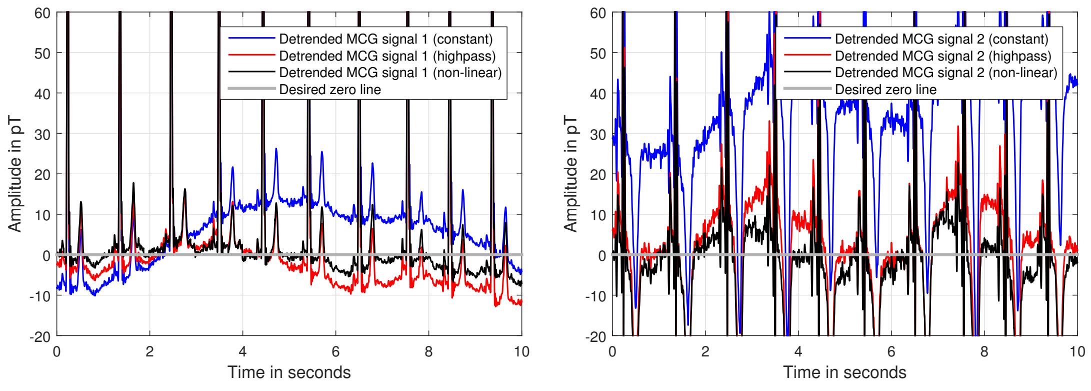

Figure 1.7: Comparison of the three methods.

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

% 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 - 1)

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

    % Use current sample **********************************************
    mean_curr_cell = mean_curr_cell + sig(k);

    % 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');
```

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.