DSP First – Laboratory Exercise #4

„M and FM Sinusoidal Signals“

1. Overview

2. Warm-up

2.1 Mathlab Synthesis of Chirp Signals
(a)

```
fsamp = 8000;
dt = 1/fsamp;
dur = 1.8;
tt = 0:dt:dur;
psi = 2*pi*(100+200*tt+500*tt.*tt); %
xx = real(7.7*exp(j*psi));
sound(xx,fsamp);
```

Es gilt:

$$f(t) = \frac{1}{2 \cdot \pi} \cdot \frac{d}{dt} \cdot \psi(t)$$

In diesem Fall ist $\psi(t) = 2\pi \cdot (100 + 200 \cdot t + 500 \cdot t^2)$

So ergibt sich $f(t) = 200 + 2 \cdot 500 \cdot t = 200 + 1000 \cdot t$

Die Minimale Frequenz ist $f(0) = 200Hz$ und die maximale $f(1{,}8) = 2000Hz$
Die hörbare Frequenz liegt also zwischen 200 und 2000 Hz.

Bei der gegebenen Abtastrate von 8000 Hz ist eine Frequenz bis etwa 4000 Hz (laut Abtasttheorem) darstellbar.

2.1
(b)

myChirp.m

```
function xx = mychirp( f1, f2, dur, fsamp )
%MYCHIRP generate a linear-FM chirp signal
%
% usage: xx = mychirp( f1, f2, dur, fsamp )
%
% f1 = starting frequency
% f2 = ending frequency
% dur = total time duration
% fsamp = sampling frequency (OPTIONAL: default is 8000)
%
if( nargin < 4 ) %-- Allow optional input argument
fsamp = 8000;
end
dt = 1/fsamp;
tt = 0: dt : dur;
u=(f2-f1)/(2*dur);
psi = 2*pi*(100+f1*tt+u*tt.*tt);
xx=real(7.7*exp(j*psi));
sound(xx,fsamp);
```

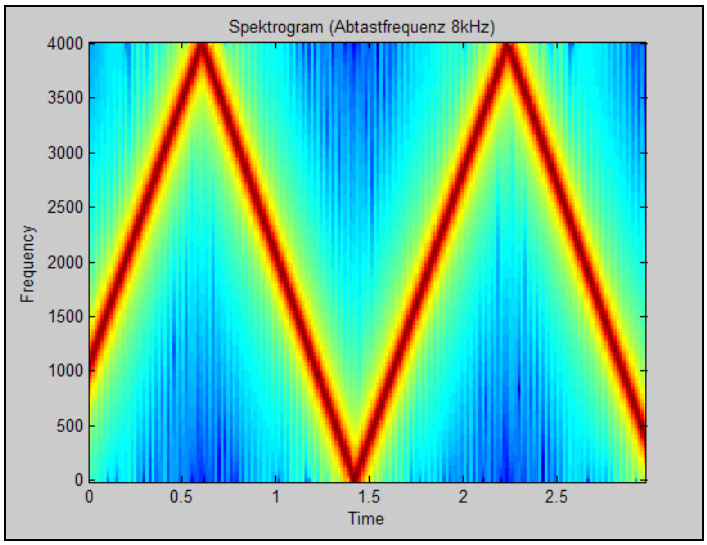
Mit Hilfe von `mychirp(200,2000,1.8,8000);` lässt sich das Geräusch aus Aufgabe (a) erzeugen.

3. Lab A: Chirps and Beats

3.1

Erzeugt man folgendes Signal:

```
xx=mychirp(15000,300,3,8000);
specgram(xx,[],8000);
```



Es scheint, als würde das Signal steigen und fallen. Dies liegt daran, dass die Sampling-rate nur 8000 Hz beträgt. Laut dem Abtast-Theorem ist also maximal eine Frequenz von 4000 Hz zulässig. Da hier 15000 Hz als Anfangsfrequenz verwendet werden, führt dies zu solch einem Verhalten. Nötig wäre für eine korrekte Darstellung eine Abtastfrequenz von mindestens 30000 Hz.

3.2 Beat Notes

(a)

