Robust and Efficient Digital Signal Processing

Gerhard Schmidt

Digital Signal Processing and System Theory
Kiel University
Germany

2017

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 . . . 14
2.6 Authors of this Chapter . . . 18

3 Prediction-based Filter Design 19

3.1 Basics . . . 19
3.2 Application Examples . . . 19
3.3 References . . . 21
3.4 Authors of this Chapter . . . 22

4 Complex Magnitude Approximations 23

4.1 Problem . . . 23
4.2 A Very Simple Approximation . . . 23
4.3 A Better Approximation . . . 23
4.4 References . . . 23
4.5 Authors of this Chapter . . . 24

Part I

Basics

Chapter 1

Fast Convolution Without Additional Delay

written by Gerhard Schmidt, Anton Namenas, Seedo Eldho Paul

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
2.5	Code Examples	14
2.6	Authors	18

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

```
%**************************************************************************
% Basic parameters
%**************************************************************************
N     = 588;               % Filter lenght
```

Remark:

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

```
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

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.

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.

Chapter 2

Recursive Computation of Signal Vector Norms

written by Gerhard Schmidt, Owe Wisch, Katharina Rebbe

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 14
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

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

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.

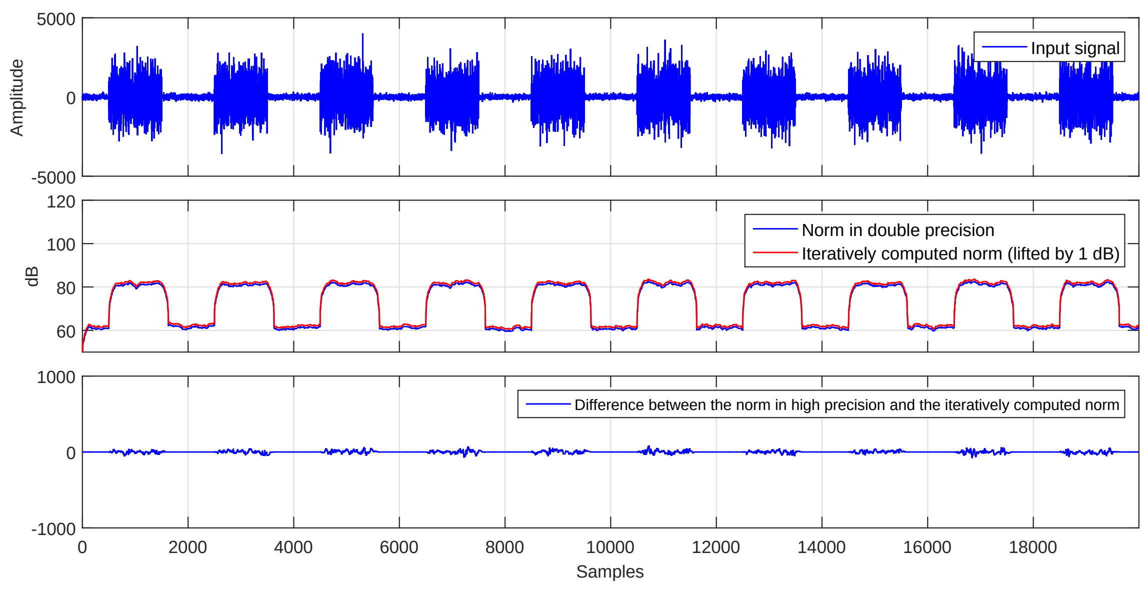

Figure 2.1: Input signal and iterative norm computations.

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:

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

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

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

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

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).

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

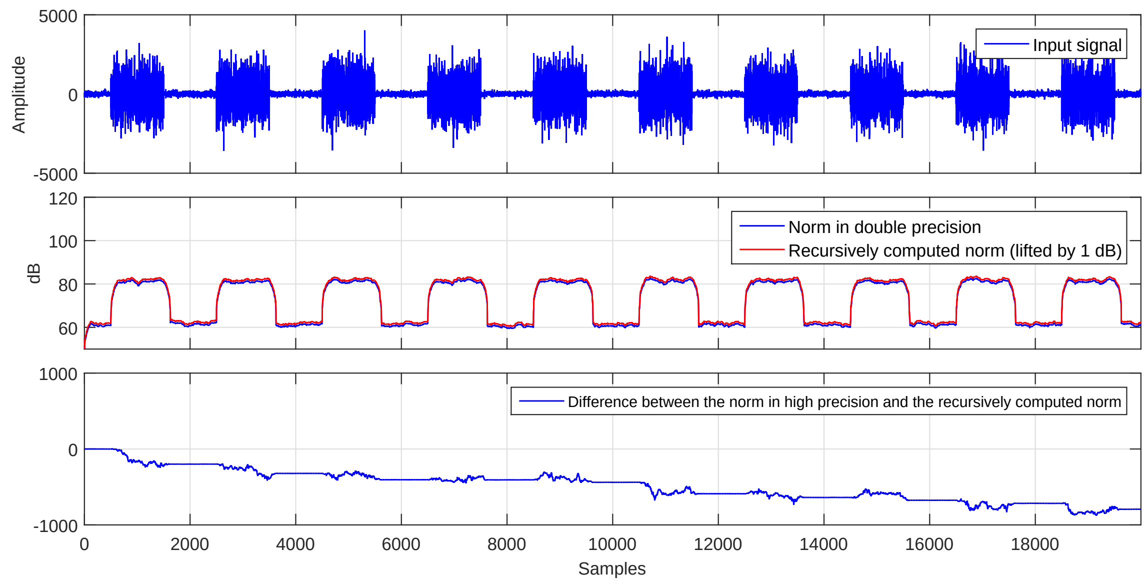

Figure 2.2: Input signal and recursive as well as iterative norm computations.

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:

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

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:

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

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.

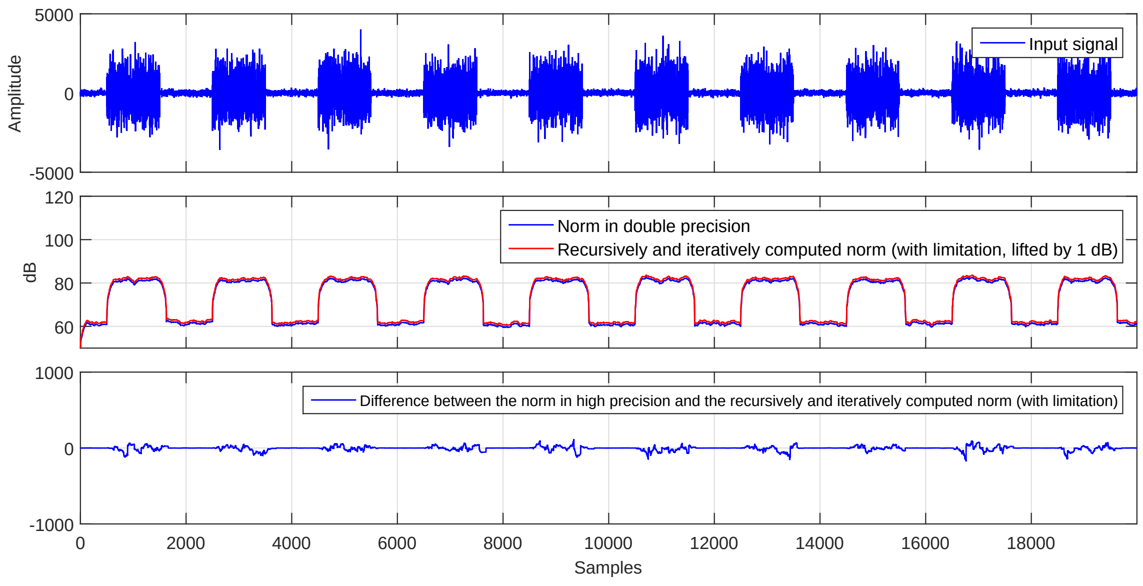

Figure 2.3: Input signal and mixed recursive/iterative as well as purely iterative norm computations.

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

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.

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.

Chapter 3

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:

3.1	Basics	19
