5– Frequentieselectieve filters

Ing. Patrick Van Houtven

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

Frequentieselectieve filters

Doel van een filter is het kunnen manipuleren van signalen.

Een frequentieselectieve filter wordt gebruikt om ofwel bepaalde delen van een frequentiespectrum door te laten ofwel te blokkeren.

Er zijn vier typen van filters:

- Laagdoorlaatfilter
- Hoogdoorlaatfilter
- Banddoorlaatfilter
- Bandsperfilter

- Je kan deze filters zowel analoog als digitaal opbouwen. We beschouwen hier enkel de digitale filters.

Filter met- (IIR) en zonder tegenkoppeling (FIR)

FIR

Finite Impuls Response Filter

- Berekent de gefilterde waarde enkel met een eindig aantal vorige waarden van het inputsignaal.
- Er is geen terugkoppeling aanwezig van de output naar de input. Hierdoor is deze filter steeds stabiel.
- Het werkingsprincipe komt overeen met het bepalen van het gemiddelde in de vorige waarden die voorzien zijn van gewichten met een factor b_k.
- FIR heeft beduidend meer coëfficiënten nodig (hoger orde) dan de IIR-filter om dezelfde eigenschappen te bekomen
- Lineaire faseresponse
- Magnitude en fase kunnen bepaald worden onafhankelijk van elkaar.

IIR

Infinite Impulse Response of recursieve filters

- Hebben een terugkoppeling van de uitgang naar de ingang. Ten gevolge van deze terugkoppeling kunnen deze filters instabiel zijn.
- Hebben beduidend minder coëfficiënten nodig dan FIR om bepaalde filtereigenschappen, zoals demping en rimpel, te bekomen
- Door het feit dat ze weinig coëfficiënten nodig hebben vertonen deze filters maar een kleine signaalvertraging (wat een voordeel is bij online (real time)
- De groepsvertraging is niet constant
- IIR-filters zijn gelijkwaardig als de analoge filter
- Hebben een niet-lineaire faseresponse.

Transferfunctie

- Transferkarakteristiek van een analoog systeem wordt door differentiaalvergelijkingen beschreven.
- Transferkarakteristiek van een digitaal systeem wordt beschreven aan de hand van verschilvergelijkingen (difference-equations)
- Wat is een verschilvergelijking?
  - Een verschilvergelijking is een rekenregel waarbij de huidige waarde van de outputsequence y_k, en de huidige waarde van de input x_k en alle vorige waarden van de input- en output sequence worden gebruikt.
  - Lineaire verschilvergelijkingen met constante coëfficiënten zijn de meest belangrijkste.
  - De coëfficiënten hebben de benaming b_n en a_n.
- Een FIR verschilvergelijking bevat enkel de inputsignalen x_k en de coëfficiënten b_n .

$$\mathbf{y_k = b_0 x_k + b_1 x_{k-1} + \ b_2 x_{k-2} + \ \ldots . + b_n x_{k-n}}$$

- Een IIR verschilvergelijking bevat de inputsignalen x_k en de outputsignalen y_k en de coëfficiënten b_n en a_n.

$$\mathbf{y_k + a_1 y_{k-1} + a_2 y_{k-2} + \ \ldots + a_n y_{k-n} = b_0 x_k + b_1 x_{k-1} + \ b_2 x_{k-2} + b_n x_{k-n}}$$

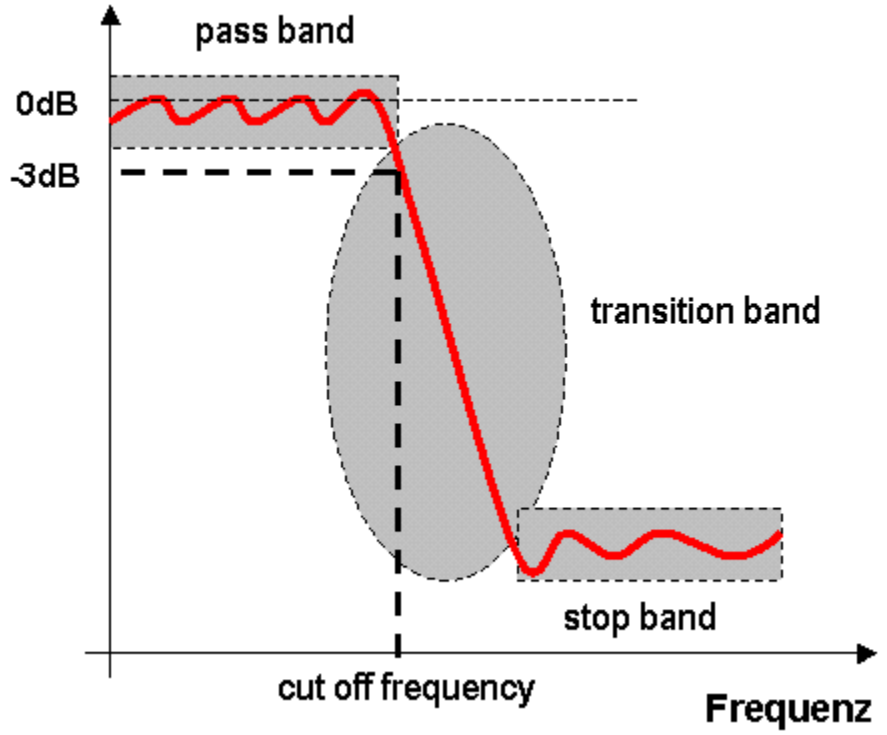
- Een IIR-filter heeft een gemeenschappelijke noemer in de transfertfunctie
- De FIR-filter heeft geen noemer (geen terugkoppeling) in zijn transfertfunctie

Vb. Transferfunctie

$$G_z = \frac{b_0 + b_1 z^{-1} + \ \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + \ \ldots + a_n z^{-n}}$$

Eigenschappen van digitale filters

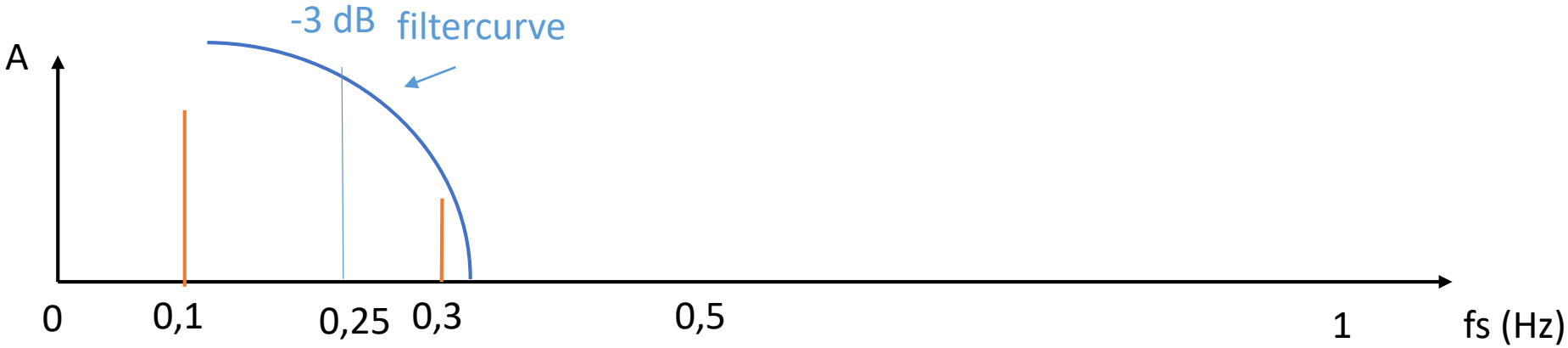
- ***Afsnijfrequentie (cutoff frequency)*** : Frequentie waarbij de amplitude -3 dB (0,707) lager ligt dan maximale amplitude in doorlaatband. Dit komt overeen met nog de helft van het vermogen
- ***Doorlaatband (pass band)*** : zou vlak moeten zijn om zo weinig mogelijk vervorming te bekomen (verschillende versterking van de frequenties die deel uit maken van het signaal)
- ***Transition band*** (overgang tussen doorlaat en sper) Deze zou zo klein mogelijk moeten zijn.
- ***Sperband (stop band)*** moet een zo hoog mogelijke dempings verhouding (hoge –dB-waarden) bevatten en eveneens vlak.
- ***Filterorde*** : hoe hoger de orde, hoe beter het filter
- ***Filtertype*** : LD, HD, bandoorlaat, bandsper
- ***Filter design procedure*** : butterworth, bessel , chebycheff, …



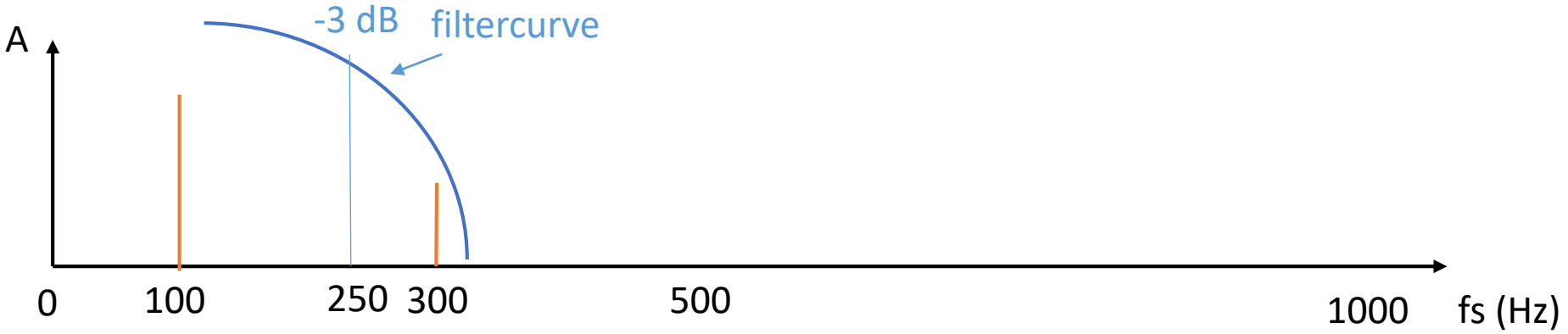
Sampling theorema en genormaliseerde frequenties

- Met een bepaalde samplefrequentie f_s kan je frequenties evalueren tot aan de nyquistfrequentie of $f_s/2$
- In digitale signaalprocessing wordt de samplefrequentie dikwijls genormaliseerd naar 1,0 Hz zodat de te evalueren frequenties liggen tussen 0,0 f_s en 0,5 f_s.
- Voorbeeld: stel dat de kritische frequentie (afsnijfrequentie) ligt op 0,25 f_s (of 0,25 Hz) dan is bij werkelijke samplefrequentie van 1 kHz de afsnijfrequentie gelijk aan 0,25 . 1 kHz of 250 Hz

Genormaliseerde samplefrequentie



Werkelijk frequentiespectrum met fs = 1 kHz



Filter toepassen op signalen

- Via de functie ***output = filter(num, den, input)*** kan je een digitale filter met transferfunctie (num/dem) toevoegen.
  - num staat voor numerator (teller)
  - dem staat voor denominator (noemer)
- Stel een eenvoudig FIR-filter, bijvoorbeeld een moving average filter waarbij de laatste twee ingangssignalen worden opgeteld en gedeeld door 2.
  - Coëfficiënten voor dit filter zijn 0,5 0,5. Deze staan in de teller (num)
  - Coëfficiënt noemer = 1 (De noemer van deze filter bevat enkel 1) => den = 1
  - In de functie filter geven we deze coëfficiënten in als vectoren en het ingangssignaal dat gefilterd moet worden.
  - Stel dat het ingangssignaal een random signaal is bestaande uit 50 waarden.
- De som van de coëfficiënten moet gelijk zijn aan 1. Op die wijze is de versterking steeds gelijk aan 1.
- Hogere ordes van filters kunnen bekomen worden door het aantal coëfficiënten te verhogen.

