

Information Retrieval for Music and Motion

Meinard Müller

Max-Planck-Institut für Informatik
Campus E1 4, 66123 Saarbrücken, Germany
meinard@mpi-inf.mpg.de

Informations- und Übungsblatt zu MATLAB

Dieses Informations- und Übungsblatt soll mit dem Umgang der MATLAB-Software anhand einiger Beispiele vertraut machen. Es handelt sich dabei nicht um eine systematische Einführung. Insbesondere der Umgang mit den Grundfunktionen der *Signal Processing Toolbox* soll eingeübt werden.

1 Allgemeines

Rufen Sie mit `help` das Hilfsmenü auf und verschaffen Sie sich einen Überblick über die verschiedenen Themen. Mit `help topic` erhalten Sie mehr Informationen zum jeweiligen Thema `topic`. Informieren Sie sich u. a. über die folgende Themen:

<code>audio</code>	Audio support.
<code>elfun</code>	Elementary math functions.
<code>elmat</code>	Elementary matrices and matrix manipulation.
<code>matfun</code>	Matrix functions - numerical linear algebra.
<code>ops</code>	Operators.
<code>paren</code>	Parentheses, braces, and brackets.
<code>signal</code>	Signal Processing Toolbox.
<code>sigdemos</code>	Signal Processing Toolbox Demonstrations.
<code>sigtools</code>	Filter Design and Analysis Tool (GUI).
<code>sptoolgui</code>	Signal Processing Toolbox GUI.

Über den Menüpunkt „Help→MATLAB Help“ können Sie die Hilfeinformationen mittels einer grafischen Oberfläche durchsuchen. Eine weitere Quelle für Informationen ist die Homepage www.mathworks.com des Herstellers Mathworks, auf der Sie auch Tutorials zu unterschiedlichen Themen finden können.

Probieren Sie weiterhin die folgenden wichtigen, allgemeinen MATLAB-Funktionen aus.

<code>ans</code>	Most recent answer.
<code>cd</code>	Change current working directory.
<code>clear</code>	Clear variables and functions from memory.
<code>demo</code>	Run demonstrations.
<code>dir</code>	List directory.
<code>doc</code>	Display HTML documentation in the Help browser.
<code>guide</code>	Open the GUI Design Environment.
<code>help</code>	On-line help, display text at command line. <code>help</code> , by itself, lists all primary help topics.
<code>help topic</code>	Information concerning the topic, e.g., “ <code>help elmat</code> ”.
<code>lookfor keyword</code>	Keyword search all M-files for keyword, e.g., “ <code>lookfor Fourier</code> ”.
<code>more</code>	Control paged output in command window.
<code>open file.m</code>	Opens m-file, e.g., “ <code>open fftfilt.m</code> ”.
<code>quit</code>	Quit MATLAB session.
<code>what</code>	List M-files in the current directory.
<code>who</code>	List current variables.
<code>whos</code>	List current variables, long form.

2 Rechnen mit Vektoren und Matrizen

Aufgabe 2.1.

Machen Sie sich die Funktionsweise der folgenden Befehlszeilen (mit ihren Ausgaben) klar.

```
>> a = 0:0.1:0.55
a =    0    0.1000    0.2000    0.3000    0.4000    0.5000

>> a = 1:10
a =     1     2     3     4     5     6     7     8     9    10

>> b = [1 zeros(1,2)]
b =     1     0     0

>> c = [a(3:5); b; ones(1,3)]
c =     3     4     5
       1     0     0
       1     1     1

>> c'
ans =     3     1     1
        4     0     1
        5     0     1

>> c*c
ans =    18    17    20
        3     4     5
        5     5     6

>> c.*c
ans =     9    16    25
        1     0     0
        1     1     1

>> c(2,:)=[ ]
c =     3     4     5
        1     1     1

>> d = repmat(c,2,3)
d =     3     4     5     3     4     5     3     4     5
        1     1     1     1     1     1     1     1     1
        3     4     5     3     4     5     3     4     5
        1     1     1     1     1     1     1     1     1
```

Aufgabe 2.2.

