Prof. Dr. Stefan Weinzierl 13.12.2012

Musterlösung: 15. Januar 2013, 15:34

Funktionsweise Digitaler Audio Workstations

In einer Digitalen Audio Workstation (DAW) sind Audioeffekte als separate Programme, die sogenannten Plug-ins, realisiert. Die DAW, in diesem Zusammenhang auch oft Host genannt, organisiert Aufnahme, Abspielen, Routing und Mixing von Audio-Dateien, sowie deren Verarbeitung durch die Plug-ins. Dabei erhalten die Plug-ins von der DAW jeweils Blöcke mit N Samples und müssen Blöcke der selben Länge zurückgeben. Da die Festlegung der Blocklänge zentral erfolgt, muss ein Plug-in mit jeder beliebigen Blockgröße umgehen können.
Die Arbeitsweise einer DAW und einfacher Plug-ins wird in den folgenden Tutorien anhand einiger Beispiele simuliert. Dabei wird die Aufgabe des Hosts von einem Matlab-Skript übernommen, das Audiosignale lädt oder erzeugt und Blockweise an eine Funktion – das Plug-in – übergibt. Innerhalb der Plug-ins soll die Verarbeitung der Audiosignale Sampleweise erfolgen und ohne Matlab-interne Funktionen (`filter.m`) gearbeitet werden.

Digitale Audioeffekte in Matlab - Implementierung eines digitalen Filters

a) Schreiben Sie in Matlab eine Funktion, die ein Audiosignal mit einem beliebigen Koeffizientensatz filtert und das Ergebnis zurückgibt.

$$y = blockFilter(b,\ a,\ x) \qquad , mit$$

x, y: Ein- und Ausgangssignal
b: nicht rekursive Filterkoeffizienten $[b_0,\ b_1, \ldots,\ b_N]$
a: rekursive Filterkoeffizienten $[1,\ a_1,\ a_2,\ \ldots,\ a_N]$

Testen Sie die Funktion mit dem Filter aus Abb. 1 und einem Audio-File aus dem Downloadverzeichnis. Testen Sie das Filter außerdem mit einem weißen Rauschen von einer Sekunde Länge, indem Sie die spektralen Eigenschaften des Ausgangssignals auswerten.

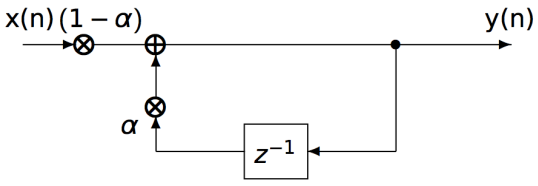


Abbildung 1: Blockschaltbild eines digitalen Filters

Lösung: Die Funktion blockFilter befindet sich am Ende.

```
%% a)
```

```
% read audio file
[x, fs] = wavread('white_noise.wav');
x = mean(x, 2);


% calculate filter coeffs
alpha = .5;
b = 1-alpha;
a = [1 alpha];

%% filter Signal
y1 = blockFilter(b, a, x);

%%
soundsc(x, fs)
sound(y1, fs)
```

b) Überprüfen Sie die Berechnung der ersten beiden Ausgangssamples innerhalb von *blockFilter.m* mit Hilfe der Debugging Funktion von Matlab.
c) Vergleichen Sie das Ergebnis der Filterung Ihrer Funktion *blockFilter.m* mit dem von der Matlab-Funktion *filter.m* berechneten Signal für dieselben Filterkoeffizienten.
Lösung: Um den Debug-Modus in Matlab zu starten muss zunächst ein Break-Point gesetzt werden. Dafür muss in der Funktion *blockFilter.m* der Maus-Cursor in eine Zeile bewegt werden und dann im Matlab-Menü unter Debug der Punkt Set/Clear Breakpoint angewählt werden. Wird die Funktion dann aufgerufen bleibt sie an der entsprechenden Stelle stehen und die Berechnung kann mit der Step Option aus dem Debug-Menü Schritt für Schritt überprüft werden.

Digitale Audioeffekte in Matlab - Blockweise Signalverarbeitung

a) Erweitern Sie die Funktion so, dass die Berechnung in Blöcken der Länge N erfolgen kann und erzeugen Sie ein "Host-Skript", das in einer Schleife die Blöcke für die Verarbeitung bereitstellt.
Lösung:

```
%% c)

% allocate memory for output vector
y2 = zeros(size(x));

% Block length
N = 512;

% number of process runs
endPos = floor(length(x)/N);

% process audio in blocks
for idx = 1:endPos

    % fill input buffer
    bufferIn = x((idx-1)*N+1:idx*N);

    % process
    if idx == 1
```

```
        [bufferOut, s] = blockFilter(b, a, bufferIn);
    else
        [bufferOut, s] = blockFilter(b, a, bufferIn, s);
    end

    % read from output buffer
    y2((idx-1)*N+1:idx*N) = bufferOut;

end

% controll if results from a) and b) are the same
plot(y1-y2)

%%
soundsc(x, fs)
soundsc(y2, fs)
```

d) Stellen Sie den Betrags- und Phasenfrequenzgang sowie das Pol/Nullstellen-Diagram des Filters aus Abb.1 dar.