Beat.m

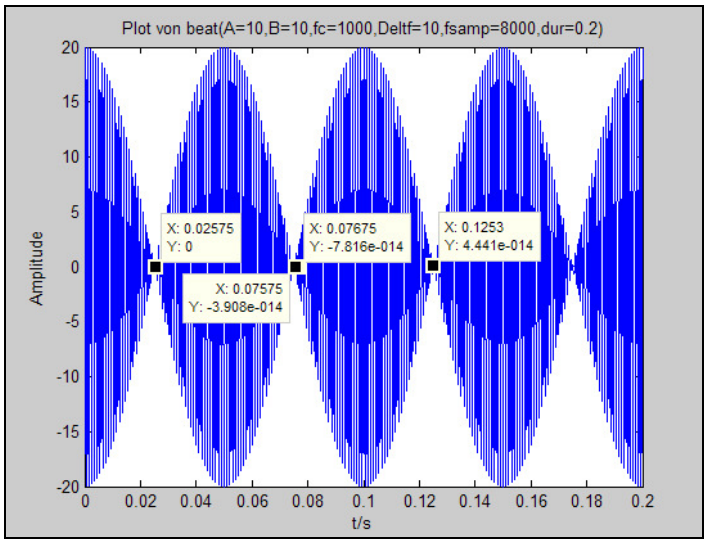
```
function [xx, tt] = beat(A, B, fc, delf, fsamp, dur)
%BEAT compute samples of the sum of two cosine waves
% usage:
% [xx, tt] = beat(A, B, f, delf, fsamp, dur)
%
% A = amplitude of lower frequency cosine
% B = amplitude of higher frequency cosine
% fc = center frequency
% delf = frequency difference
% fsamp = sampling rate
% dur = total time duration in seconds
% xx = output vector of samples
%--OPTIONAL Output:
% tt = time vector corresponding to xx
tt=[0:(1/fsamp):dur];
sig1=A*cos(2*pi*(fc-delf).*tt);
sig2=B*cos(2*pi*(fc+delf).*tt);
xx=sig1+sig2;
```

Funktion zur Erstellung einer Schwebung aus zwei cosinus-Signalen.

(b)

Erstellung einer Schwebung:

```
[xx,tt] = beat(10,10,1000,10,8000,0.2);
plot (tt,xx);
title('Plot von beat(A=10,B=10,fc=1000,Deltf=10,fsamp=8000,dur=0.2)');
```



Beide Frequenz-Anteile lassen sich bestimmen:
T=0.1253s-0.0258s=0.0995s => f ≈ 10Hz
T=0.07675s-0.07575s=0.001s => f=1000Hz

(c)

Hört man mit beatcon das Signal mit f_Δ=10Hz an, kann bei einem Signal mit f=1kHz erkannt werden, dass es abwechselnd lauter und leiser wird, da sich die Amplitude zehn mal in der Sekunde auf 0 absenkt.

(d)

Experimente mit f_Δ ergeben, dass je höher diese Frequenz eingestellt wird, desto stärker das Signal oszilliert, bis man schließlich meint ein nicht oszillierendes Signal zu vernehmen.

4 Lab B: FM Synthesis of Instrument Sounds

4.1 Generating the Bell Envelopes

Bellenv.m

```
function yy = bellenv(tau, dur, fsamp);
% BELLENV produces envelope function for bell sounds
%
%     usage: yy = bellenv(tau, dur, fsamp);
%
%          Where tau = time constant
%                dur = duration fo the envelope
%              fsamp = sampling frequency
%
%          Returns:
%                 yy = decaying exponential envelope
%
% note: produces exponential decay for positive tau.
tt=[0:(1/fsamp):dur];
yy=exp(-tt./tau);
```

Damit wird die Einhüllende für den Glocken-Ton erzeugt.

4.2 Parameters for the Bell

Bell.m

```
function [xx fi_t tt] = bell(ff, Io, tau, dur, fsamp)
% BELL produce bell sound
%
%     usage: xx = bell(ff, Io, tau, dur, fsamp)
%
%     Where: ff = frequency vector( containing fc and fm)
%            Io = scale factor for modulation index
```

```
%           tau = decay parameter for A(t) and I(t)
%         fsamp = sampling rate

tt=[0:(1/fsamp):dur];
A_t=bellenv(tau,dur,fsamp);
I_t=Io*bellenv(tau,dur,fsamp);
xx=A_t.*cos((2*pi*ff(1)*tt) + I_t.*cos((2*pi*ff(2)*tt)-(pi/2))-(pi/2));

% Ergänzung für 4.3 c. :
fi_t = (1/(2*pi))*diff((2*pi*ff(1)*tt)+  I_t.*cos((2*pi*ff(2)*tt)-(pi/2)) -
(pi/2))./diff(tt);
```

Funktion zum Erzeugen des Glocken-Tons.

4.3 The Bell Sound

Definition des 6 Fälle:

```
[xx1, fi_t1, tt1]=bell([110,220],10,2,6,11025);
[xx2, fi_t2, tt2]=bell([220,440],5,2,6,11025);
[xx3, fi_t3, tt3]=bell([110,220],10,12,3,11025);
[xx4, fi_t4, tt4]=bell([110,220],10,0.3,3,11025);
[xx5, fi_t5, tt5]=bell([250,350],5,2,5,11025);
[xx6, fi_t6, tt6]=bell([250,350],3,1,5,11025);
```

(a)

Abspielen dieser Töne:

```
sound(xx1,11025)
pause
sound(xx2,11025)
pause
sound(xx3,11025)
pause
sound(xx4,11025)
pause
sound(xx5,11025)
pause
sound(xx6,11025)
```

(b)