3.2	Application Examples	19
3.3	References	21
3.4	Authors	22

3.1 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.

3.2 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.

Remark:

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

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

- **Loudspeaker equalization**

  To improve the playback quality of loudspeakers equalization filters might be placed before the DA converters of playback devices (see Fig. 3.1). These filters are designed such that the frequency response of the system consisting of the loudspeaker itself and the equalization filter should be close to a predefined curve.

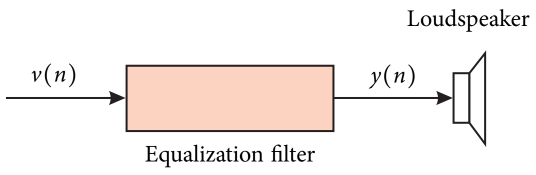

Figure 3.1: Basic structure of loudspeaker equalization schemes.

If more than one loudspeaker should be equalized often additional restrictions such as linear phase behaviour (constant group delay) are desired. Fig. 3.2 shows an example of such a desired frequency response together with a non-equalized loudspeaker and its equalized counterpart.

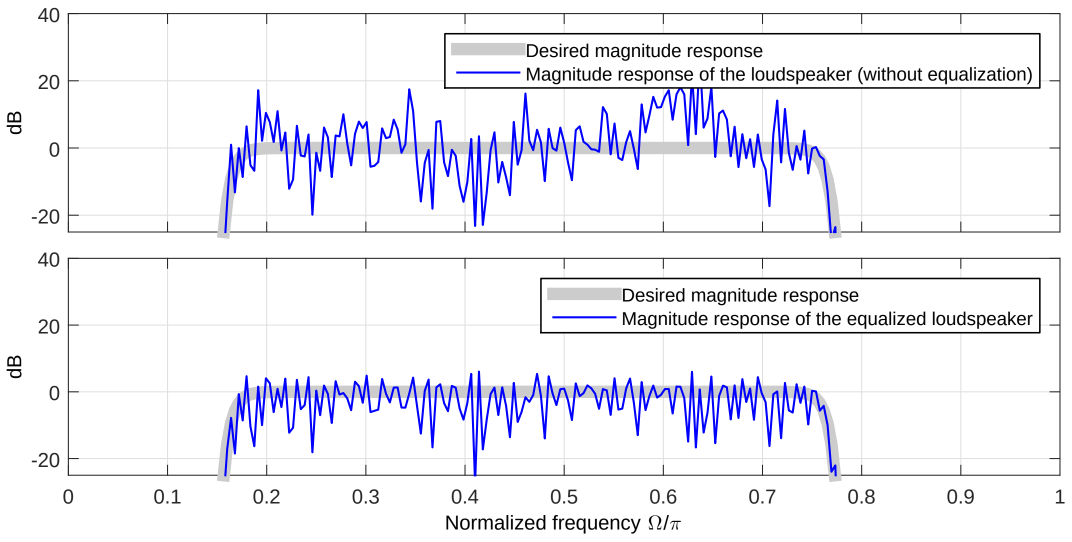

Figure 3.2: Magnitude responses of the non-equalized and the equalized loudspeaker.

- **Low-delay noise suppression**

  Whenever a desired signal is superimposed by noise signal enhancement techniques can be applied (see Fig. 3.3). Usually, statistically optimized, time-variant filters such as so-called *Wiener filters* [1] are utilized here.

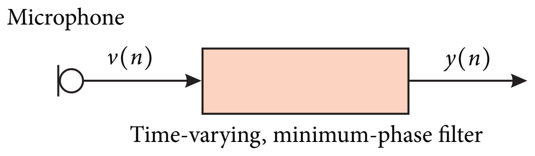