Folgende Befehle sind nützlich zum Sortieren und Thresholding von Folgen. Benutzen Sie die `help`-Funktion für weitere Einzelheiten zu den vorkommenden Befehlen.

```
>> x = rand(1,8)
x = 0.3311  0.5629  0.4206  0.4007  0.0548  0.0213  0.4543  0.0578

>> y=sort(x)
y = 0.0213  0.0548  0.0578  0.3311  0.4007  0.4206  0.4543  0.5629

>> abs(x)<0.4
ans = 1      0      0      0      1      1      0      1

>> i = find(abs(x)<0.4)
i = 1      5      6      8

>> x(i) = zeros(size(i))
x = 0      0.5629  0.4206  0.4007  0      0      0.4543  0
```

Aufgabe 2.3.

Machen Sie sich im Stile der beiden letzten Aufgaben die folgenden, stichwortartig angeordneten Funktionalitäten von MATLAB klar:

- Pfeiltasten \uparrow und \downarrow : Command history durchblättern.
- Befehlspräfix in der Kommandozeile zur eingeschränkten History-Suche.
- MATLAB als Taschenrechner: $3*2/3-5^2$
- Formate (`help datatypes`): `double`, `single`, `int8`, `int 16`, `unit8`, `struct`, `char`, `cell`, ...
- Variablen (beachten Sie Groß- und Kleinschreibung): `x=10`, `X='test'`
- Typen: `int`, `real`, `complex`, `inf (1/0)`, `NaN`
- Funktionsweise von Komma (,) und Semikolon (;)
- Funktionen (`help elfun`): `sin`, `cos`, `sin`, `tan`, `exp`, `acos`, `atan`, `log`, `log10`, `sinc`, `sqrt`, ...
- Allgemeine Funktionen: `who`, `whos`, `clear var`, `clear all`, `load`, `save`
- Spaltenvektoren: `v=[1 2 3]`, `w=[3:0.5:4 0:0.1:0.45]` (\rightarrow `help colon`)
- Zeilenvektoren: `v = [1; 2; 3]`
- Transposition: `v=w'`
- Länge eines Vektors: `length(v)`

- Leerstellen sind in MATLAB wichtig: `v1=[0+ 1 2 3]`, `v2=[4 +5 6]`, `v3=[4+5 6]`
- Verkettung: `w1=[v1 v2]`, `w2=[v1;v2]`, `w3=[v1' v2']`
- Sortieren: `sort(v)`
- Zugriff auf Vektorelemente: `v=(1:10)`; `v(2)`, `v(2:2:5)`
- Addition: `v1+v2`
- Skalarmultiplikation: `3*v1`
- Skalarprodukt: `v1*v2'` oder `dot(v1,v2)`
- Punktweise Operationen: `v1.*v2`, `v1./v2`, `v1.^2`,...
- Norm: `norm(v1)` oder `sqrt(v1*v1')`
- Faltung: `conv(v1,v2)`
- Vektoren komplexer Zahlen: `x=[1+3i, 2-2i]`
- Komplexe Konjugation und Transposition: Vergleiche `x'` und `x.'`
- Funktionen im Zusammenhang mit komplexen Zahlen: `real`, `imag`, `isreal`, `conj`, `abs`, `angle`
- Matrizen: `A=[1 2 3 4; 5 6 7 8; 1 2 3 4; 8 7 6 5]`
- Dimensionen einer Matrix: `size(A)`
- Funktionen im Zusammenhang mit Matrizen: `eye`, `zeros`, `ones`, `diag`, `rand`, `inv`, `convn`, ...
- Darstellung von Matrizen: `K=[diag(1:4) A; A' zeros(4,4)]`, `spy(K)`
- Zugriff auf Matrixelemente: `A(3,3)=A(15)+3*A(2,2)`, `A(:,2:3)`
- Dünne Matrizen: `i=[1,2,5]`; `j=[2,3,4]`; `v=[10 11 12]`; `S=sparse(i,j,v)`, `T=full(S)`
- Funktionen zum Lesen und Schreiben: `fopen`, `fclose`, `fscan`, `fprintf`, `wavread`, `wavwrite`, ...
- Schleifen: `for`, `while`
- Bedingung: `a=rand(1);b=rand(1);if a<b b, elseif a==b '='', else a, end`
- Datentyp bool: `x=-1:0.05:1;plot(x,1.*(x>=0));axis([-1 1 -0.2 1.2]);`
- Weitere MATLAB-Funktionen: `round`, `fix`, `floor`, `ceil`, `sign`, `rem`, `mod`, `sum`, `max`, `min`, `find(y>0)`...
- Plot-Befehle: `plot`, `plot3`, `mesh`, `surf`, ...
- M-files
- Eigene Funktionsdeklarationen (\rightarrow `help function`)