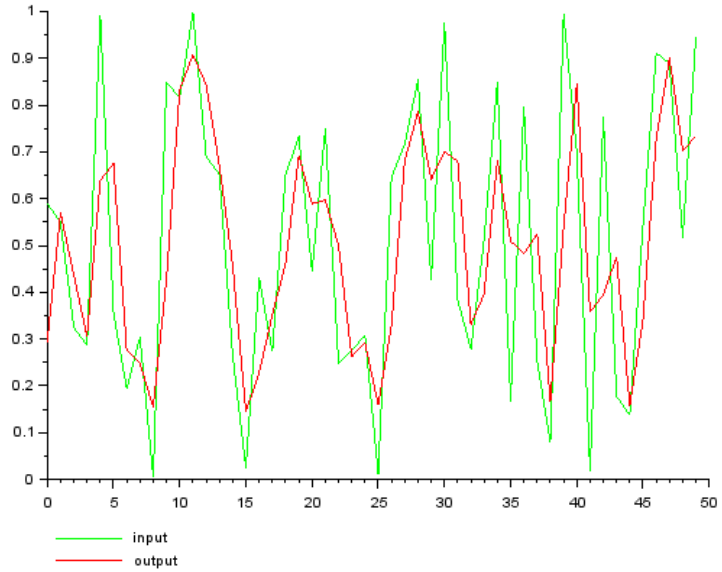
```
1  clf
2  //vb FIR-filter
3  n = [0 : 1 : 49]
4  // genereren inputsignaal (50 waarden random)
5  x = rand(50,1);
6  //definiëren FIR met twee coëfficiënten 0.5 0.5
7  //filter([(num)0.5,0.5],[(dem)1], (input) x)
8  y = filter([0.5, 0.5],[1],x)
9  //plot met legende en aanpassen kleurstijl
10 // plot horizontaal n (aantal waarden -- samples)
11 //plot vertikaal amplitude random waarden
12 plot2d (n,[x y],leg="input@output",...
13  style=[color("green"),color("red")]);
```

$$y_k = b_0 x_k + b_1 x_{k-1}$$

$$y_k = 0,5 x_k + 0,5 x_{k-1}$$

Transfertfunctie filter (FIR):

$$G_z = \frac{0,5z + 0,5z^{-1}}{1}$$



Filterontwerp

- Functies binnen Scilab voor het ontwerpen van FIR-filters (coëfficiënten)
  - ***wfir()*** : FIR-filter met lineaire faseresponse
  - ***filt()*** : coëfficiënten van een laagdoorlaatfilter
  - ***eqfir()*** : benadering van een FIR-filter
  - ***fsfirlin()*** : ontwerp via de samplingmethode

Filterontwerp : wfir()

- Met de functie wfir() kan je de filterparameters bepalen voor digitale filters zoals LD, HD, banddoorlaat en bandsper

[coefficients, amplitude, frequency] = wfir(filter-type, filter-order, [fg1 fg2], windowtype, [par1 par2])

- Wfir heeft volgende parameters nodig:
  - ***filter-type*** : lp,hp,bp,sb (laagdoorlaat hoogdoorlaat, banddoorlaat, bandsper)
  - ***Filter-orde*** : minimum 2 (2^{de} orde)
  - ***[fg1 fg2]*** : vector met twee afsnijfrequenties in het gebied tussen 0 en 0,5. Bij hoog- en laagdoorlaatfilter wordt enkel de eerste waarde fg1 gebruikt.
  - ***window-type*** : re, tr, hm, kr, ch (rectangular, triangular, hamming, kaiser, chebyshev)
  - ***[par1 par2]*** : vector met parameter voor het window-type

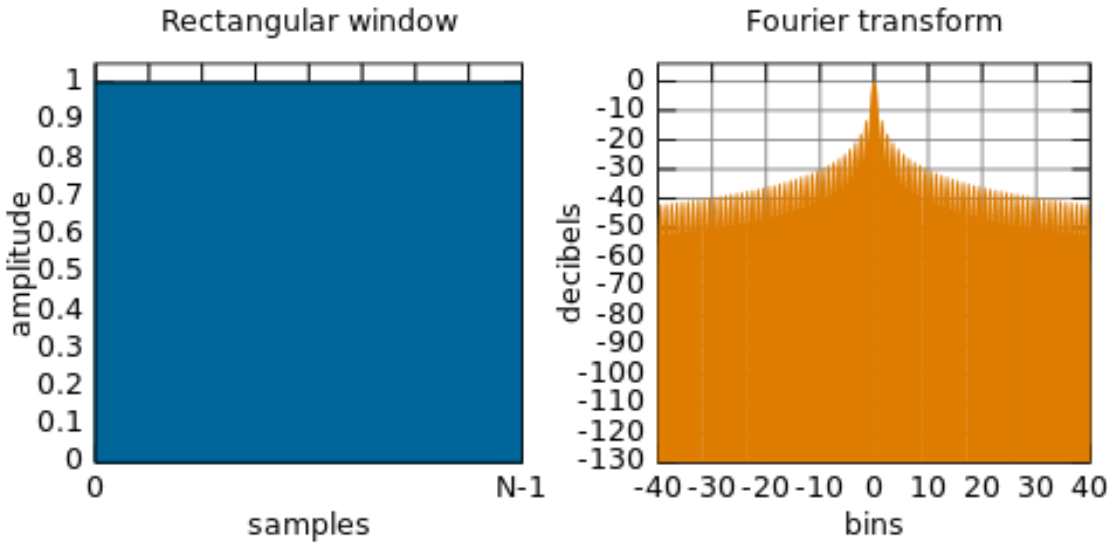
- De returnwaarden van de functie wfir() zijn:
  - ***coefficients :*** filtercoëfficiënten
  - ***amplitude*** : vector met lengte 256 met de amplitudewaarden
  - ***frequency*** : vector met lengte 256 met de frequenties in het gebied tussen 0 tot 0,5

Filterontwerp : wfir()

Typen Windows

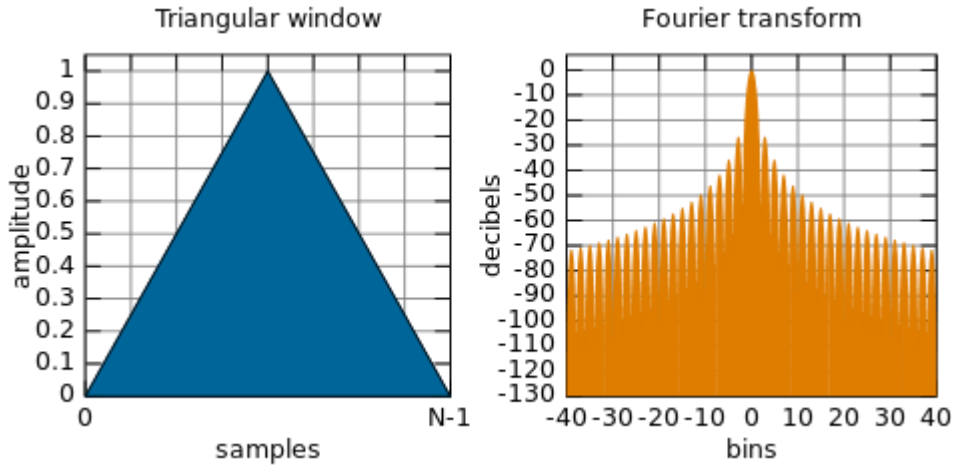
- Een window wordt gebruikt om in het ontwerp van een digitale filter, met een 'ideale' impulseresponse die oneindig lang is (bv. een sincfunctie), om te vormen naar een "finitive impulse response" (FIR) ontwerp. Dit wordt de window methode genoemd.

- ***Rectangular window*** (ook boxcar of Dirichiet genoemd)
  - Eenvoudigste window-equivalent om alle waarden op nul te plaatsen behalve de waarden n die doorgelaten worden.
  - $w(n) = 1$ met w(n) de windowfunctie
  - Nadeel van deze filter zijn de plotselinge veranderingen tussen niet doorgelaten en wel doorgelaten waarden waardoor er ongewenste effecten in de discrete time Fourier transformatie (DTFT) ontstaan

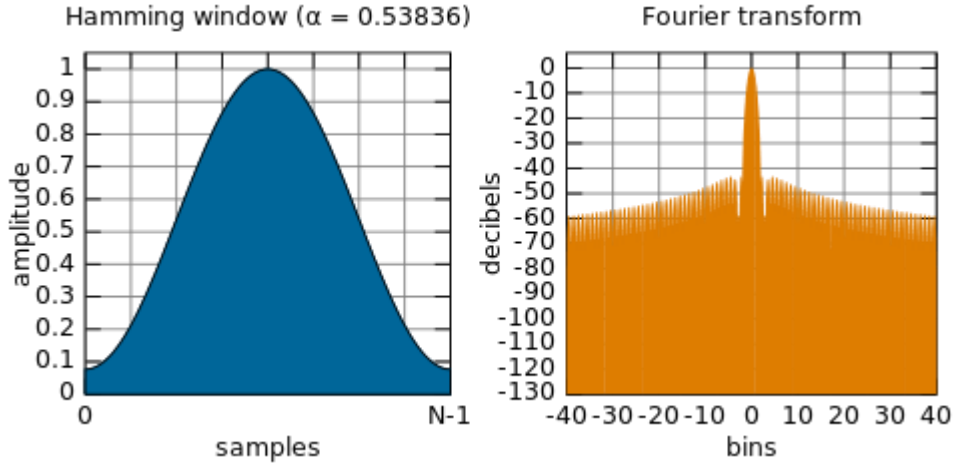


Filterontwerp : wfir()

- ***Triangular Window***
  - Windowfunctie : $w(n) = 1 - \frac{n - \frac{N-1}{2}}{\frac{L}{2}}$
    - Hierin kan L gelijk zijn aan N, N+1 of N-1
  - Kan gezien worden als de convolutie van twee N/2 brede rectangular windows.



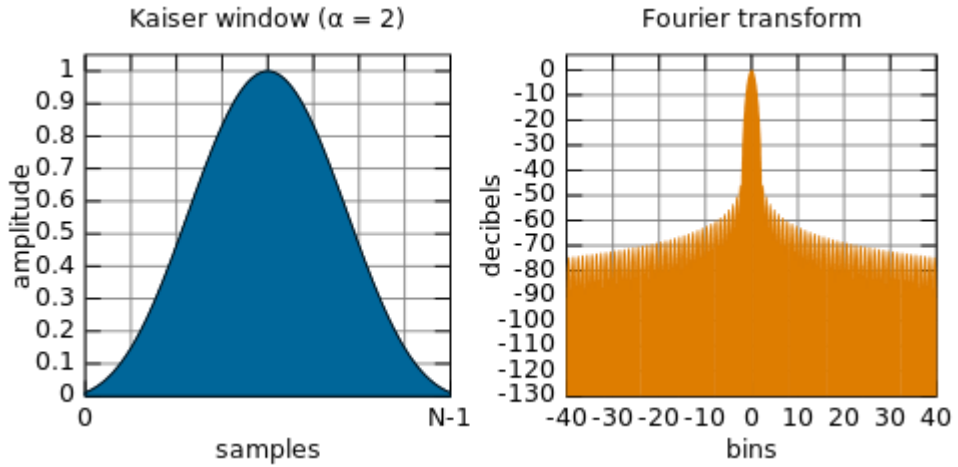
- ***Hamming window***
  - Windowfunctie : $w(n) = \alpha - \beta cos\frac{2\pi n}{N-1}$
  - Met α = 0,54 en β= α -1 = 0,46
  - Het window is zodanig geoptimaliseerd dat het maximum van de zijlob, die het dichtste bij het window ligt, een amplitude heeft van 1/5 van dat van het Hamming window

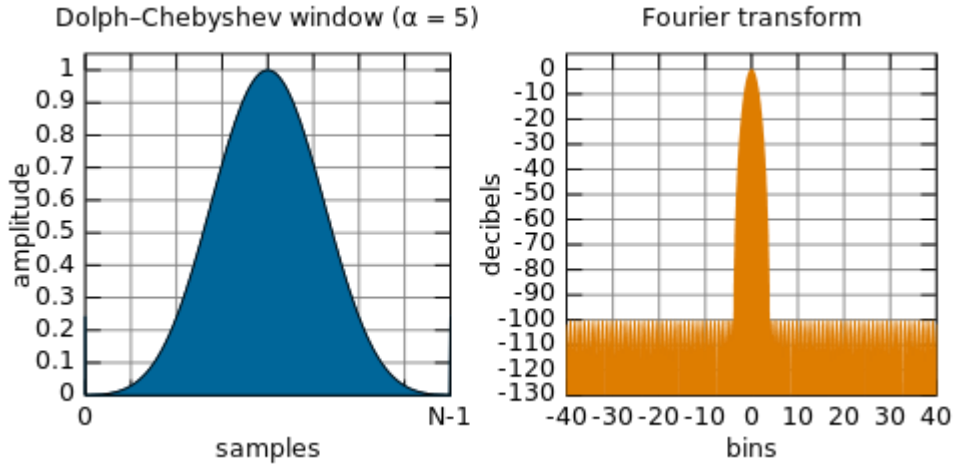


Hamming window, α = 0.53836 and β = 0.46164; B = 1.37. The original Hamming window would have α = 0.54 and β = 0.46; B = 1.3628.