Figure 3.3: Low-delay noise suppression.

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. 3.4 shows an example of a noisy speech signal (filter input) and the corresponding noise-reduced signal (filter output).

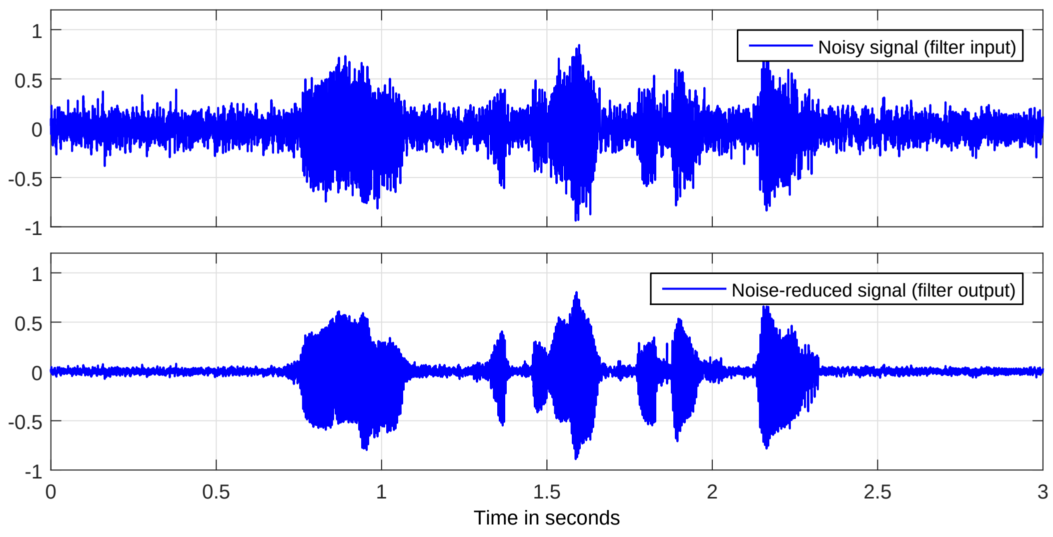

Figure 3.4: Signal before and after noise suppression.

- **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. 3.5).

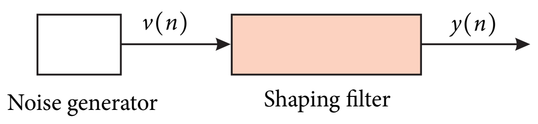

Figure 3.5: Signal generation.

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. 3.6 shows an example of in input and output PSDs (in blue) together with the desired PSD (in grey).

3.3 References

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

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

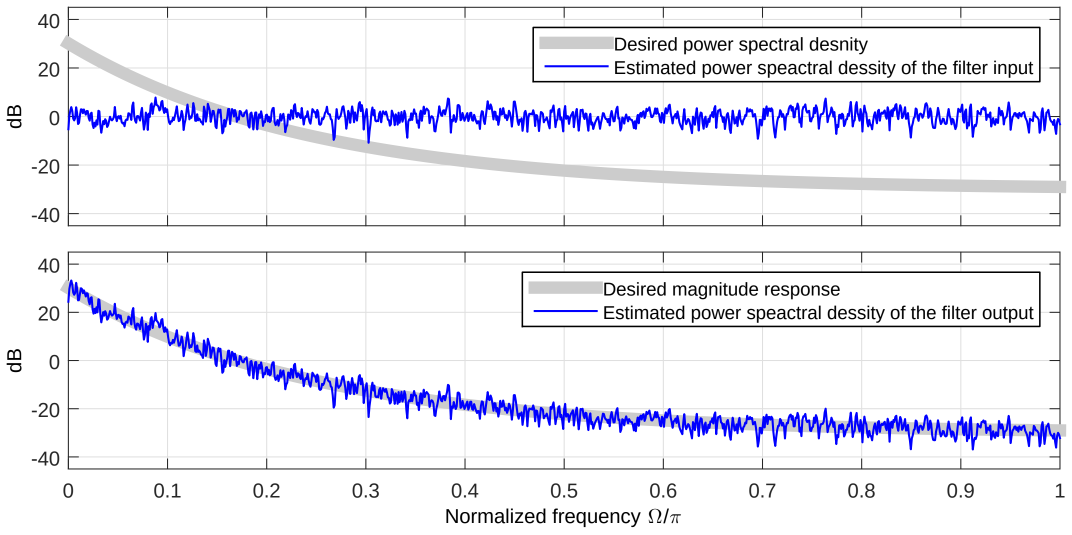

Figure 3.6: Power spectral density before and after the shaping filter.

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

3.4 Authors of this Chapter

Gerhard Schmidt   Text GUS

Chapter 4

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:

4.1	Problem	23
4.2	A Very Simple Approximation	23
4.3	A Better Approximation	23
4.4	References	23
4.5	Authors	24

4.1 Problem

- Need for magnitude instead of magnitude square values

- Complexity of square root computations

4.2 A Very Simple Approximation

Some text ...

4.3 A Better Approximation

Some text ...

4.4 References

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

4.5 Authors of this Chapter

Gerhard Schmidt Text GUS