3 Signale, Fouriertransformation, Spektrogramm

Aufgabe 3.1.

Nachdem Sie sich mit den Grundlagen vertraut gemacht haben, legen Sie ein Arbeitsverzeichnis an und wechseln auf der Kommandozeile von MATLAB in dieses Verzeichnis (\rightarrow `cd`). Kopieren Sie die auf der Webseite zu den Übungen bereitgestellten Wav-Dateien in dieses Verzeichnis und bearbeiten Sie die folgenden Punkte:

- Lesen Sie die Wav-Datei mit dem Namen „ding8000.wav“ (\rightarrow `wavread`).
- Welche Abtastrate hat das Audiosignal?
- Hören Sie sich das Audiosignal mit der richtigen Abtastrate an (\rightarrow `sound`).
- Wie lang (in Sekunden) ist das Audiosignal?
- Weisen Sie der Variablen L die Abtastwerte des linken Kanals und der Variablen R die Abtastwerte des rechten Kanals zu.
- Stellen Sie den linken und den rechten Kanal in zwei Koordinatensystemen in einer Abbildung (`figure`) graphisch dar (\rightarrow `plot`, `subplot`).
- Stellen Sie in einer neuen Abbildung beide Kanäle in einem Koordinatensystem und verschiedenen Farben übereinander dar (\rightarrow `figure`, `hold`, `plot`).
- Stellen Sie das Differenzsignal der beiden Kanäle des Audiosignals graphisch dar.
- Versetzen Sie die Achsen mit entsprechenden Bezeichnungen (\rightarrow `get`, `set`, `gcf`, `gca`, `xlabel`, `ylabel`, `title`).

Aufgabe 3.2.

Betrachten Sie folgendes MATLAB-Programm:

```
t = (0:1/100:2-1/100);  
x = sin(2*pi*8*t) + sin(2*pi*20*t);  
y = fft(x);  
w = (0:length(y)-1)*100/length(y);  
subplot(2,2,1); plot(t,x);  
subplot(2,2,2); plot(w,abs(y));  
subplot(2,2,3); plot(w,angle(y));  
subplot(2,2,4); plot(w,unwrap(angle(y)));
```

Inwieweit ist es gerechtfertigt, y als Spektrum von x zu bezeichnen? Beantworten Sie dabei die folgenden Fragen:

- Welche Fourierkoeffizienten werden von y repräsentiert (siehe auch Abbildung 1)?
- Warum ist y symmetrisch?
- Was ist die Nyquist-Frequenz von x ?
- Welche Bedeutung haben $\text{abs}(y)$, $\text{angle}(y)$ und $\text{unwrap}(\text{angle}(y))$?

Ersetzen Sie x durch andere Signale (z. B. durch die Wav-Dateien auf der Webseite) und versuchen Sie, die Resultate zu interpretieren.