Filterontwerp : wfir()

- ***Kaiser Window (kaiser-Bessel window)***
  - Eenvoudige benadering van de DPSS window die gebruik maakt van Besselfuncties. DPSS staat voor Discrete Prolate Spheroidal Sequences
  - Windowfunctie : $w(n) = \frac{I_0(\pi\sqrt{1-\left(\frac{2n}{N-1}-1\right)^2})}{I_0(\pi\alpha)}$
  - Met I_0 de 0de orde gewijzigde Besselfunctie en α een variabele parameter die de afweging bepaalt tussen de hoofdlob en zijlobniveaus van het spectrale leakage pattern. Typische waarde voor α is drie.
- ***Dolph-Chebyshev***
  - Minimaliseert de chebyshev norm van de zijlobes voor een gegeven hoofdlobebreedte.
  - De window-functie is meestal gedefinieerd in termen van real-valued discrete Fouriertransformatie $W_0(k)$.





Filterontwerp : wfir()

- Voorbeeld:
  - Ontwerp een laagdoorlaatfilter met orde 40 met afsnijfrequentie 40 Hz en window-type Hamming
  - Bepalen van filtercoëfficiënten, amplitude en frequentie:

[LD_coeff, amplitude, frequentie] = wfir ('lp', 40, [0.04 0], 'hm' [0 0]);

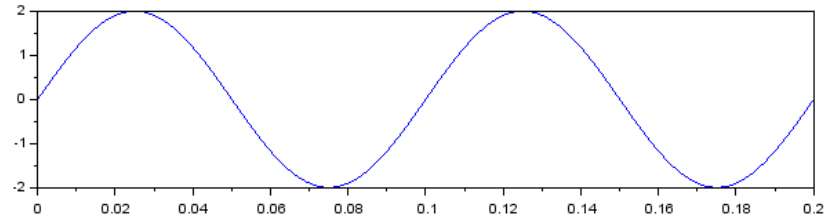
    - Parameters van wfir()
      - 'lp' : laagdoorlaatfilter
      - 40 : orde filter
      - [0,04 0] : bandbreedte (tussen 40 Hz en 0 Hz; normalisatie met fs = 1 kHz
      - 'hm' : hamming window
      - [0 0] parameters voor hamming window
  - De returnwaarden van de functie wfir() moeten vervolgens omgezet worden naar het scilab-formaat zodat de functie flts() gebruik kan maken van de filterparameters. De omvorming is in de voorbeeldcode verwerkt.

```
//filterontwerp LD-filter
//met orde 40 fc=40Hz en fs 1 kHz (BW = 40 Hz))
[LD_coeff, amplitude, frequentie] =...
   wfir ('lp', 40, [0.04 0], 'hm', [0 0]);
// bepalen van transferfunctie van de filter
// voor gebruik in scilab
// maken van polynoom
LD_polynoom = poly(LD_coeff, 'z', 'coeff')
//omvormen van z-coeff naar 1/z coeff
LD_functie = horner(LD_polynoom, (1/%z))
//aangeven werken met discrete waarden
LD_lineair_system = syslin('d', LD_functie)
```

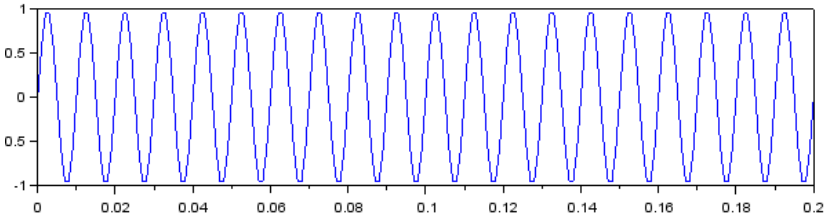
Filterontwerp : wfir()

- Maken van een testsignaal om de filter uit te testen.
- De filter heeft een fc = 40 Hz. Door een gecombineerd signaal te maken met signalen die bestaan uit een sinusgolf van 10 Hz, 100 Hz en 300 Hz bekomen we een testsignaal waarmee we de filter kunnen uittesten.

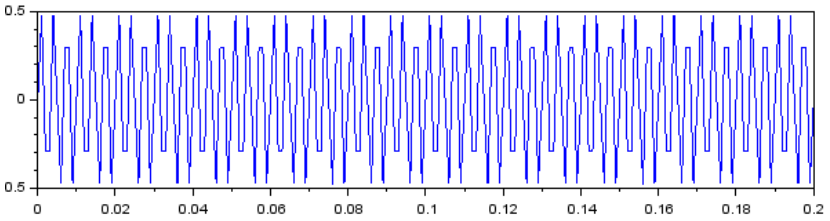
10 Hz



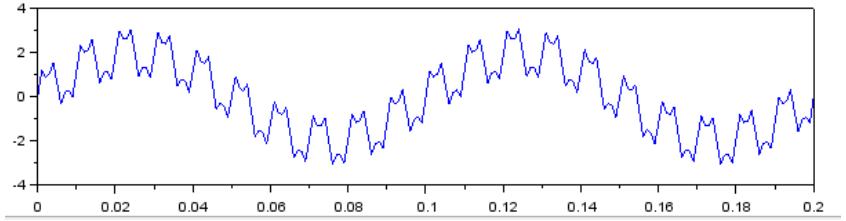
100 Hz



300 Hz



test

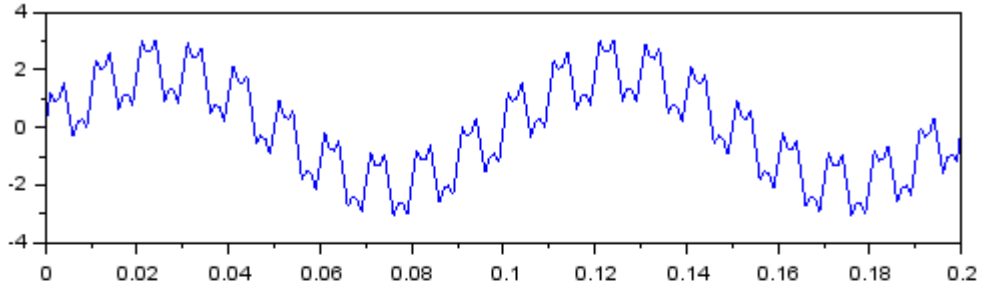


```
1  //genereren van testsignaal
2  // bestaande uit 10 Hz, 100 Hz en 300 Hz
3  // fs = 1 kHz
4  //tijdsduur van het testsignaal : 200 ms
5  //genereren tijdsvector (0-200ms) interval 0.001ms
6  t = [0:0.001:0.2];
7  // genereren sinus 10 Hz Vp = 2V
8  sin_10Hz = 2*sin(2*%pi*10*t);
9  // genereren sinus 100 Hz Vp = 1 V
10 sin_100Hz = 1*sin(2*%pi*100*t);
11 // genereren sinus 300 Hz Vp = 0,5 V
12 sin_300Hz = 0.5*sin(2*%pi*300*t);
13 //genereren testsignaal
14 testsign = sin_10Hz + sin_100Hz + sin_300Hz
15 //weergeven alle signalen
16 subplot(411)
17 plot(t,sin_10Hz);
18 subplot(412)
19 plot(t,sin_100Hz);
20 subplot(413)
21 plot(t,sin_300Hz);
22 subplot(414)
23 plot(t,testsign)
```

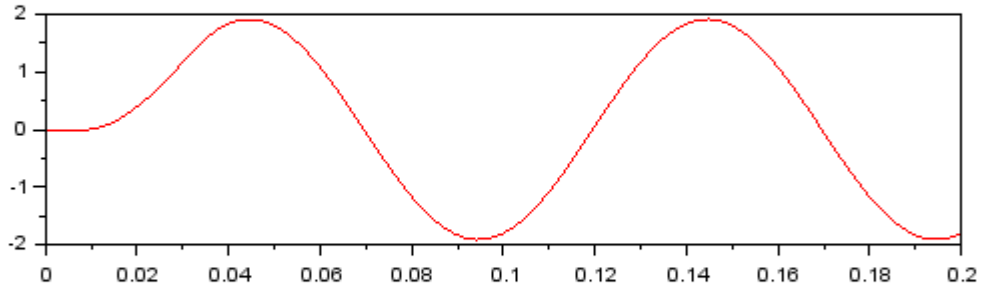
Filterontwerp : wfir()

- Uittesten van de filter orde 40, fc=40 Hz bij fs= 1 kHz.
- Via flts() kan de filter worden doorlopen met het testsignaal
  - LD_output = flts(testsign, LD_linear_system

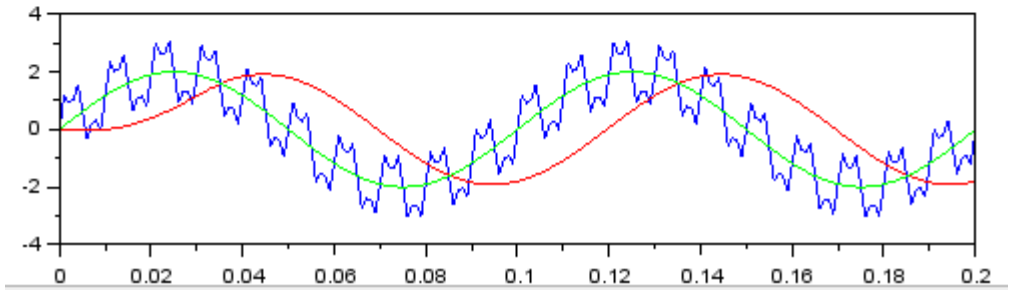
testsignal



Output filter (10 Hz)



Testsignal 10 Hz (groen)
Output filter (rood)

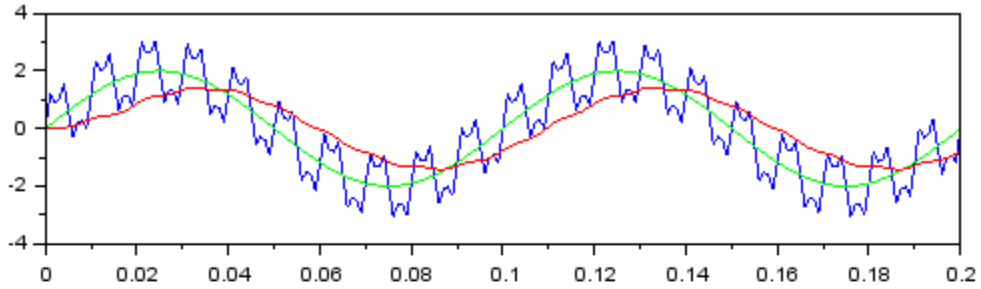


```
//filteren testsignaal met FIR
//fc filter is 40 Hz
//testsignaal
t = [0:0.001:0.2];
sin_10Hz = 2*sin(2*%pi*10*t);
sin_100Hz = 1*sin(2*%pi*100*t);
sin_300Hz = 0.5*sin(2*%pi*300*t);
testsign = sin_10Hz + sin_100Hz + sin_300Hz
//filterontwerp lp orde 40 fc 40 Hz, fs 1 kHz BW 40 Hz
[LD_coeff, amplitude, frequentie] = ...
   wfir('lp', 40, [0.04 0], 'hm', [0 0]);
LD_polynoom = poly(LD_coeff, 'z', 'coeff');
LD_functie = horner(LD_polynoom, 1/%z);
LD_lineair_system = syslin('d', LD_functie);
// testen werking van de filter
//filteren via scilab met functie flts()
LD_output = flts(testsign, LD_lineair_system)
// weergeven van testsignaal
subplot (311)
plot(t,testsign)
//weergeven filteroutput
subplot (312)
plot(t, LD_output, 'r');
subplot(313)
plot(t, testsign)
plot (t, LD_output, 'r')
plot (t, sin_10Hz, 'g')
```

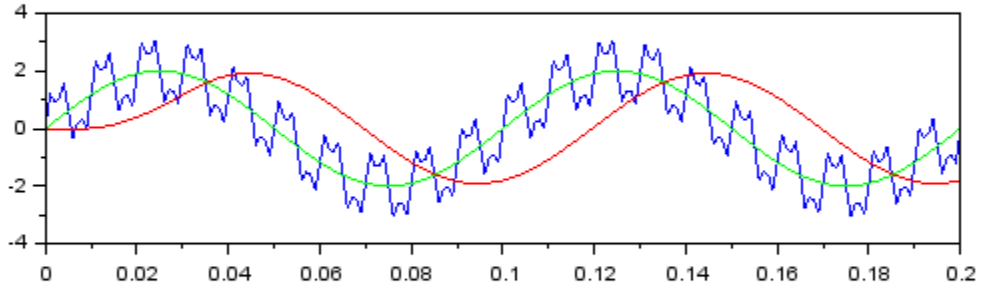
Filterontwerp : wfir()

- Opgave: pas de filter aan naar orde 20 en naar orde 80.
- Geef de output weer voor orde 20, 40 en 80 telkens onder elkaar. Per output geef je ook het testsignaal en de sinus van 10 Hz (onderdeel van het testsignaal) weer.

