Robust and Efficient Digital Signal Processing

Gerhard Schmidt

Digital Signal Processing and System Theory
Kiel University
Germany

2017

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:

1 Problem

In several signal processing applications

2 References

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

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')
```

4 Authors of this Chapter

Anton Namenas received the ...

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.