Die Fundamentalfrequenz ist der Größte gemeinsame Teiler der beiden Frequenzen. (Trägerfrequenz und Modularfrequenz)

```
X1=gcd(110,220)
X2=gcd(220,440)
X3=gcd(110,220)
X4=gcd(110,220)
X5=gcd(250,350)
X6=gcd(250,350)
```

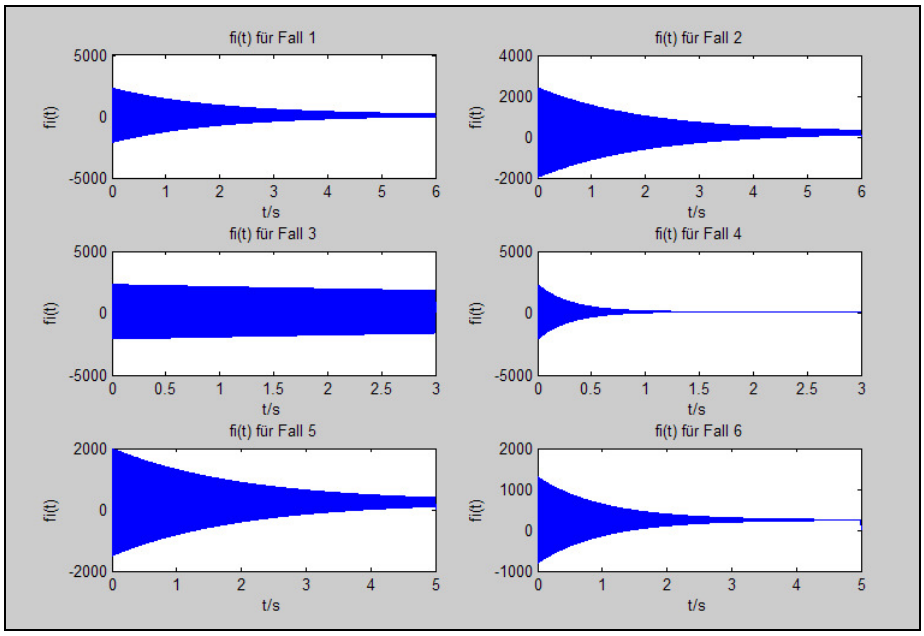
Es folgt:

```
X1 =   110
X2 =   220
X3 =   110
X4 =   110
X5 =    50
X6 =    50
```

(c)

Man kann hören, dass die Anzahl der beteiligten Frequenzen mit zunehmender Zeit abnimmt.

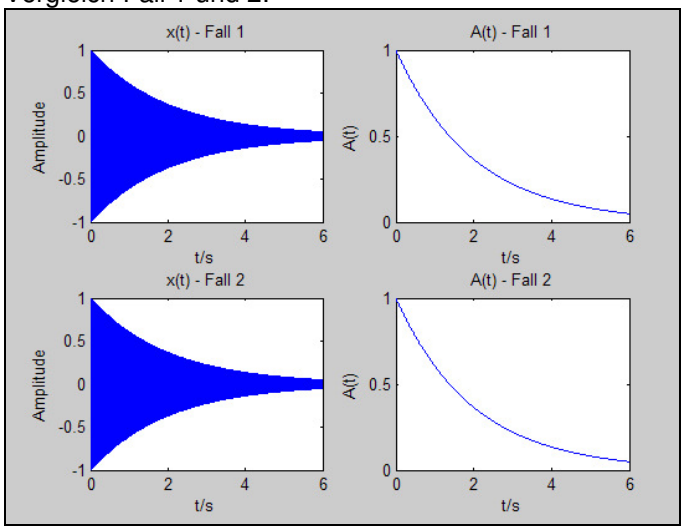
Vergleich der 6 Fälle:



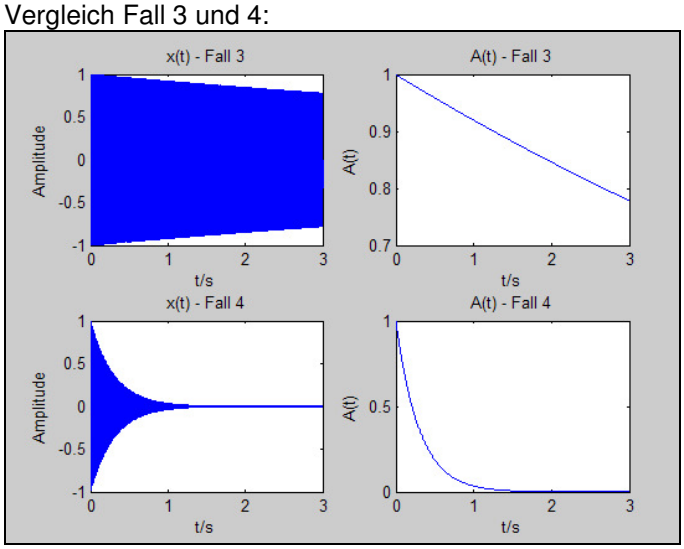
(e)

Man erkennt, dass die A(t) ähnlich aussieht, wie I(t). Nur, dass A(t) von 1 bis nach 0 läuft und nicht wie I(t) mit einem Faktor multipliziert wird. A(t) bestimmt außerdem die Amplitude des „Bell Sound“.