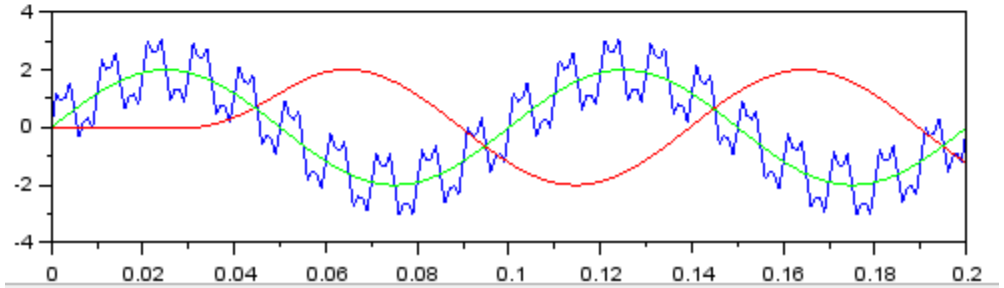
Orde 20



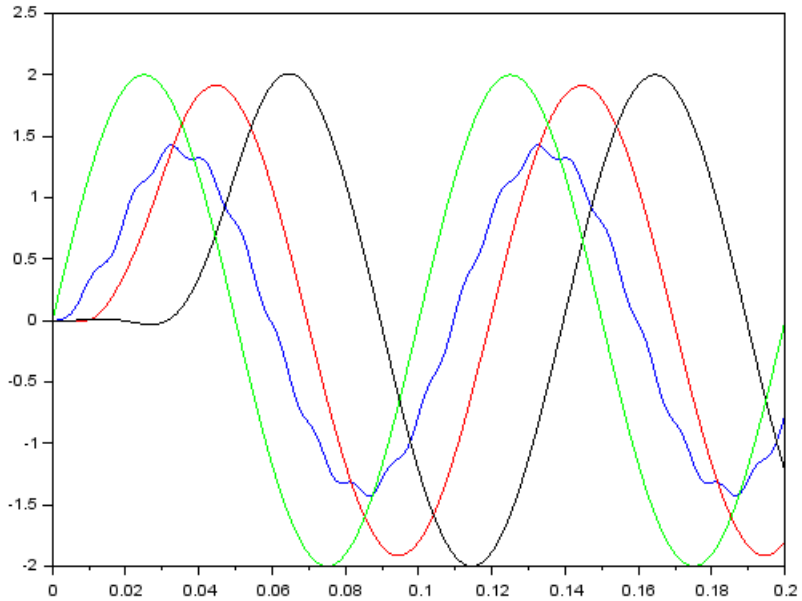
Orde 40



Orde 80



- Hoe lager de filterorde bij dezelfde afsnijfrequentie (40 Hz), hoe meer gedemd het signaal aan de uitgang verschijnt, en hoe meer zichtbaarder de hogere frequentiecomponenten.
- **FIR-filter heeft een relatief hoge filterorde nodig om tot goede resultaten te bekomen**
- **Hoe hoger de orde, hoe groter de faseverschuiving**



Groen: originele sinus 10 Hz
Blauw : filteroutput 20ste orde filter
Rood : filteroutput 40ste orde filter
Zwart : filteroutput 80ste orde filter

Filterontwerp : wfir()

- Opgave: pas de filter aan naar orde 20 en naar orde 80.
- Geef de output weer voor orde 20, 40 en 80 telkens onder elkaar. Per output geef je ook het testsignaal en de sinus van 10 Hz (onderdeel van het testsignaal) weer.

Orde 20

Orde 40

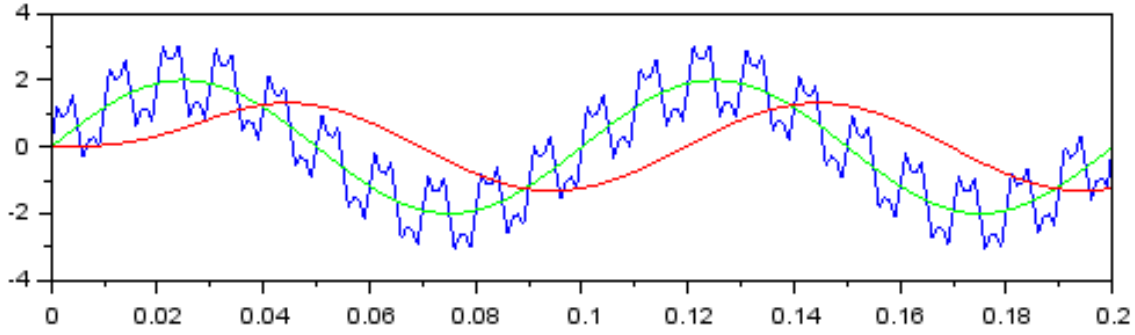
Orde 80

```
//filteren testsignaal met FIR
//fc filter is 40 Hz
//testsignaal
t = [0:0.001:0.2];
sin_10Hz = 2*sin(2*%pi*10*t);
sin_100Hz = 1*sin(2*%pi*100*t);
sin_300Hz = 0.5*sin(2*%pi*300*t);
testsign = sin_10Hz + sin_100Hz + sin_300Hz
//filterontwerp lp orde 40 fc 40 Hz, fs 1 kHz BW 40 Hz
[LD_coeff, amplitude, frequentie] = ...
   wfir('lp', 40, [0.04 0], 'hm', [0 0]);
LD_polynoom = poly(LD_coeff, 'z', 'coeff');
LD_functie = horner(LD_polynoom, 1/%z);
LD_lineair_system = syslin('d', LD_functie);
// testen werking van de filter
//filteren via scilab met functie flts()
LD_output = flts(testsign, LD_lineair_system)
// weergeven van testsignaal
subplot (311)
plot(t,testsign)
//weergeven filteroutput
subplot (312)
plot(t, LD_output, 'r');
subplot(313)
plot(t, testsign)
plot (t, LD_output, 'r')
plot (t, sin_10Hz, 'g')
```

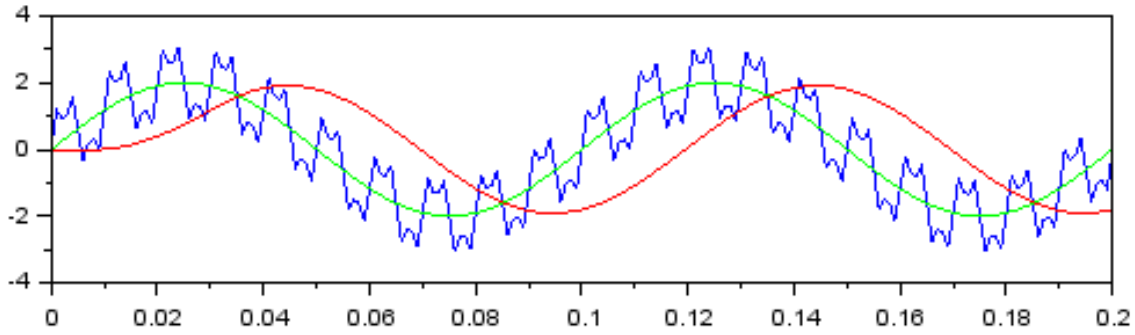
Filterontwerp : wfir()

- Opgave: pas de filter aan naar fc = 20 Hz en fc = 80 Hz.
- Geef de output weer voor fc = 20, 40 en 80 telkens onder elkaar. Per output geef je ook het testsignaal en de sinus van 10 Hz (onderdeel van het testsignaal) weer.

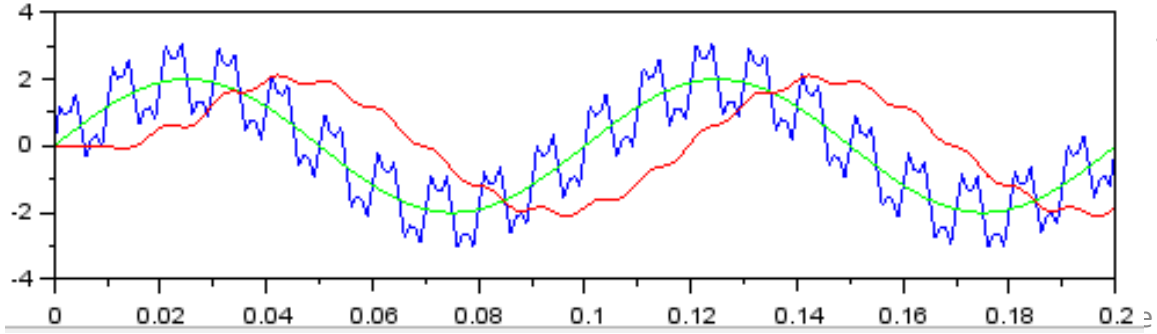
fc = 20



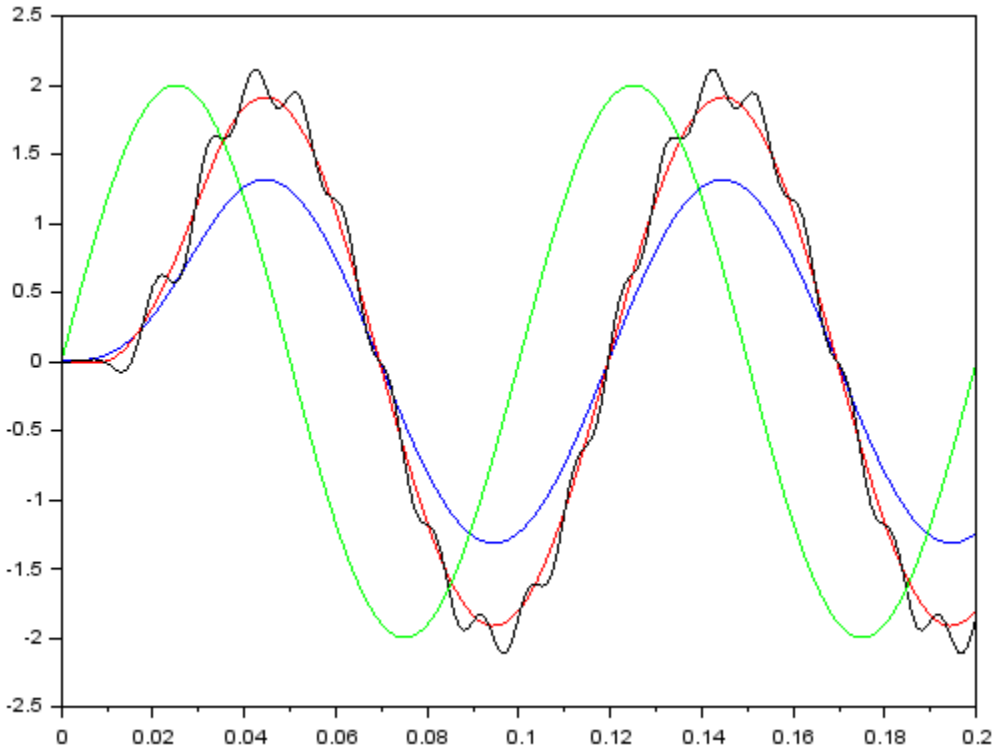
fc = 40



fc = 80



- Hoe lager de afsnijfrequentie bij dezelfde filterorde, hoe gedempter het uitgangssignaal wordt
- **Hoe hoger de afsnijfrequentie, hoe meer invloed van de hogere frequenties in het signaal aanwezig zijn**