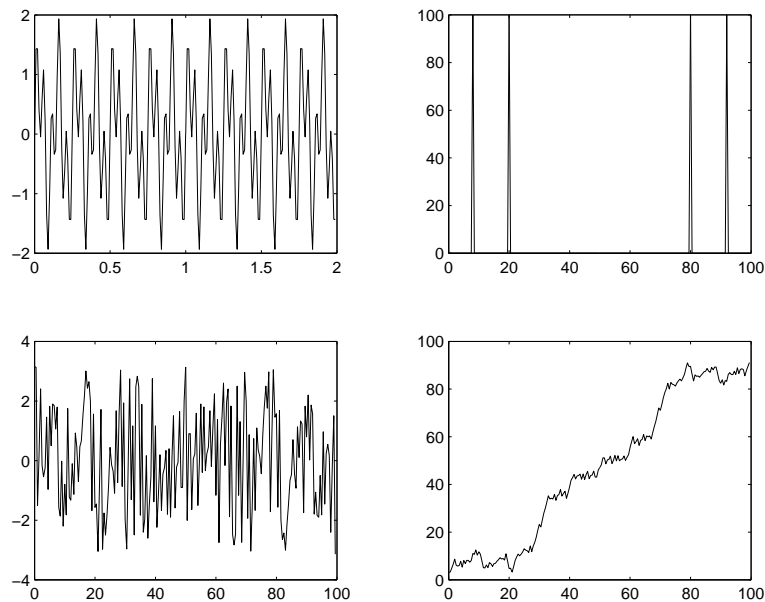


Abbildung 1: Aufgabe 3.2.

Aufgabe 3.3.

Das folgende MATLAB-Programm, welches Sie im Übungsverzeichnis unter dem Dateinamen „Aufgabe_WFT_chirp.m“ finden, berechnet das Spektrogramm eines abgetasteten Chirpsignals. Das so berechnete Spektrogramm kann als Approximation der gefensterten Fouriertransformation (WFT) angesehen werden. Beachten Sie, dass es bei der Berechnung sehr viele Wahlmöglichkeiten bei den Parametern gibt:

- Fensterbreite,
- Fensterfunktion,
- Überlappungsbreite,
- Größe der verwendeten FFT (Frequenzauflösung).

Machen Sie sich die Bedeutung der einzelnen Parameter klar. Schauen Sie mit `help specgram` nach, was die Funktion `specgram` eigentlich berechnet. Welche Interpretation haben die berechneten Werte? Verändern Sie die Parameter und testen Sie, ob das Resultat mit Ihren theoretischen Überlegungen übereinstimmt.

```
% Chirp signal and windowed Fourier transform
fs =1000;                %sampling rate
a=0;b=3;                 %time interval
o = 400;                 %frequency of chirp function at time 1
t = a:1/fs:b ;           %time interval [a,b]
f = sin(o*pi*t.*t);      %computing chirp function

nfft = 32;               %size of FFT (nfft must be larger than window_size)
window_size = 16;        %size of window
window = hanning(window_size); %window function
overlap = 4;             %overlap of window functions

axes('Position',[0.1,0.8,0.8,0.1])
plot(t,f)
title('Chirp function f(t)=sin(400\pi t^2) and WFT with Hanning window')
axis([a,b,-1.4,1.4])
axes('Position',[0.1,0.1,0.8,0.63])
specgram(f,nfft,fs,window,overlap)
```

Diese Ausgabe des Programms finden Sie in Abbildung 2. Sie können übrigens die von MATLAB erzeugten Abbildung mit dem Befehl `print -deps filename.eps` als Eps-Datei abspeichern.