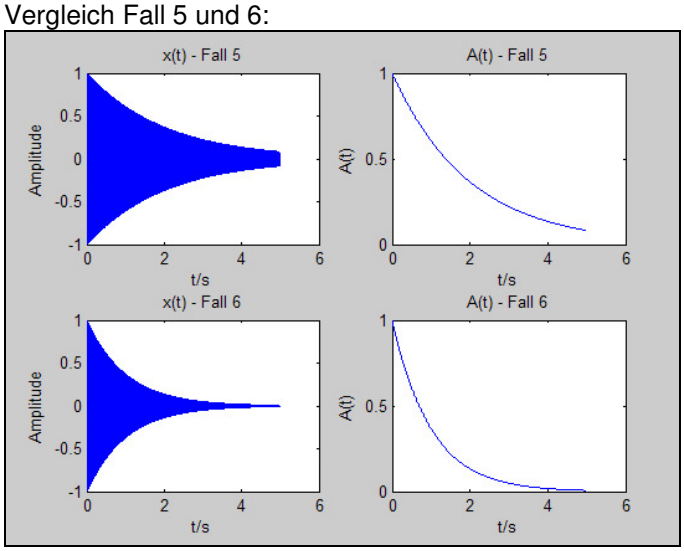
Vergleich Fall 1 und 2:



Vergleich Fall 3 und 4:

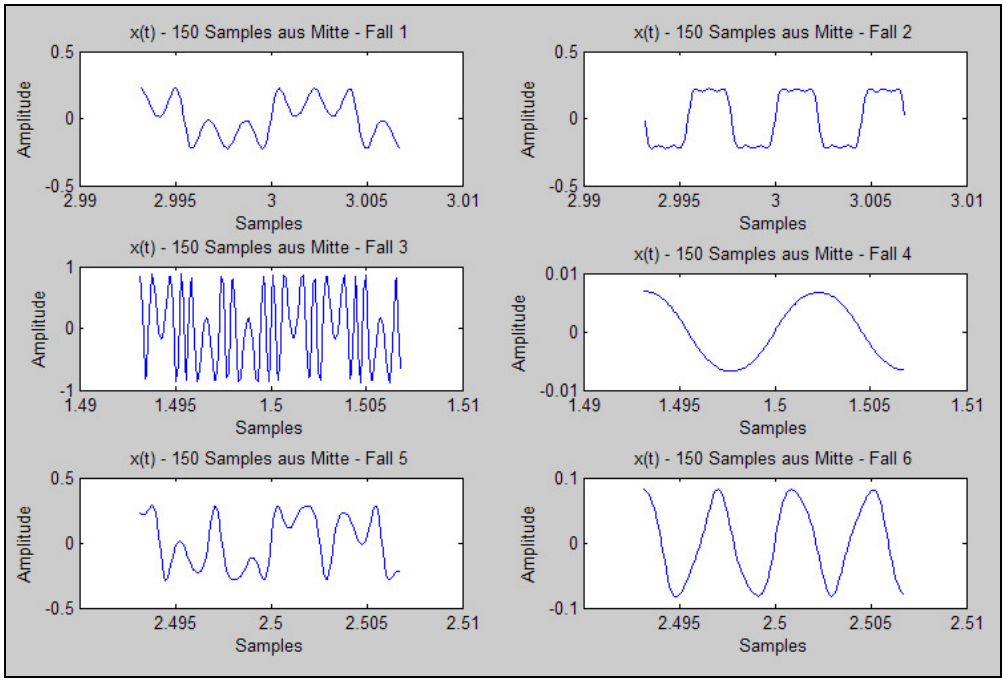


Vergleich Fall 5 und 6:



(f)

Vergleich von 100 Samples aus der Mitte der Signale:



Man erkennt, dass je kleiner die Zeitkonstante Tau ist, desto „schöner“ das Signal. (z.b. Fall 4)
Wenn Tau zunimmt (wie bei Fall 3) dann gibt es Überlappungen der Obertöne und es entsteht ein eher „unsauberes“ Signal.

4.4 Comments about the Bell

- Tests mit f_c / f_m -Verhältnis 1:2 bei verschiedenen Fundamentalfrequenzen:

Fall 3:

```
[xx3a, fi_t3a, tt3]=bell([110,220],10,12,3,11025);
[xx3b, fi_t3b, tt3]=bell([320,640],10,12,3,11025);
[xx3c, fi_t3c, tt3]=bell([510,1020],10,12,3,11025);

sound(xx3a,11025)
pause
sound(xx3b,11025)
pause
sound(xx3c,11025)
```

```
pause
```

Fall 4:

```
[xx4a, fi_t4a, tt4]=bell([110,220],10,0.3,3,11025);
[xx4b, fi_t4b, tt4]=bell([310,620],10,0.3,3,11025);
[xx4c, fi_t4c, tt4]=bell([510,1020],10,0.3,3,11025);

sound(xx4a,11025)
pause
sound(xx4b,11025)
pause
sound(xx4c,11025)
pause
```

Je höher die Fundamentalfrequenz wird, desto höher und schärfer erscheint der Ton. Außerdem nimmt durch ein hohes Tau die Abklingdauer zu.