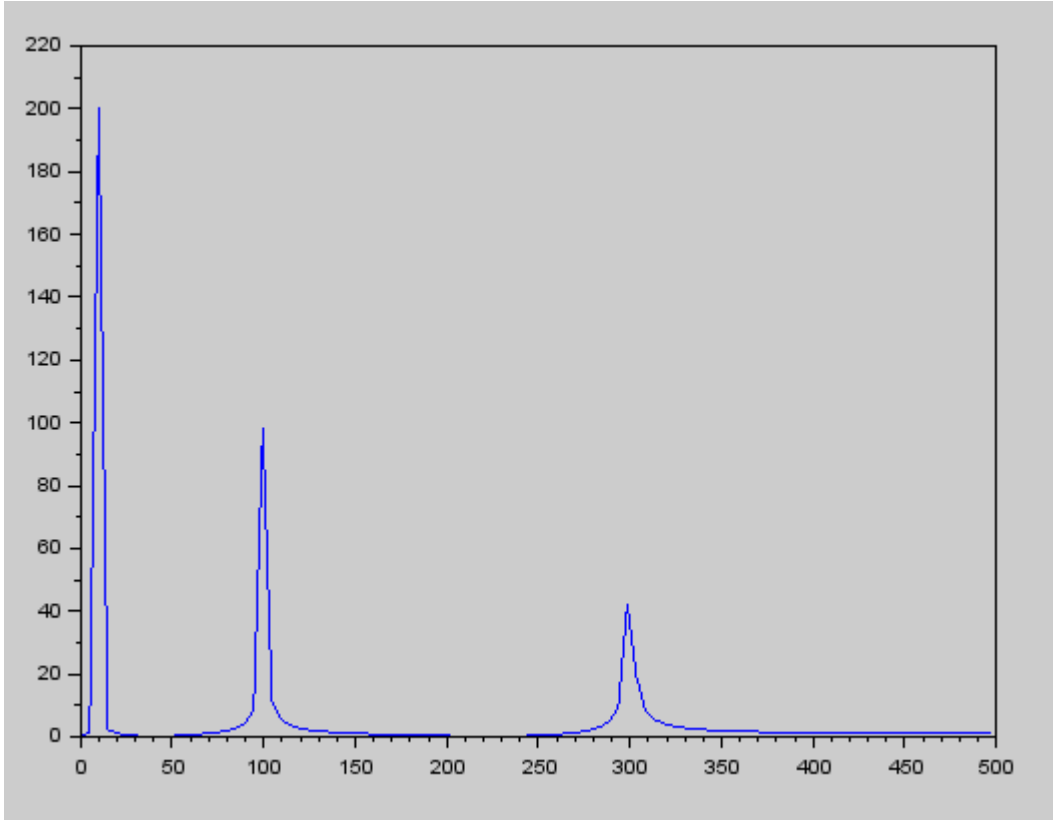


Groen: originele sinus 10 Hz
Blauw : fc = 20 Hz
Rood : fc = 40 Hz
Zwart: fc = 80 Hz

Filterontwerp : FFT-spectrum met FFT()

- FFT-spectrum methode wordt in een afzonderlijk hoofdstuk besproken. We halen het hier kort aan om het signaal te kunnen decomposeren.
- Met de functie FFT() kan je een signaal decomposeren in de sinuscompenten waarmee dit signaal is samengesteld.

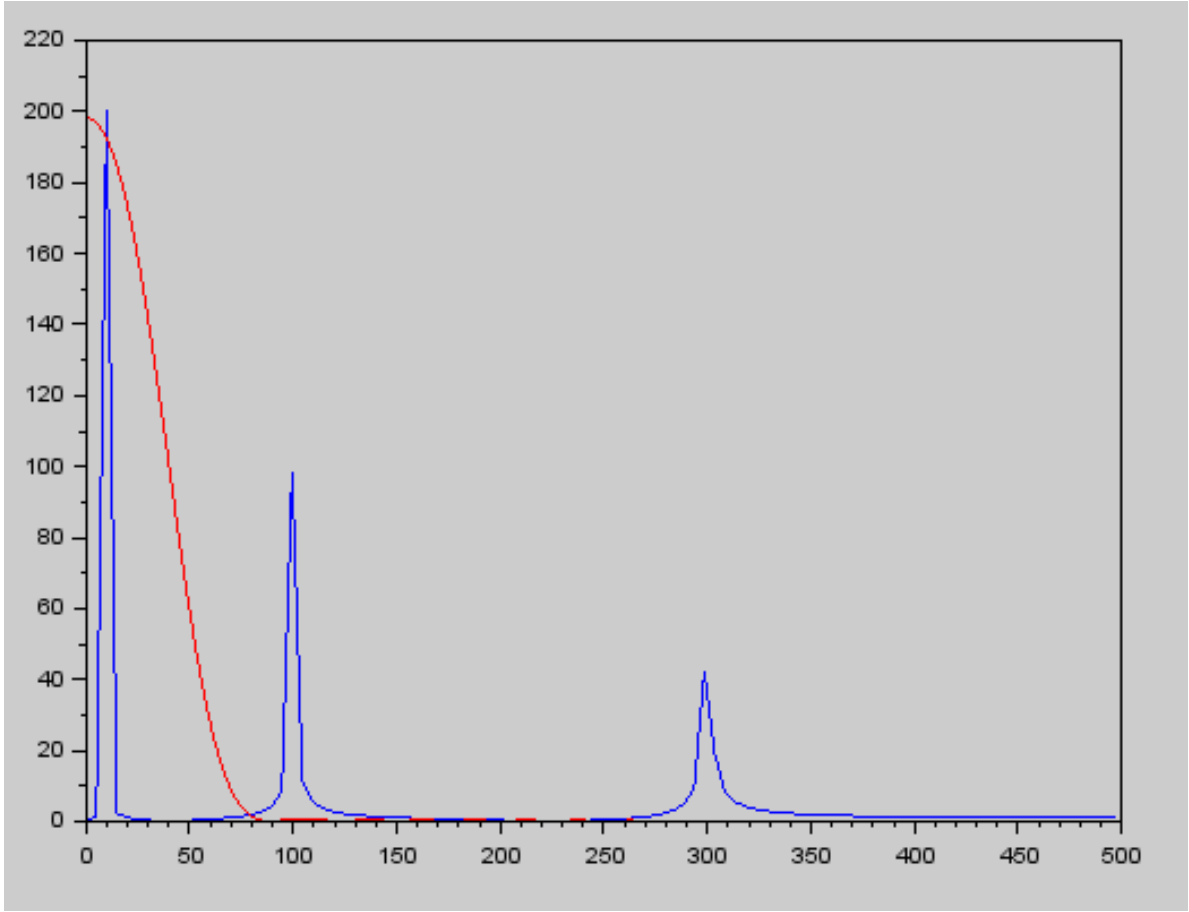
```
1  //testsignaal (10Hz, 100 Hz, 1000 Hz)
2  t = [0:0.001:0.2];
3  N=size(t,'*'); //aantal samples
4  sin_10Hz = 2*sin(2*%pi*10*t);
5  sin_100Hz = 1*sin(2*%pi*100*t);
6  sin_300Hz = 0.5*sin(2*%pi*300*t);
7  testsign = sin_10Hz + sin_100Hz + sin_300Hz
8
9  Frequentie_FFT = abs(fft(testsign));
10 //1000 = fs
11 f=1000*(0:(N/2))/N;//geassocieerde frequentievector
12 n=size(f,'*');
13 figure;
14 plot(f,Frequentie_FFT(1:n));
```



Filterontwerp : FFT-spectrum met FFT()

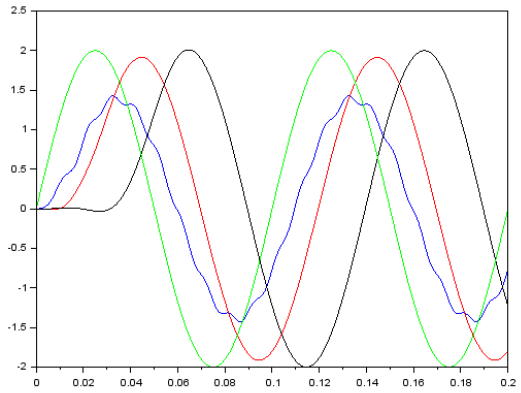
- De wfir() functie levert ook de magnitude response op en de overeenstemmende frequentievector van de filtertransferfunctie.

```
//testsignaal (10Hz, 100 Hz, 1000 Hz)
t = [0:0.001:0.2];
N=size(t,'*'); //aantal samples
sin_10Hz = 2*sin(2*%pi*10*t);
sin_100Hz = 1*sin(2*%pi*100*t);
sin_300Hz = 0.5*sin(2*%pi*300*t);
testsign = sin_10Hz + sin_100Hz + sin_300Hz

Frequentie_FFT = abs(fft(testsign));
//1000 = fs
f=1000*(0:(N/2))/N;//geassocieerde frequentievector
n=size(f,'*');
figure;
plot(f,Frequentie_FFT(1:n));

[LD_coeff, amplitude, frequentie] =...
   wfir ('lp', 40, [0.04 0], 'hm', [0 0]);

plot(frequentie*1000, amplitude*N, 'r')
```

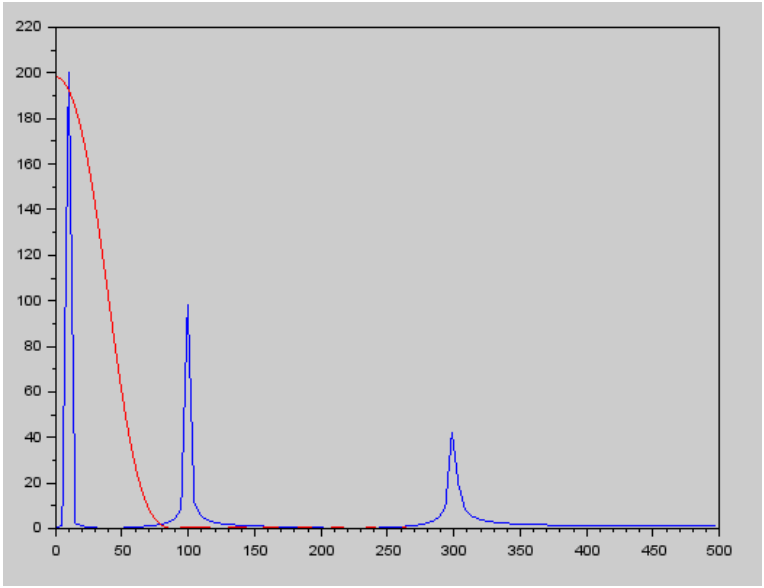


Filterontwerp : FFT-spectrum met FFT()

- De wfir() functie levert ook de magnitude response op en de overeenstemmende frequentievector van de filtertransferfunctie.



```
15 [LD_coeff, amplitude, frequentie] =...
16    wfir ('lp', 80, [0.04 0], 'hm', [0 0]);
17
18 plot(frequentie*1000, amplitude*N, 'r')
```



