Prof. Dr. Stefan Weinzierl 09.01.2013

Musterlösung: 11. Januar 2014, 18:25

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()`) gearbeitet werden.

Digitale Audioeffekte in Matlab - largeImplementierung eines digitalen Filters

a) Wir wollen einen digitalen Filter implementieren, der in Abbildung 1 dargestellt ist. Stellen Sie hierzu zunächst die Übertragungsfunktion auf und stellen sie Amplituden- und Phasenfrequenzgang, sowie ein Pol-Nullstellen-Diagramm grafisch dar (z.B. für $\alpha = 0.5$).

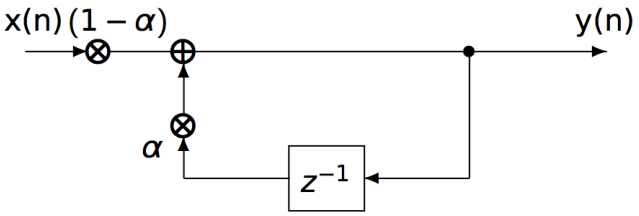


Abbildung 1: Blockschaltbild eines digitalen Filters

Lösung: Die Differenzengleichung des Filters ist:

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

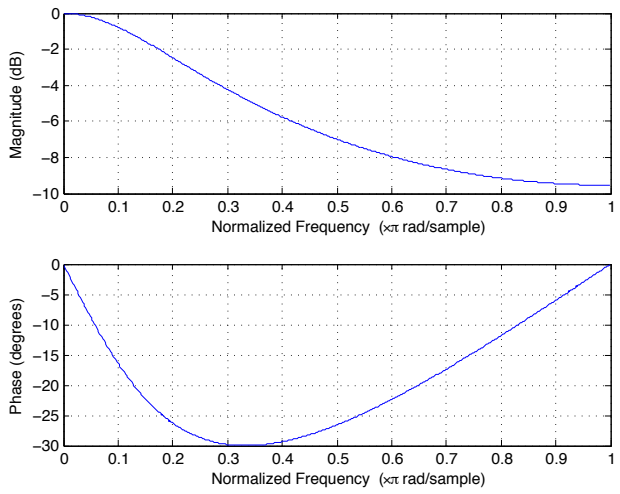
Z-Transformierte und Übertragungsfunktion sind dann:

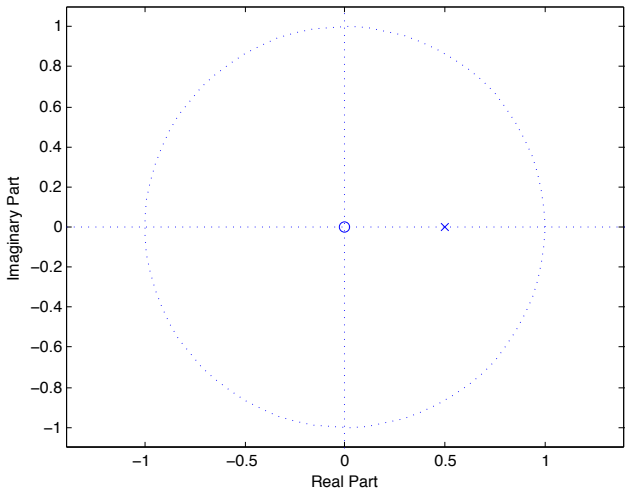
$$\begin{aligned} Y(z) &= (1-\alpha)X(z) + \alpha Y(z)z^{-1} \\ Y(z) - \alpha Y(z)z^{-1} &= (1-\alpha)X(z) \\ Y(z)(1-\alpha z^{-1}) &= (1-\alpha)X(z) \end{aligned}$$

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

Anschließend kann Matlab für die Darstellung benutzt werden

```
% Aufgabe a)
clear all
close all
clc

% Definieren eines Beispiel-alphas:
alpha = 0.5;

% Berechnen der Filterkoeffizienten:
b = (1-alpha);
a = [1 -alpha];

% Plotten des Amplituden und Phasenfrequenzgangs:
figure;
freqz(b,a);

% Plotten des Pol-Nullstellendiagramms:
figure;
zplane(b,a);
```





b) Erzeugen Sie ein Host-Skript, welches eine DAW mit einem Dummy-Plugin `blockFilter` simuliert. Das heißt ein Skript, dass eine Mono-Datei einliest und in Blöcken der Länge N an das Dummy-Plugin weitergibt, welches seinen Namen trägt, weil es nichts mit dem Eingang macht und ihn sofort ausgibt. Implementieren Sie `blockFilter` als Matlab-Funktion der Form: `y = blockFilter(x)`