- Test mit verschiedenen f_c / f_m -Verhältnissen:

```
% 1:2 bei f0 = 110 Hz
[xx3d, fi_t3d, tt3]=bell([110,220],10,12,3,11025);
% 3:4 bei f0 = 110 Hz
[xx3e, fi_t3e, tt3]=bell([330,440],10,12,3,11025);
% 5:7 bei f0 = 110 Hz
[xx3f, fi_t3f, tt3]=bell([550,770],10,12,3,11025);
% 7:5 bei f0 = 110 Hz
[xx3g, fi_t3g, tt3]=bell([770,550],10,12,3,11025);
% 3:8 bei f0 = 110 Hz
[xx3h, fi_t3h, tt3]=bell([330,380],10,12,3,11025);

sound(xx3d,11025)
pause
sound(xx3e,11025)
pause
sound(xx3f,11025)
pause
sound(xx3g,11025)
pause
sound(xx3h,11025)
pause
```

Es zeigen sich deutliche Unterschiede bei dem Klang. Dieser entfernt sich zunehmend vom charakteristischen „Bell-Sound“.

DSP First Laboratory Exercise #5

„FIR Filtering of Sinusoidal Waveforms“

1 Overview

Definition FIR-Filter:

$$y[n] = \sum_{k=0}^{M} b_k \cdot x[n-k]$$

1.1 Frequency Response of FIR Filters

Definition der Frequenzantwortfunktion:

$$H(e^{j\hat{\omega}}) = \sum_{k=0}^{M} b_k e^{-j\hat{\omega}k}$$

2 Warm-up

2.1 Frequency Response of the Three-Point Averager

(a)

Siehe Handschriftliche Unterlagen im Anhang

(b) & (c)

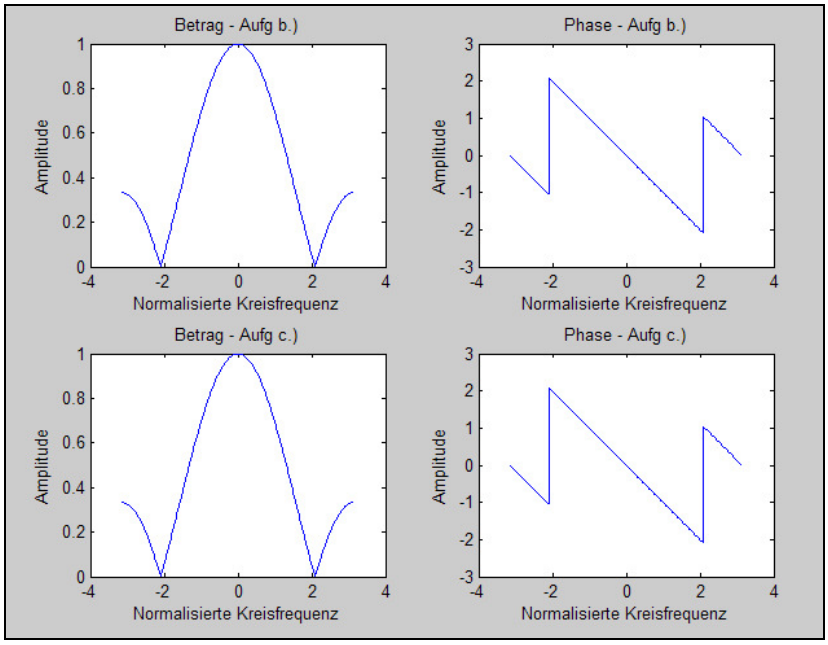
Direkte Implementierung der Funktion aus (a) und Vergleich mit einer mit Hilfe von `freqz` berechneten Frequenzantwortfunktion:

```
ww=-pi:(pi/200):pi;

H1=((2*cos(ww)+1)/3).*exp(-j*ww);
subplot(2,2,1)
plot(ww,abs(H1)), title('Betrag - Aufg b.)');
subplot(2,2,2)
plot(ww,angle(H1)), title('Phase - Aufg b.)');


bb= 1/3*ones(1,3);
H2= freqz(bb,1,ww);

subplot(2,2,3)
plot(ww,abs(H2)), title('Betrag - Aufg c.)');
subplot(2,2,4)
plot(ww,angle(H2)), title('Phase - Aufg c.)');
```



3 Lab: FIR Filters

3.1 Filtering Cosine Waves

Betrachtung der zeitdiskreten Funktion der Form:
$x[n] = A \cdot \cos(\hat{\omega} n + \varphi)$ für n=0,1,2…,L-1
$\hat{\omega}$ ist definiert als $0 \le \hat{\omega} \le \pi$