Geef de transfertfunctie weer voor een filter met orde 20 en 80

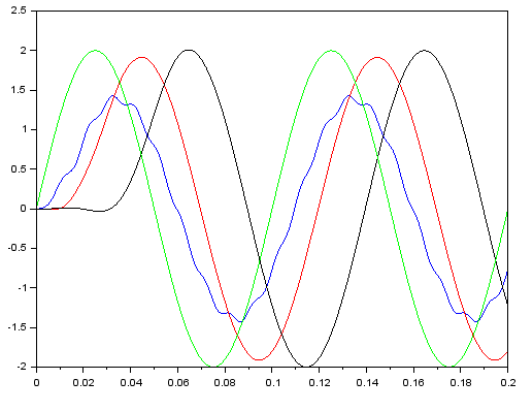
Orde = 40

Orde = 20

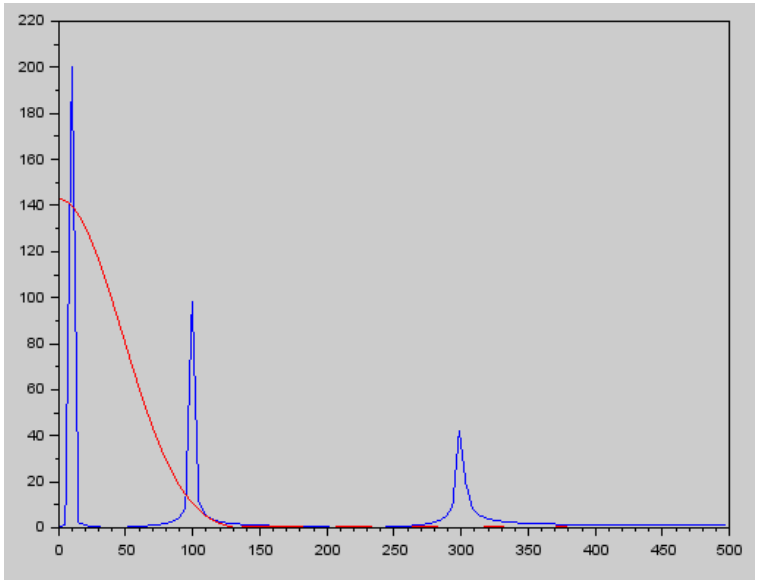
orde = 80

Filterontwerp : FFT-spectrum met FFT()

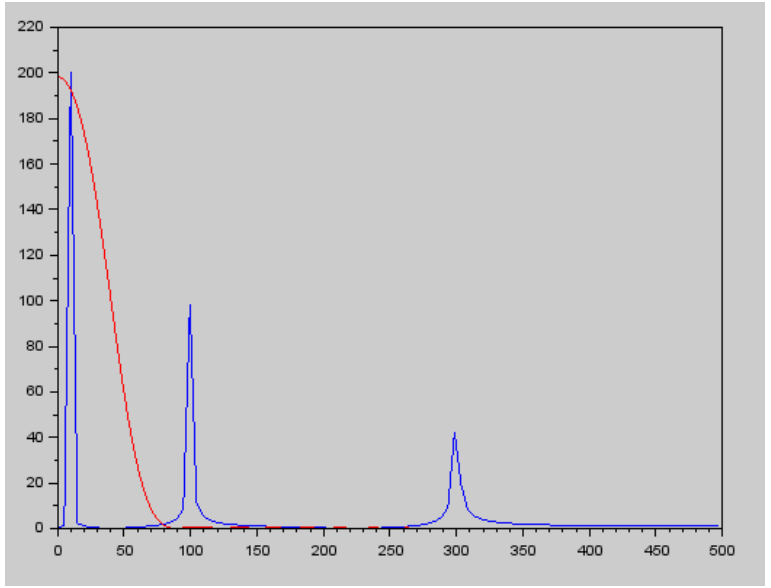
- De wfir() functie levert ook de magnitude response op en de overeenstemmende frequentievector van de filtertransferfunctie.



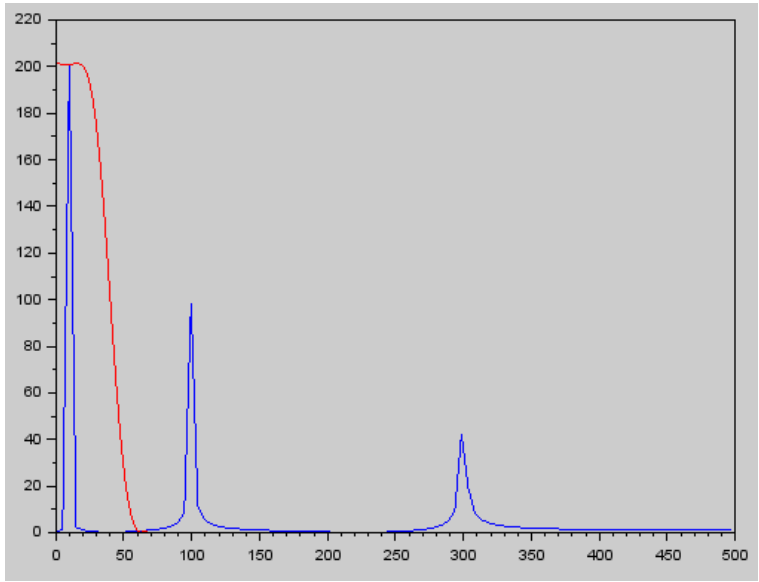
```
15 [LD_coeff, amplitude, frequentie] =...
16    wfir ('lp', 80, [0.04 0], 'hm', [0 0]);
17
18 plot(frequentie*1000, amplitude*N, 'r')
```



Orde = 20



Orde = 40

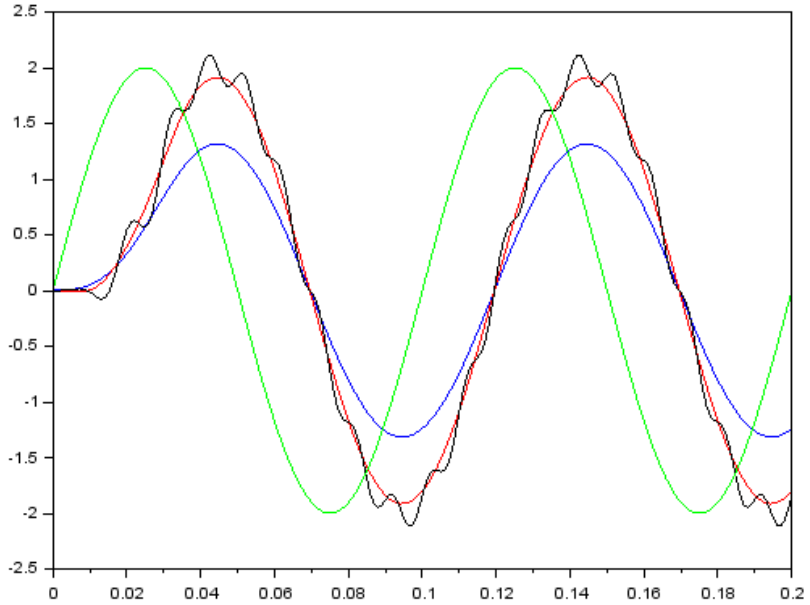


orde = 80

Filterontwerp : FFT-spectrum met FFT()

- De wfir() functie levert ook de magnitude response op en de overeenstemmende frequentievector van de filtertransferfunctie.

- Opgave: hou de orde op 40 en pas de afsnijfrequentie aan naar 20 Hz en 80 Hz
- Teken de transfertfunctie in het frequentiespectrum



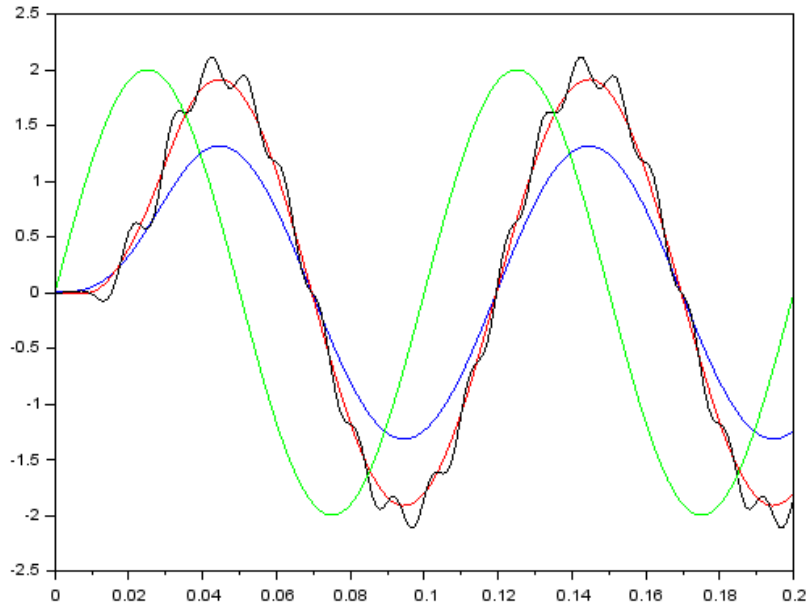
fc = 20 Hz

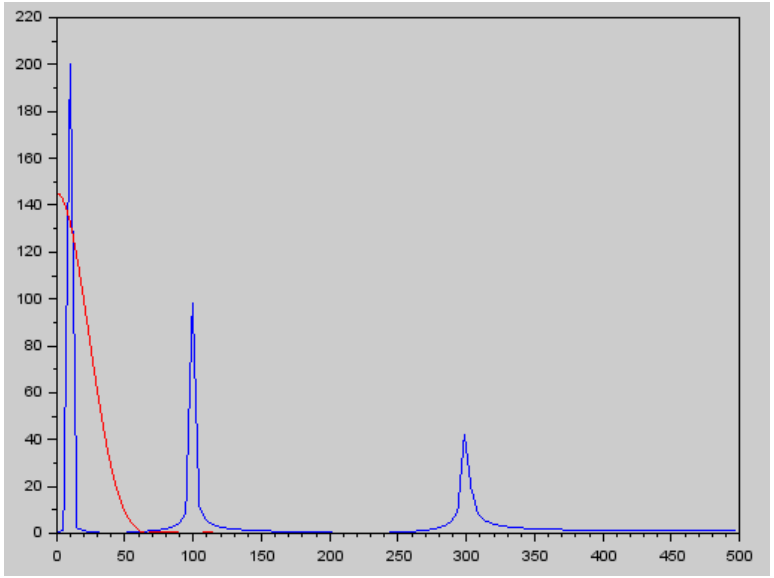
fc = 40 Hz

fc = 80Hz

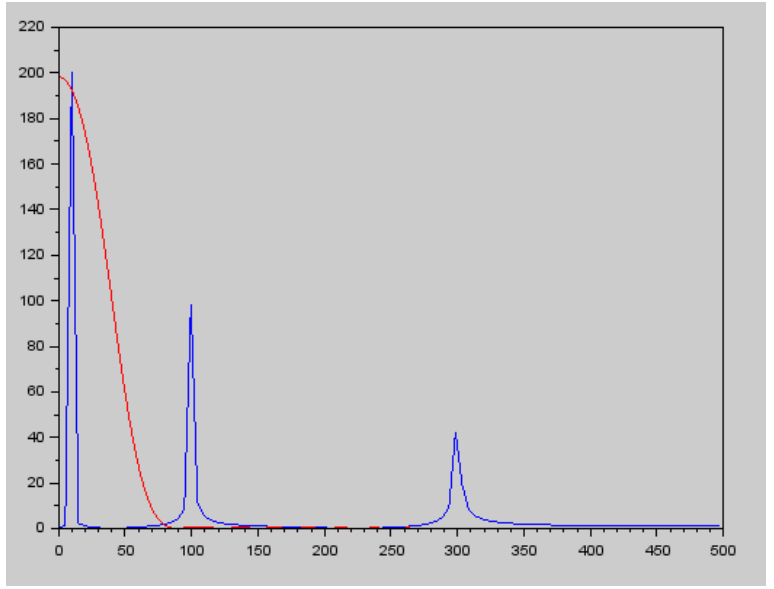
Filterontwerp : FFT-spectrum met FFT()

- De wfir() functie levert ook de magnitude response op en de overeenstemmende frequentievector van de filtertransferfunctie.