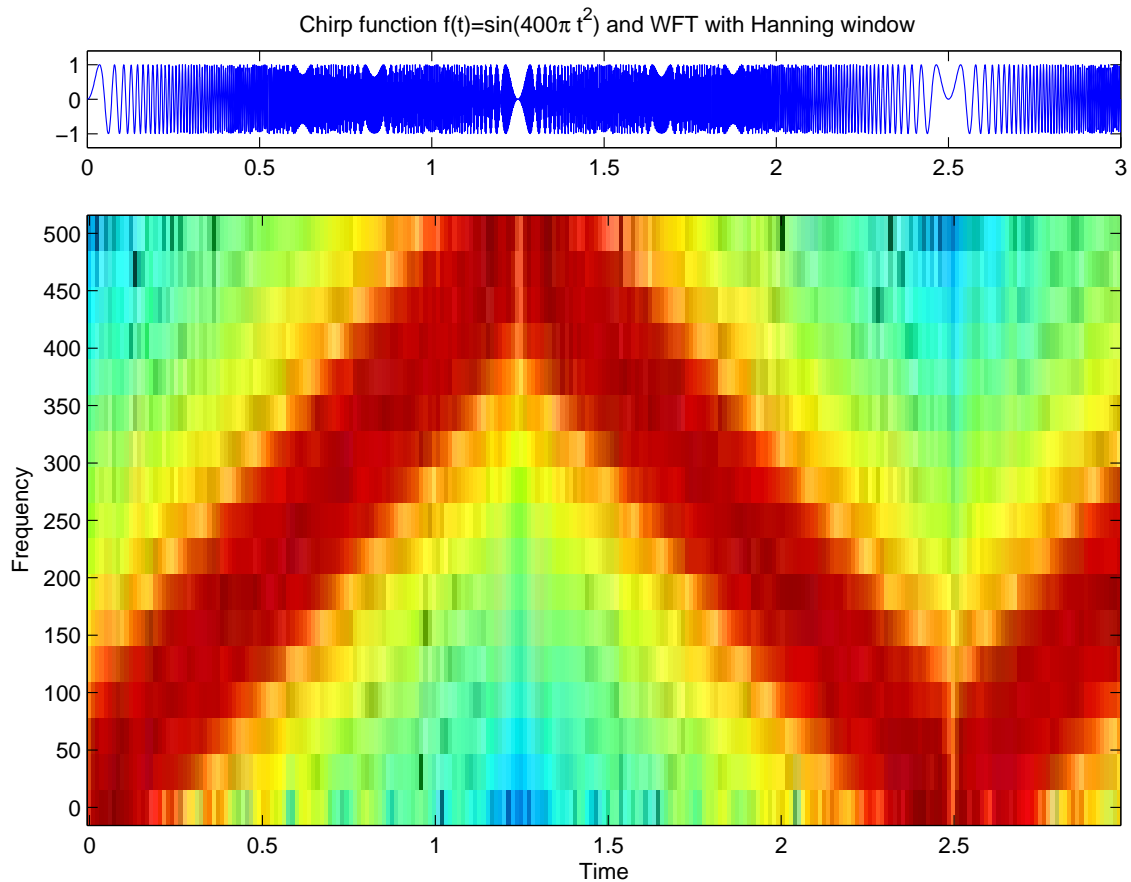


Abbildung 2: Aufgabe 3.3.

Aufgabe 3.4.

Die Momentanfrequenz eines Chirpsignals zum Zeitpunkt t ergibt sich als Ableitung der Phase zum Zeitpunkt t . Diskutieren Sie den Aliasingeffekt, der bei dem in Aufgabe 3.3 definierten abgetasteten Chirp-Signal auftritt. Wie spiegelt sich der Aliasingeffekt in dem in Abbildung 2 dargestellten Spektrogramm wider?

4 Filter

Aufgabe 4.1.

Im folgenden behandeln wir nur kausale Faltungsfilter, die in MATLAB-Notation durch einen Vektor $b = (b(1), \dots, b(m+1))$ der Länge $m+1$ und einen Vektor $a = (a(1), \dots, a(n+1))$ der Länge $n+1$ gegeben sind. Das Filter $h = (b, a)$ ist dann über die z-Transformation H von h wie folgt definiert:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b(1) + b(2)z^{-1} + \dots + b(m+1)z^{-m}}{a(1) + a(2)z^{-1} + \dots + a(n+1)z^{-n}}.$$

(Hierbei folgen wir der MATLAB-Konvention, dass die Indizierung der Vektoren mit dem Index 1 beginnt.) Testen Sie die folgenden für die Signalverarbeitung fundamentalen MATLAB-Funktionen:

- Faltung zweier Vektoren x und y : `z=conv(x,y)`
Welche Länge hat z in Bezug auf die Längen von x und y (\rightarrow `length`)?
- Filtern eines Signals x mit dem Filter $h = (b, a)$: `y=filter(b,a,x)`
- Frequenzantwort eines Filters: `[h,w] = freqz(b,a,n)`
Machen Sie sich die Funktionsweise dieses wichtigen Befehls am Beispiel des Haarfilter `h1=(b1,a1)` und des Differenzenfilters `h2=(b2,a2)` klar, welche durch die Koeffizienten `b1=1/2*[1 1]`, `a1=[1]`, `b2=1/2*[1 -1]` und `a2=[1]` gegeben sind.
 - Berechnen Sie mittels `freqz` eine Anzahl von Abtastwerten (z.B. $n = 100$) der Frequenzantworten `H1` und `H2` der beiden Filter. (Beachten Sie, dass die Abtastwerte nur approximativ berechnet werden!)
 - Berechnen Sie hieraus die jeweiligen Amplituden- und Phasengänge und stellen diese graphisch dar. Versehen Sie die Darstellungen mit den richtigen Achsenbeschriftungen. Beachten Sie insbesondere die richtige Interpretation der Frequenzachse.
 - Filtern Sie mit `h1` und `h2` ein Signal ihrer Wahl und stellen Sie die Ergebnisse graphisch dar.
- Gruppenlaufzeit (group delay) eines Filters: `[gd,w] = grpdelay(b,a,n)`
- Impulsantwort eines Filters: `[h,z] = impz(b,a)`

Aufgabe 4.2.

MATLAB stellt mehrere GUIs zur Visualisierung (\rightarrow `fvtool`), zum Filterentwurf (\rightarrow `fdatool`) und zur Signalverarbeitung (\rightarrow `sptool`) zur Verfügung. Machen Sie sich insbesondere mit den Funktionalitäten von `fvtool` und `fdatool` vertraut. Intern werden unter anderem folgende Befehle zum Filterentwurf aufgerufen:

FIR filter design.

- `fir1` - Window based FIR filter design - low, high, band, stop, multi.
`b = fir1(n,wn)`
- `fir2` - FIR arbitrary shape filter design using the frequency sampling method.
`b = fir2(n,f,a)`
- `fircls` - Constrained Least Squares filter design - arbitrary response.
- `fircls1` - Constrained Least Squares FIR filter design - low and highpass.
- `remez` - Optimal Chebyshev-norm FIR filter design.
- `remezord` - Remez design based filter order estimation.

IIR digital filter design.

- `butter` - Butterworth filter design.
- `cheby1` - Chebyshev Type I filter design (passband ripple).
- `cheby2` - Chebyshev Type II filter design (stopband ripple).
- `ellip` - Elliptic filter design.
- `maxflat` - Generalized Butterworth lowpass filter design.
- `yulewalk` - Yule-Walker filter design.

IIR filter order estimation.

- `buttord` - Butterworth filter order estimation.
- `cheb1ord` - Chebyshev Type I filter order estimation.
- `cheb2ord` - Chebyshev Type II filter order estimation.
- `ellipord` - Elliptic filter order estimation.

Eine Übersicht über weitere Methoden zum Filterentwurf erhält man mittels `help signal`.

Aufgabe 4.3.

Die einfachste Methode zur Konstruktion von FIR-Filtern mit linearer Phase ist das Abschneiden der (unendlichen) Impulsantwort des entsprechenden idealen Filters mit einer geeigneten Fensterfunktion. In MATLAB wird diese Methode zum Filterentwurf durch die Funktion `fir1` realisiert. In Abbildung 3 sind die Amplituden- und Phasenantworten verschiedener durch `fir1` erzeugter FIR-Filter dargestellt. (Als Voreinstellung wird dabei das *Hamming-Fenster* verwendet.) Die Abbildung wurde wie folgt erzeugt:

(i) `b = fir1(100,[0.3 0.4]); freqz(b,1,512)`

(ii) `b = fir1(256,[0.3 0.4 0.7 0.8]); freqz(b,1,512)`

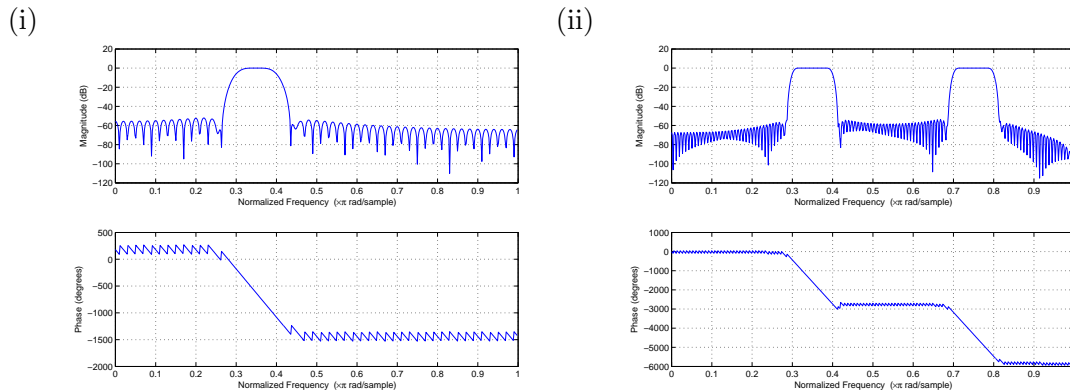


Abbildung 3: Aufgabe 4.3.

Machen Sie sich mit der Funktion `fir1` vertraut. Generieren Sie Tief-, Hoch- und Bandpassfilter unterschiedlicher Filterordnungen unter Verwendung verschiedener Fensterfunktionen und vergleichen Sie die Ergebnisse. Diskutieren Sie Nachteile des Filterentwurfs mittels `fir1` hinsichtlich folgender Punkte:

- Anzahl der Filterkoeffizienten,
- Größe der Gruppenlaufzeit,
- Spezifikation des Filters,
- Qualität des resultierenden Filters.

Weitere Befehle zum FIR-Filterentwurf sind unter anderem `fir2`, `firls` oder `remez`. Die Funktion `remez` erzeugt einen sogenannten „Minmax“- oder auch „Equiripple“-Filter, der eine wesentlich bessere Filterspezifikation zulässt.

Aufgabe 4.4.

Der wichtigste Vorteil von IIR-Filtern gegenüber FIR-Filtern ist, dass sich Filterspezifikationen im allgemeinen durch IIR-Filter mit wesentlich kleinerer Filterordnung realisieren lassen als mit vergleichbaren FIR-Filtern. In dieser Aufgabe behandeln wir exemplarisch den Filterentwurf mit den MATLAB-Funktionen `cheby2` und `cheb2ord`, durch die ein *Chebyshevfilter vom Typ II* erzeugt werden kann. Hierbei verfährt man in zwei Schritten:

- (1) Zuerst berechnet man mit der Funktion `cheb1ord` diejenige Ordnung n des Chebyshevfilters, die zur Erfüllung der vorgegebenen Filterspezifikationen benötigt wird.
- (2) Mit der Funktion `cheby1` wird dann der Chebyshevfilter der Ordnung n berechnet.

In Abbildung 3 sind die Amplituden- und Phasenantworten verschiedener Chebyshevfilter dargestellt, die von MATLAB berechnet wurden. Die Abbildung wurde wie folgt erzeugt:

- (i)

```
[n Wn]=cheb2ord([0.3 0.4],[0.29 0.41],1,50)
[b,a]=cheby2(n,50,Wn);freqz(b,a,512)
```
- (ii)

```
[n Wn]=cheb2ord([0.3 0.4],[0.299 0.401],1,50)
[b,a]=cheby2(n,50,Wn);freqz(b,a,512)
```