Lösung: Die Differenzengleichung des Filters ist:

$$y[n] = (1-\alpha)x[n] + \alpha y[n-1] \tag{1}$$

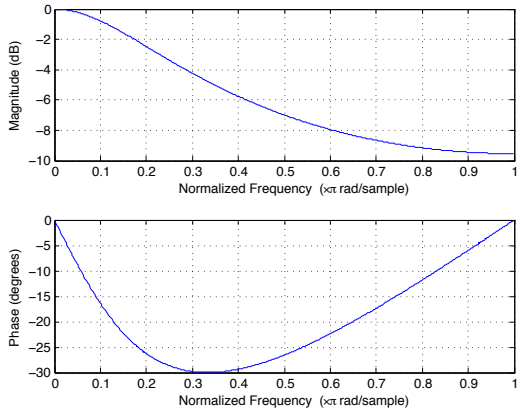
Z-Transformierte und Übertragungsfunktion sind dann:

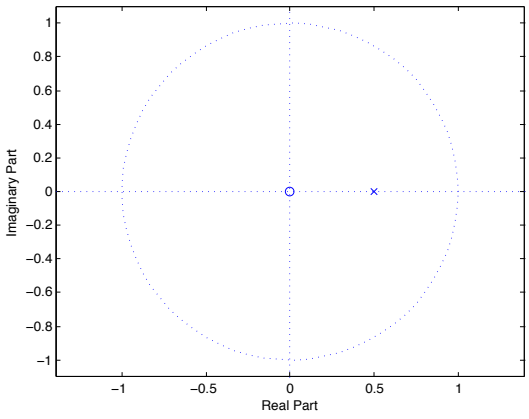
$$\begin{aligned} Y(z) &= (1-\alpha)X(z) + \alpha Y(z)z^{-1} \\ Y(z) - \alpha Y(z)z^{-1} &= (1-\alpha)X(z) \\ Y(z)(1-\alpha z^{-1}) &= (1-\alpha)X(z) \end{aligned}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1-\alpha z^{-1}}$$

Anschließend kann Matlab für die Darstellung benutzt werden

```
%% d)

% a z-trandformation is needed. That switches the sign of the recursiv
% coefficients

a2    = -a;
a2(1) = 1;

% plot magnitude and phase response
figure
freqz(b, a2)

% plot pole/zero diagram
figure
zplane(b, a2)
```






```
% [y s] = blockFilter(b, a, x, s)
%
% filters signal x which must be MONO with coefficients given by b and a.
% It returns two buffers in struct s, that are needed for blockwise
% processing of an input signal.
%
% input:
% b = [b0 b1 b2 ... bN]
% a = [a0 a1 a2 ... aN]
% x = [x0 x1 x2 ... xN]
% s : input and output buffers from last function call (optional)
%
% output:
% y : filtered signal
% s : input and output buffers

function [y s] = blockFilter(b, a, x, s)

% ---------------------------------- preparation ------------------------------------ %
% check input format
if size(x,2) > size(x,1);    x = x';     end
if size(b,1) > size(b,2);    b = b';     end
if size(a,1) > size(a,2);    a = a';     end
% initalite values
y       = zeros(size(x));
N       = length(x);
% create buffer if not passed to function
if nargin == 3
    s.buff_x = zeros(size(b));
    s.buff_y = zeros(size(a));
end

% ---------------------------------- process loop ----------------------------------- %
% filter audio sample by sample
for idx = 1:N
    % fill buff_b with current input x(idx)
    s.buff_x    = circshift(s.buff_x, [0 1]);
    s.buff_x(1) = x(idx);

    % calculate output and write to y
    % b coefficients
    for n = 1:length(b)
        y(idx) = y(idx) + s.buff_x(n)*b(n);
    end

    % a coefficients
```

```
    for n = 2:length(a)
        y(idx) = y(idx) + s.buff_y(n)*a(n);
    end

    % fill buff_a with current output y(idx)
    s.buff_y    = circshift(s.buff_y, [0 1]);
    s.buff_y(2) = y(idx);

end
```