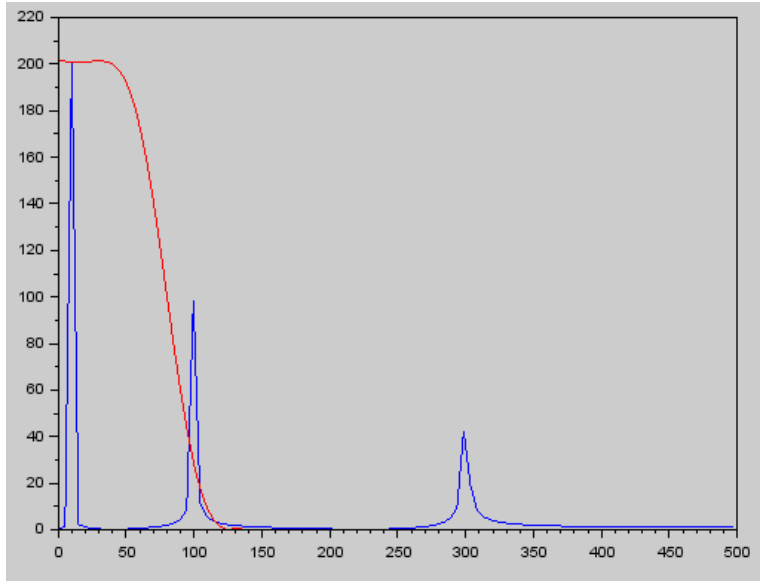




fc = 20 Hz



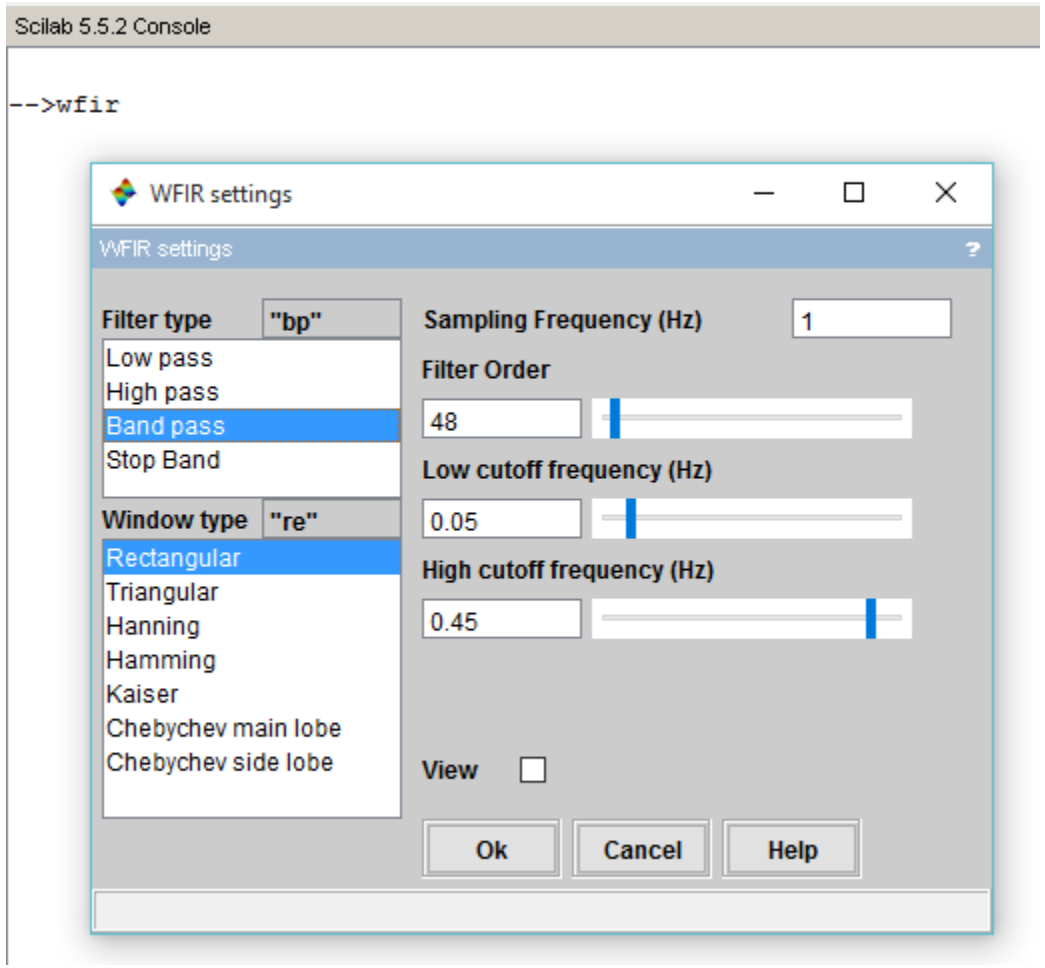
fc = 40 Hz



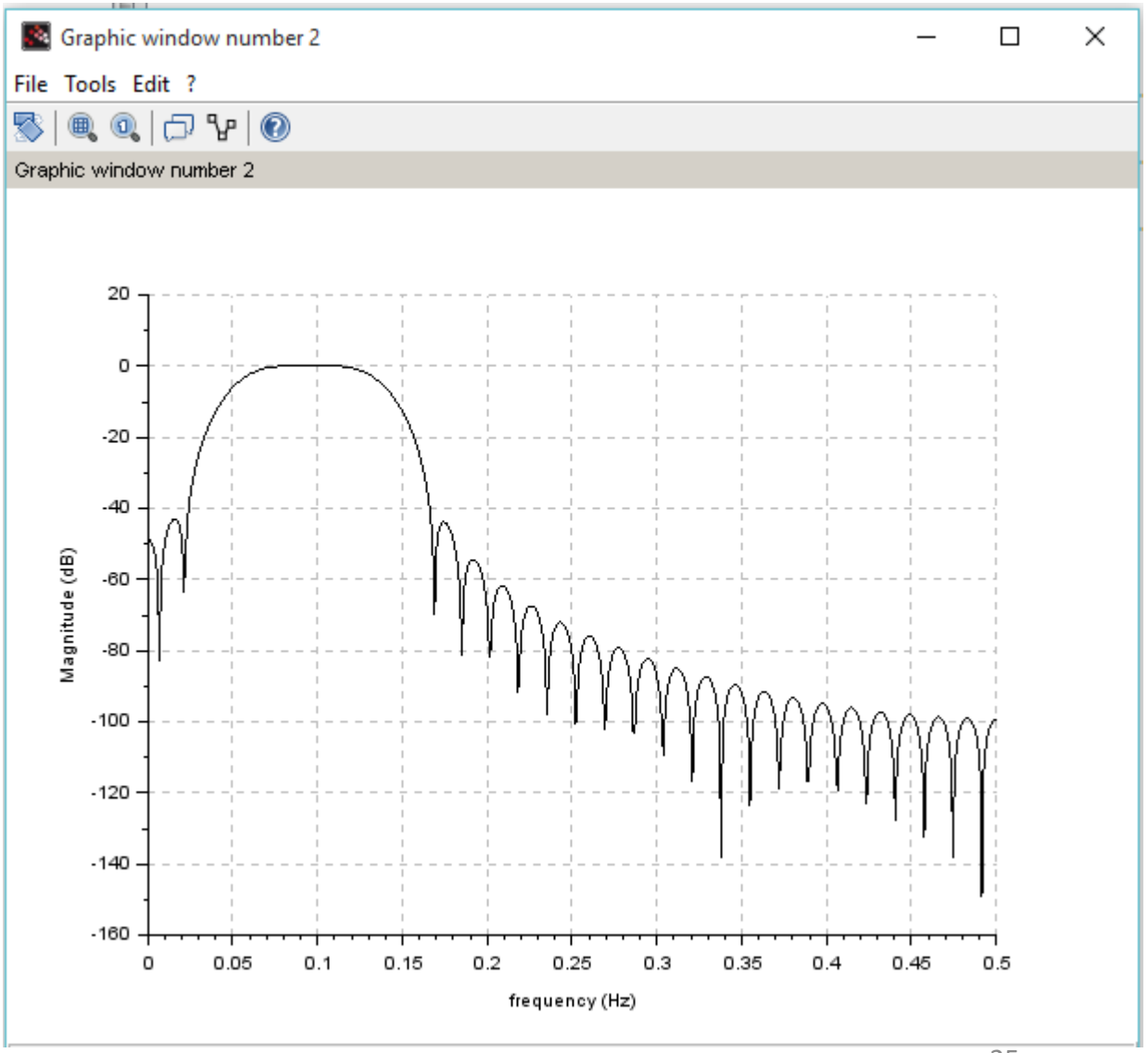
fc = 80Hz

Filterontwerp : snel en eenvoudig een filter ontwerpen via wfir in console

- Open console in scilab en typ het commano ***wfir***

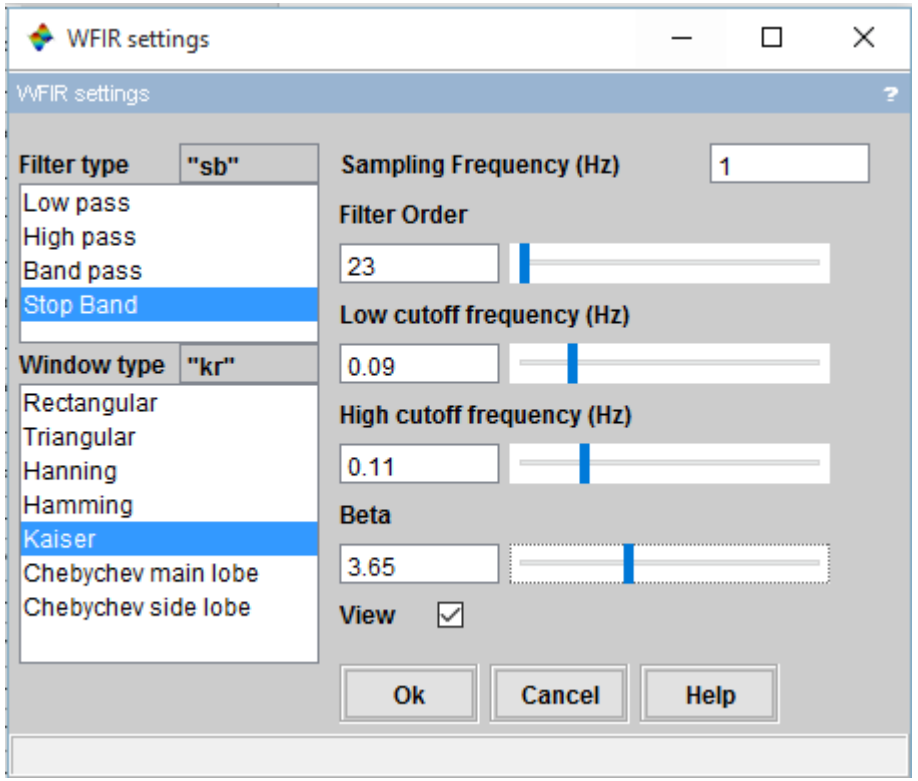


- Eens de settings gemaakt, klik in het venster WIFIR settings op View

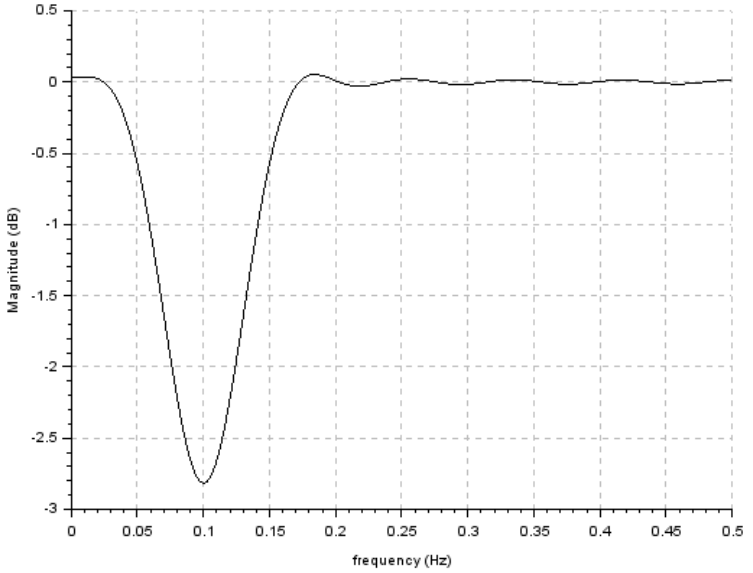


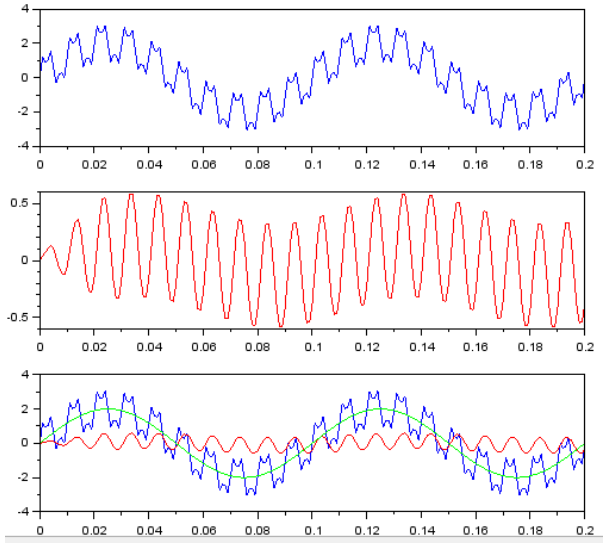
Filterontwerp : snel en eenvoudig een FIR-filter ontwerpen via wfir in console