Wenn die zeitdiskrete Funktion aus der Abtastung eines zeitkontinuierlichen Signals erfolgt, gilt $\hat{\omega} = \omega T_s = 2\pi / f_s$

3.2 First Difference Filter

Generierung des Signals:

```
A=7;
Fi=pi/3;
w=0.125*pi;
n=0:49;

xx=A*cos(w*n+Fi);

bb=[5 -5];
yy=firfilt(bb,xx);
```

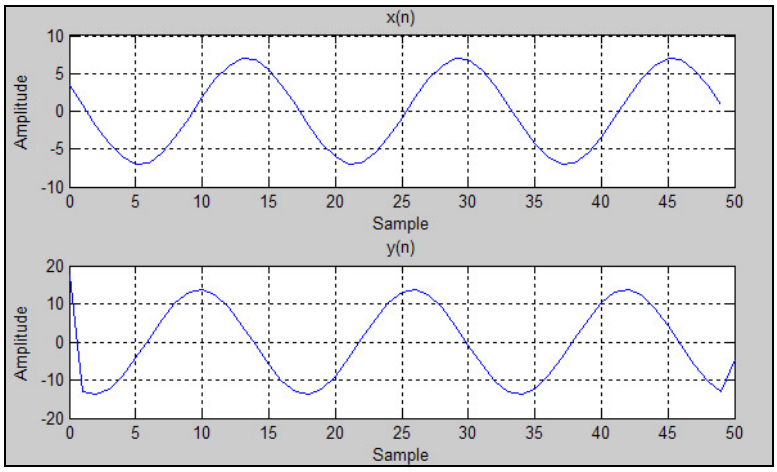
(a)

Es gilt: `length(yy)=length(xx)+length(bb)-1`
In diesem Fall also: 51 = 50 + 2 - 1

Die Ausgabesequenz kann nämlich bis zu M - Werte mehr umfassen als die Eingabesequenz.

(b)

Plot von Ein- und Ausgabe:



(c)

Ermittlung Amplitude und Phase von x[n] aus dem Plot:

Durch Ablesen:
Amplitude = 7

Rechnerisch ergibt sich für die Phase:

$$\varphi = \frac{-2\pi}{T} t = \frac{-2\pi}{16} \cdot (-3) = 0.375\pi$$

(d)

Denn Grund für den anfänglichen anderen Verlauf der Kurve findet man in der Formel für $y(n)$ - man rechnet nämlich für $y(0) = 5x(0) - 5x(-1)$ und dabei ist x(-1) nicht definiert, und wird als ‚0' gewertet.

(e)

Amplitude = 14

Phase Rechnerisch:
Man erkennt eine Phasenverschiebung um -6 Punkte, bei T=16

Daraus folgt: $\varphi = \frac{-2\pi}{16} \cdot (-6) = 0.75\pi$

Die Frequenz ist: $\frac{1}{T} = \frac{1}{16s} = 0.0625Hz$

(f)

Die Relative Amplitude beträgt: $A_{rel} = \frac{A_y}{A_x} = \frac{14}{7} = \frac{2}{1}$

Die Relative Phase beträgt: $\varphi_{rel} = \varphi_y - \varphi_x = 0.75\pi - 0.375\pi = 0.375\pi$

(g)

Vorgegebenes Signal: $x[n] = e^{j\hat{\omega}n}$

Vorgegebene Frequenz: $\hat{\omega} = 0.125\pi$

Die Frequenzantwortfunktion ist dabei: $H(e^{j\hat{\omega}}) = \sum_{k=0}^{M} b_k e^{-j\hat{\omega}k} = 5 - 5^{j \cdot 0.125\pi}$

Damit gilt:
Amplitudengang:

```
abs(5-5*exp(-j*0.125*pi))
ans = 1.9509
```

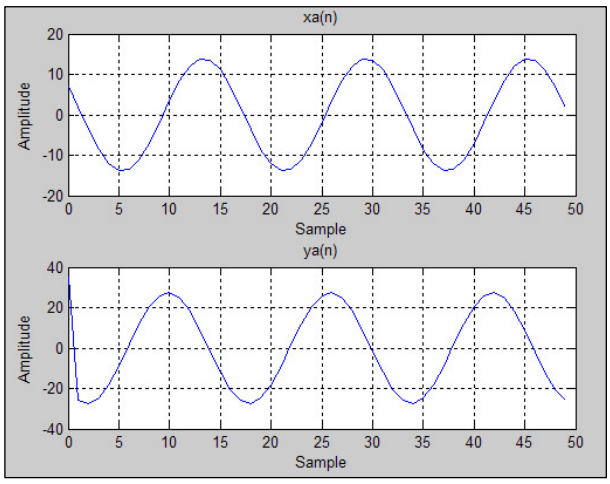
Phasengang:

```
angle(5-5*exp(-j*0.125*pi))
ans = 1.374
```

3.3 Linearity of the Filter

(a)

```
xa=2*xx;
ya=firfilt(bb,xa);
subplot(2,1,1);
plot(n,xa), title('xa(n)');
subplot(2,1,2);
plot(n,ya(1:50)), title('ya(n)');
```



Die Relative Amplitude beträgt: 2
Und die Relative Phase: 0.375π

Beide ändern sich also nicht, da derselbe Filter verwendet wird.