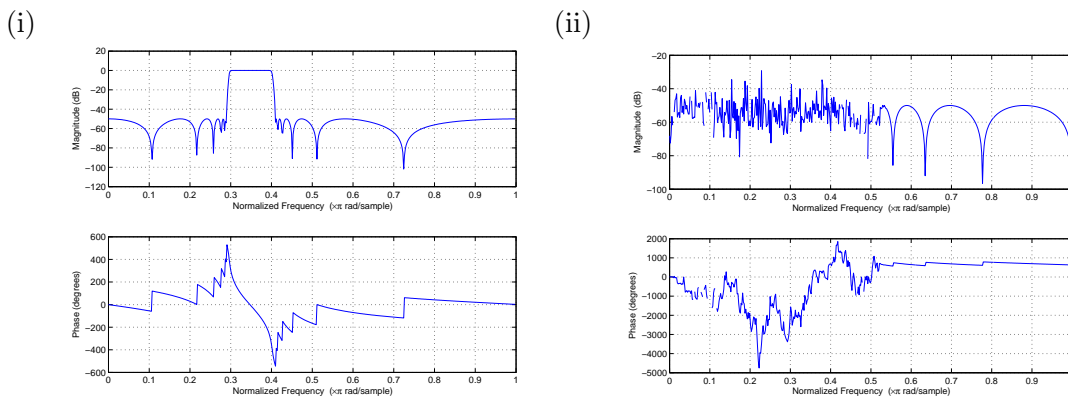


Abbildung 4: Aufgabe 4.4.

Im Fall (i) hat der Filter die Ordnung $n = 12$. Vergleichen Sie die Amplitudenantwort dieses IIR-Filters mit der Amplitudenantwort des FIR-Filters der Ordnung $n = 100$, die in (i) von Abbildung 3 dargestellt ist. Beachten Sie, dass der IIR-Filter keine lineare Phase hat.

Fall (ii) soll dokumentieren, dass die MATLAB-Funktionen `cheby2` und `cheb2ord` nicht für alle Filterspezifikationen die gewünschten Chebyshevfilter produzieren kann. In diesem Fall sind die Filterspezifikationen zu restriktiv, so dass die Verfahren versagen. (MATLAB gibt in diesem Fall jedoch keine Fehlermeldung aus!)

Überprüfen Sie also immer, ob die berechneten Filter auch tatsächlich die Filterspezifikationen erfüllen.

Aufgabe 4.5.

In dieser Übung geht es um die Änderung der Abtastrate eines Signals. Hierbei ist aufgrund des Aliasingeffekts äußerste Vorsicht geboten. Sei x ein Signal mit Abtastrate f_{sx} . Dann erzeugt

$$y = \text{resample}(x, p, q)$$

ein Signal y mit Abtastrate $f_{sy} = (p/q) f_{sx}$ (mit entsprechenden Rundungen, falls f_{sy} keine ganze Zahl ist). Hierbei wendet MATLAB automatisch einen FIR-Antialiasing-Filter vor dem eigentlichen Resampling an.

Führen Sie folgendes Experiment durch, um die Folgen von Aliasing hörbar zu machen:

- (1) Lesen Sie die Wav-Datei eines Audiosignals ihrer Wahl.
(Z.B. `[x,fsx] = wavread('Tonleiter.wav')`)
- (2) Halbieren Sie die Abtastrate mit dem Befehl `y=downsample(x,2)`. (Dieser Befehl arbeitet ohne vorherige Tiefpassfilterung!)
- (3) Hören Sie sich das Audiosignal y mittels `sound(y,fsx/2)` an.
- (4) Setzen Sie `x=y` und `fsx=fsx/2` und wiederholen Sie das Experiment ab Schritt (2).

Wiederholen Sie nun das Experiment, indem Sie in Schritt (2) vor dem Halbieren der Abtastrate einen geeigneten Tiefpassfilter anwenden. Machen Sie sich dabei klar, was in diesem Fall „geeignet“ überhaupt bedeuten soll. Entwerfen und testen Sie geeignete Tiefpassfilter mit den Filterentwurfsmethoden von Aufgabe 4.1.

Die folgenden Befehle werden von MATLAB zur Mutiraten-Signalverarbeitung zur Verfügung gestellt (\rightarrow `help signal`):

Multirate signal processing.

<code>decimate</code>	- Resample data at a lower sample rate.
<code>downsample</code>	- Downsample input signal.
<code>interp</code>	- Resample data at a higher sample rate.
<code>interp1</code>	- General 1-D interpolation. (MATLAB Toolbox)
<code>resample</code>	- Resample sequence with new sampling rate.
<code>spline</code>	- Cubic spline interpolation.
<code>upfirdn</code>	- Up sample, FIR filter, down sample.
<code>upsample</code>	- Upsample input signal.