- Eens filter in orde is, klik je op OK en in de console staan de coëfficiënten voor het FIR-filter




```
[SB_coeff, amplitude, frequentie] = ...
    wfir('sb', 23, [0.09 0.11], 'kr', [0 0]);
```





```
-->wfir
 ans  =


         column 1 to 7

 - 0.0035635  - 0.0074487  - 0.0089475  - 0.0046720    0.0060137    0.0193226    0.0281637

         column  8 to 14

   0.0259435    0.0109315  - 0.0117089  - 0.0319275    0.96  - 0.0319275  - 0.0117089

         column 15 to 21

   0.0109315    0.0259435    0.0281637    0.0193226    0.0060137  - 0.0046720  - 0.0089475

         column 22 to 23

 - 0.0074487  - 0.0035635
-->
```

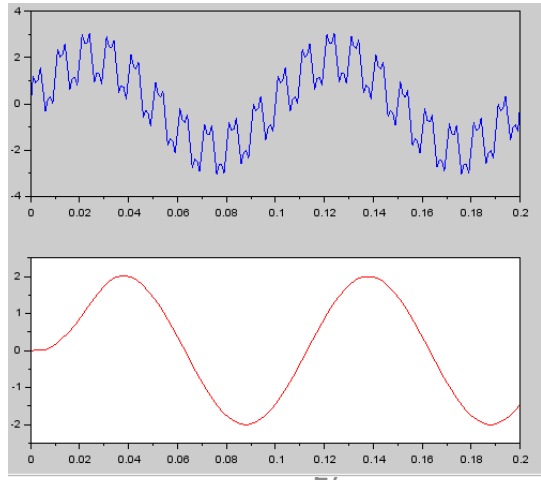
Filterontwerp : IIR-filter

- Via de functie iir() kan je een digitale IIR-filter ontwerpen in een analoog domein en daarna getransformeerd met de bilinear method in het Z-domein.
- ***Voorbeeld : IIR(3,'lp','butt'[0,04 0],[0 0])***
- Hierbij worden volgende argumenten gebruikt:
  - 3 : zet de maximale filterorde
  - "lp" : één van de 4 designmethoden (lp, hp, bp en sb)
  - 'butt' (Butterworth); andere filtertypen kunnen zijn:
    - 'butt" butterworth
    - 'cheb1' : Chebytchev
    - 'cheb2'
    - 'ellip' : Elliptic
  - [0,04 0] : afsnijfrequenties (tussen 0,0 en 0,5)
  - [0 0] : de parameters voor de rimpel in de betreffende designmethoden
- De IIR-functie returns een transferfunctie in het formaat Z^n.
- Voorbeeld : Z, Z^1, Z^2,Z^3

$$\frac{0.0015670 + 0.0047010z^{-2} + 0.0047010z^{-3} + 0.0015670z^{-3}}{-0.6041097 + 2.1152541z^{-2} - 2.4986083z^{-3} + z^{-3}}$$

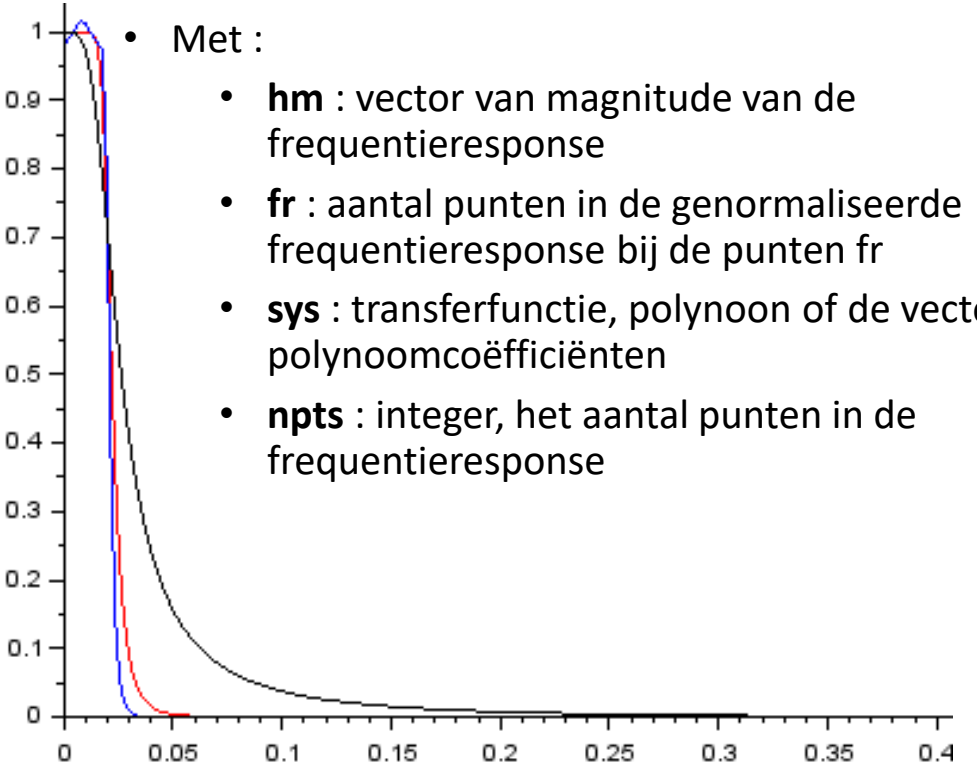
- Met de transferfunctie in het Z-domein en de functie flts() kan men een filtering uitvoeren.

```
//filteren testsignaal met IIR
//fc LD- filter is 40 Hz
//testsignaal
t = [0:0.001:0.2];
sin_10Hz = 2*sin(2*%pi*10*t);
sin_100Hz = 1*sin(2*%pi*100*t);
sin_300Hz = 0.5*sin(2*%pi*300*t);
testsignal = sin_10Hz + sin_100Hz + sin_300Hz
//filterontwerp lp orde 5 fc 40 Hz, fs 1 kHz BW 40 Hz
LD_IIR_filter = iir(5, 'lp', 'butt', [0.04 0], [0 0])
filter_output = flts(testsignal, LD_IIR_filter)
//afzonderlijk graphics window via figure
figure
    subplot(211)
    plot(t, testsignal)
    subplot(212)
    plot(t,filter_output,'r')
```



Filterontwerp : Filterorde van IIR-filter

- De filterorde n is de graad van teller en noemer van de polynoom in G(z). Hoe hoger de filterorde, hoe scherper de flanken van de filter, maar hoe langer de berekeningstijd.
- Hoe hoger de orde, hoe kleiner het gebied van de transit-area wordt tussen doorlaat en sper
- Functie frmag() berekent de magnitude van de frequentie van FIR en IIR-filters
- Voorbeeld: [hm, fr]=frmag(sys, npts)
  - Met :
    - **hm** : vector van magnitude van de frequentieresponse
    - **fr** : aantal punten in de genormaliseerde frequentieresponse bij de punten fr
    - **sys** : transferfunctie, polynoon of de vector met polynoomcoëfficiënten
    - **npts** : integer, het aantal punten in de frequentieresponse




```
1  //-LD_iir-transfer-orde-2-6-en--12
2
3  //testsignaal
4  t-=-[0:0.001:0.2];
5  sin_10Hz-=-2*sin(2*%pi*10*t);
6  sin_100Hz-=-1*sin(2*%pi*100*t);
7  sin_300Hz-=-0.5*sin(2*%pi*300*t);
8  testsignal-=-sin_10Hz-+-sin_100Hz-+-sin_300Hz
9  //-6de-orde-lijn-rood
10 LD_IIR_filter-=-iir-(6,-'lp',-'butt',-[0.02-0],-[0-0]);
11 [hm,fr]=frmag(LD_IIR_filter,256);
12 plot(fr,hm,'r')
13 //12de-orde-lijn-blauuw
14 LD_IIR_filter-=-iir-(12,-'lp',-'butt',-[0.02-0],-[0-0]);
15 [hm,fr]=frmag(LD_IIR_filter,256);
16 plot(fr,hm,'b')
17 //2de-orde-lijn-zwart
18 LD_IIR_filter-=-iir-(2,-'lp',-'butt',-[0.02-0],-[0-0]);
19 [hm,fr]=frmag(LD_IIR_filter,256);
20 plot(fr,hm,'k')
```

Filterontwerp : Opdracht

1, Ontwerp een IIR- en FIR-filter die de 7 harmonische filteren uit een blokgolf met amplitude 5 V en f=10 Hz. De samplingfrequentie is 2 kHz.

Maak de filterkarakteristiek zichtbaar, het ingangsignaal (2 perioden) en het uitgangssignaal.

2, Ontwerp een FIR-filter die de frequenties 300 – 3000 Hz doorlaat en de frequenties tussen 0 – 200 Hz met minstens 40 dB verzwakt en de frequenties 3300 – 20 000 Hz met -60 dB minimaal. De samplefrequentie is 8 kHz.

Voorbeeld : IIR(3,'lp','butt'[0,04 0],[0 0])

Hierbij worden volgende argumenten gebruikt:

- 3 : zet de maximale filterorde
- "lp" : één van de 4 designmethoden (lp, hp, bp en sb)
- 'butt' (Butterworth); andere filtertypen kunnen zijn:
  - 'butt" butterworth
  - 'cheb1' : Chebytchev
  - 'cheb2'
  - 'ellip' : Elliptic
- [0,04 0] : afsnijfrequenties (tussen 0,0 en 0,5)
- [0 0] : de parameters voor de rimpel in de betreffende designmethoden

```
//filterontwerp lp orde 5 fc 40 Hz, fs 1 kHz BW 40 Hz
LD_IIR_filter = iir(5, 'lp', 'butt', [0.04 0], [0 0])
filter_output = flts(testsignal, LD_IIR_filter)
```

```
LD_IIR_filter = iir (6, 'lp', 'butt', [0.02 0], [0 0]);
[hm,fr]=frmag(LD_IIR_filter,256);
```

Filterontwerp : Opdracht

1, Ontwerp een IIR- en FIR-filter die de 7 harmonische filteren uit een blokgolf met amplitude 5 V en f=10 Hz. De samplingfrequentie is 2 kHz.

Maak de filterkarakteristiek zichtbaar, het ingangsignaal (2 perioden) en het uitgangssignaal.

Voorbeeld : IIR(3,'lp','butt'[0,04 0],[0 0])

Hierbij worden volgende argumenten gebruikt:

- 3 : zet de maximale filterorde
- "lp" : één van de 4 designmethoden (lp, hp, bp en sb)
- 'butt' (Butterworth); andere filtertypen kunnen zijn:
  - 'butt" butterworth
  - 'cheb1' : Chebytchev
  - 'cheb2'
  - 'ellip' : Elliptic

```
1  // testsignaal 10 Hz; sampling 2 kHz; 2 perioden
2  t=(0:0.0005 : 0.2);
3  sin_10Hz = 2*sin(2*%pi*10*t)
4  //plot2d1('onn',t,[2*sin(t),1.5*squarewave(t),squarewave(t,10)])
5  blok_10Hz = 2*squarewave(sin_10Hz)
6  //plot (t, blok_10Hz)
7  BP_iir = iir(5,'bp', 'butt', [0.03 0.04],[0,0] )
8  filter_output = flts (blok_10Hz, BP_iir)
9  plot2d (t, blok_10Hz)
10 plot2d (t, filter_output)
```