Lösung:

blockFilter.m:

```
function  y = blockFilter( x )
%BLOCKFILTER Does nothing to input and outputs it. Lazy lazy function!
y = x;

end
```

DAWhost.m

```
clear all
```

```
2  close all
3  clc
4
5  % Einlesen des zu filternden Signals
6  [signal , Fs] = audioread('Musik.wav');
7  temp_signal = signal;
8
9  % Festsetzen der Blocklaenge:
10 N = 512;
11
12
13 % Berechnen wieviele Bloecke der Laenge N aus dem Eingangssignal erstellt
14 % werden koennen und falls die Anzahl der Bloecke nicht ganzzahlig ist,
15 % erweitern mit Nullen:
16 numFrames = ceil(length(signal)/N);
17 if mod(length(signal),N)~= 0
18     numNull = N-mod(length(signal),N);
19     temp_signal = [temp_signal;zeros(numNull,1)];
20 end
21
22 % Berechnen des Outputs in Bloecken:
23 for idx = 1:numFrames
24     % Frame aus dem Gesamtsignal herausloesen:
25     bufferIn = temp_signal((idx-1)*N+1:idx*N,1);
26
27     % Output Berechnen:
28     bufferOut = blockFilter(bufferIn);
29
30     output((idx-1)*N+1:idx*N,1) = bufferOut;
31 end
32
33 % Kuerzen des Outputs um die am Anfang angehaengten Nullen
34 output = output(1:end-numNull);
```

Alternativ hätte man Zeilen 23-31 auch wie im Tutorium angemerkt wie folgt schreiben können:

```
1  for idx = 1:N:1+N*(numFrames-1)
2      % Frame aus dem Gesamtsignal herausloesen:
3      bufferIn = temp_signal(idx:idx+N-1,1);
4
5      % Output Berechnen:
6      bufferOut = blockFilter(bufferIn);
7
8      output(idx:idx+N-1,1) = bufferOut;
9  end
```

c) Ändern Sie das Dummy-Plugin so ab, dass es ein Audiosignal mit einem beliebigen Koeffizientensatz (b, a) 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.
Lösung:

blockFilter.m:

```
function [y,s] = blockFilter( b, a, x, s )
%BLOCKFILTER.m 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 block
% s : input and output buffers

% check input format:
if size(x,2) > size(x,1); x = x'; end
if size(b,1) > size(x,2); b = b'; end
if size(a,1) > size(x,2); a = a'; end
% initialize 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

% filter audio sample by sample
for idx = 1:N
    % fill buffer 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 buffer with current output y(idx)
    s.buff_y    = circshift(s.buff_y, [0 1]);
    s.buff_y(1) = y(idx);
end

end
```

DAWhost.m

```
clear all
close all
```

```
clc

% Einlesen des zu filternden Signals
[signal , Fs] = audioread('Musik.wav');
temp_signal = signal;

% Festsetzen der Blocklaenge:
N = 512;

% Berechnen der Filterkoeffizienten:
alpha = 0.5;
b = 1 - alpha;
a = [1 alpha];

% Berechnen wieviele Bloecke der Laenge N aus dem Eingangssignal erstellt
% werden koennen und falls die Anzahl der Bloecke nicht ganzzahlig ist,
% erweitern mit Nullen:
numFrames = ceil(length(signal)/N);
if mod(length(signal),N)~= 0
    numNull = N-mod(length(signal),N);
    temp_signal = [temp_signal;zeros(numNull,1)];
end

% Berechnen des Outputs in Bloecken:
for idx = 1:numFrames
    % Frame aus dem Gesamtsignal herausloesen:
    bufferIn = temp_signal((idx-1)*N+1:idx*N,1);

    if  idx == 1
        [bufferOut,s] = blockFilter(b,a,bufferIn);
    else
        [bufferOut,s] = blockFilter(b,a,bufferIn,s);
    end

    output((idx-1)*N+1:idx*N,1) = bufferOut;
end

% Kuerzen des Outputs um die am Anfang angehaengten Nullen
output = output(1:end-numNull);
```

d) Überprüfen Sie die Berechnung der ersten beiden Ausgangssamples innerhalb von `blockFilter.m` mit Hilfe der Debugging Funktion von Matlab.

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.