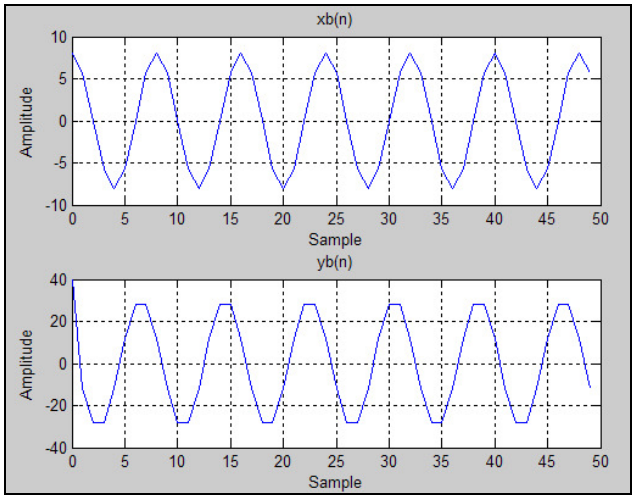
(b)

Es wird eine neue Eingabe definiert:
$x_b[n] = 8\cos(0.25 \cdot \pi \cdot n)$

```
xb=8*cos(0.25*pi*n);
yb=firfilt(bb,xb);

subplot(2,1,1);
plot(n,xb), title('xb(n)');

subplot(2,1,2);
plot(n,yb(1:50)), title('yb(n)');
```



Die Relative Amplitude beträgt: $A_{rel} = \frac{A_y}{A_x} = \frac{30}{8} = 3.75$

Außerdem beträgt die Relative Phase: $\varphi_{rel} = \varphi_y - \varphi_x = \frac{3}{8}\pi - 0 = 0.375\pi$

Man erkennt, dass erst das gefilterte Signal verschoben ist. Es weißt außerdem eine größere Amplitude auf. Die Periodendauer stimmt dagegen bei beiden überein.

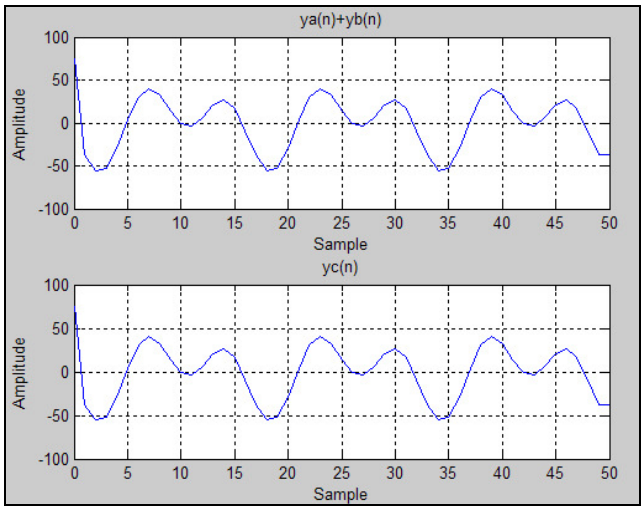
(c)

Vergleich:

```
xc=xa+xb;
yc=firfilt(bb,xc);

subplot(2,1,1);
plot(m,ya+yb), title('ya(n)+yb(n)');

subplot(2,1,2);
plot(m,yc), title('yc(n)');
```



Beide Plots erscheinen gleich. Es zeigt sich also eine erkennbare Linearität. Es spielt keine Rolle. ob man zwei Signale zuerst addiert und dann filtert oder erst jedes Signal filtert und dann addiert.

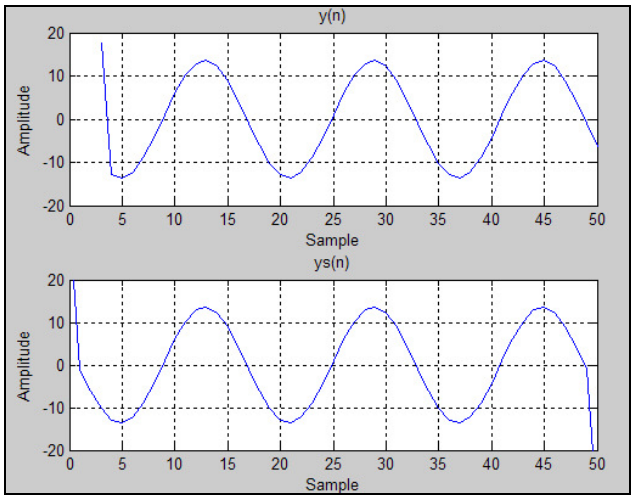
3.4 Time-Invariance of the Filter

Verschiebung bzw. Zeitverzögerung:

```
xs=7*cos(0.125*pi*(n-3)+(pi/3));
ys=firfilt(bb,xs);

subplot(2,1,1);
plot(m+3,yy), title('y(n)');

subplot(2,1,2);
plot(m,ys), title('ys(n)');
```



Es zeigt sich also zusätzlich noch eine Zeitinvarianz. So spielt es dann keine Rolle, ob ein Signal vor oder nach der Filterung verzögert wird.

3.5 Cascading Two Systems

Die Kaskadierung verschiedener Filter-Systeme wird getrachtet. Speziell in diesem Fall ein Quadrierer und ein Zeitdiskreter Filter.

(a)

Das System wird beschrieben als:

$$w[n] = (x[n])^2$$
$$y[n] = w[n] - w[n-1]$$

Implementierung:

```
bb=[1 -1];

ww=xx.^2;
yy=firfilt(bb,ww);
```

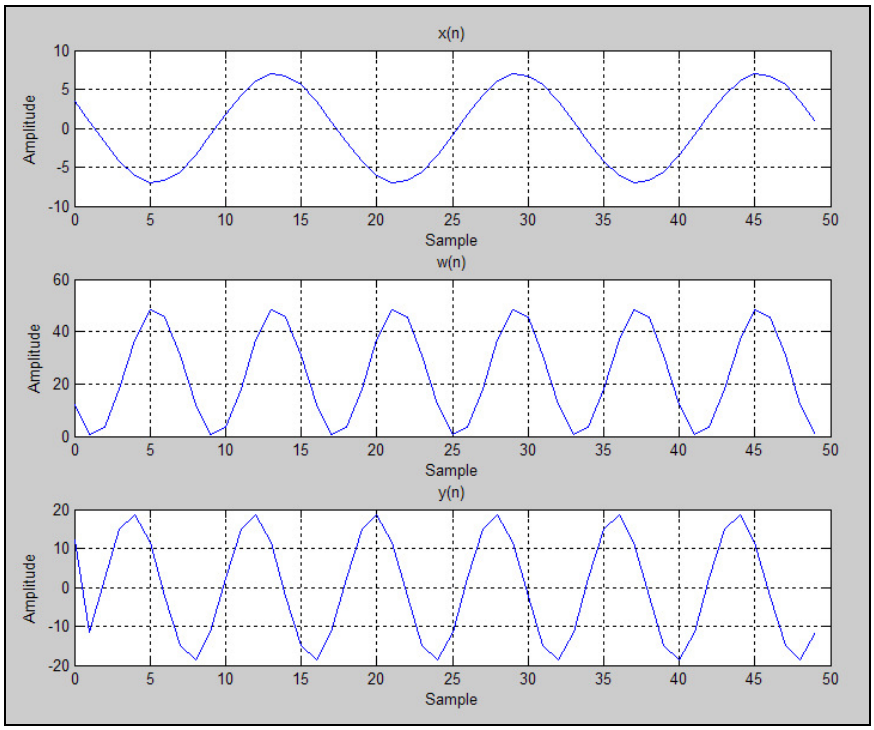
(b)

Plot aller drei Signale:

```
subplot(3,1,1);
plot(n,xx(1:50)), title('x(n)');

subplot(3,1,2);
plot(n,ww(1:50)), title('w(n)');

subplot(3,1,3);
plot(n,yy(1:50)), title('y(n)');
```



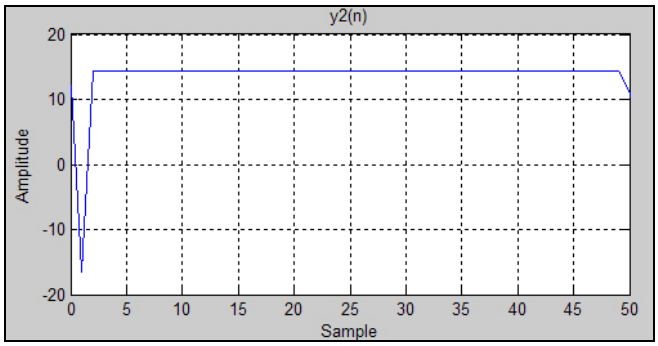
(c) – (f)

Siehe Handschriftliche Unterlagen.

Ergänzung zu (f):

Betrachtung eines neuen Filters, der Form: $y_2[n] = w[n] - 2\cos(0.25\pi)w[n-1] + w[n-2]$

```
bb2=[1 -2*cos(0.25*pi) 1];
y2=firfilt(bb2,ww);
mm=0:51;
plot(mm,y2), title('y2(n)');
```



Durch den Filterkoeffizienten $-2\cos(0.25\pi)$ werden bei der Filterung Frequenzanteile weggefiltert, so dass nur ‚0' als Frequenzanteil übrig bleibt.