Videosessie DSP

Frequentieselectieve filters

AP HOGESCHOOL ANTWERPEN

AP.BE

Introductie

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.

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Berekening waarde uitgang enkel afhankelijk van de huidig waarde en een aantal vorige waarden van het inputsignaal.

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Berekening waarde uitgang enkel afhankelijk van de huidig waarde en een aantal vorige waarden van het inputsignaal.

- Berekening waarde uitgang gebeurt met zowel een aantal waarden afkomstig van de ingang als van de uitgang (terugkoppeling van uitgang naar ingang)

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Er is geen terugkoppeling aanwezig van de uitgang naar de ingang. Hierdoor bestaat de gefilterde waarde van deze filter enkel met een eindig aantal waarden. De filter is hierdoor steeds stabiel.

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Er is geen terugkoppeling aanwezig van de uitgang naar de ingang.
- Hierdoor bestaat de gefilterde waarde van deze filter enkel met een eindig aantal waarden.
- De filter is hierdoor steeds stabiel.

- Hebben een terugkoppeling van de uitgang naar de ingang.
- Hierdoor bestaat de gefilterde waarde van deze filter theoretisch uit een oneindig aantal waarden
- Ten gevolge van deze terugkoppeling kunnen deze filters instabiel zijn.

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Het werkingsprincipe komt overeen met het bepalen van het gemiddelde in de huidige en vorige waarden die voorzien zijn van gewichten met een factor b_k.

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Het werkingsprincipe komt overeen met het bepalen van het gemiddelde in de huidige en vorige waarden die voorzien zijn van gewichten met een factor b_k.

- Het werkingsprincipe komt overeen met het samenvoegen van een vorm van het bepalen van gemiddelde met zowel de huidige als met vorige in- en uitgangangswaarden.
- Zowel de in- als uitgangswaarden die hierbij betrokken zijn, zijn voorzien van gewichten.

Filters met(IIR) en zonder tegenkoppeling (FIR)

- FIR heeft beduidend meer coëfficiënten nodig (hoger orde) dan de IIR−filter om dezelfde eigenschappen te bekomen

Filters met(IIR) en zonder tegenkoppeling (FIR)

- FIR heeft beduidend meer coëfficiënten nodig (hoger orde) dan de IIR−filter om dezelfde eigenschappen te bekomen

- 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)

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Lineaire faseresponse

- De groepsvertraging is niet constant
- Hebben een niet-lineaire faseresponse

Filters met(IIR) en zonder tegenkoppeling (FIR)

- Magnitude en fase kunnen bepaald worden onafhankelijk van elkaar.

- IIR-filters zijn gelijkwaardig als de analoge filter

Transfertfunctie

- In analoge systemen beschreven via differentiaalvergelijkingen
- In digitale systemen beschreven aan de hand van verschilvergelijkingen

- ***Wat is een verschilvergelijking?***

Transfertfunctie

- In analoge systemen beschreven via differentiaalvergelijkingen
- In digitale systemen beschreven aan de hand van verschilvergelijkingen

- ***Wat is een verschilvergelijking?***
  - ***Een verschilvergelijking is een rekenregel waarbij de huidige waarde van de output sequence y_n, en de huidige waarde van de input $x_n$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 $b_k$en a_k.

Verschilvergelijking

- ***FIR-verschilvergelijking***

  - ***Bevat enkel de ingangsignalen*** x_n , de uitgangssignalen y_n en de coëfficiënten b_k ***en*** a_k.

  - $y_n + a_1 y_{n-1} + a_2 y_{n-2} + \cdots + a_k y_{n-k} = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} + \ldots + b_k x_{n-k}$

Transfertvergelijking

Voorbeeld FIR-transfertvergelijking

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

De FIR-filter heeft geen noemer (geen terugkoppeling) in zijn transfertfunctie

Voorbeeld IIR-transfertvergelijking

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

Een IIR-filter heeft een gemeenschappelijke noemer in de transfertfunctie

Eigenschappen digitale filters

Afsnijfrequentie

- *Frequentie met nog de helft van het vermogen (-3 dB)*

Doorlaatband (pass band)

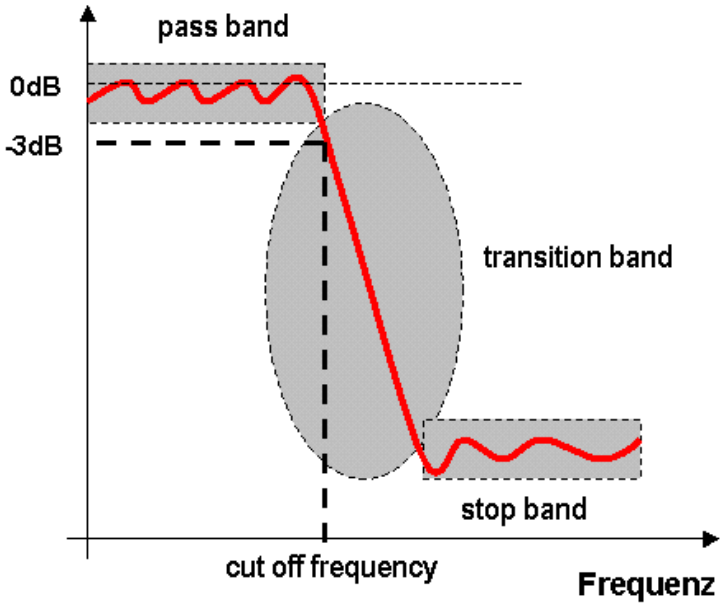
- *Zo vlak mogelijk om zo min mogelijk vervorming te bekomen*

Transistion band (overgangsband)

- *Moet zo klein mogelijk zijn*

Sperband

- *Zo hoog mogelijke dempingsverhouding (en eveneens vlak)*



Eigenschappen digitale filters

Filterorde

- *Hoe hoger de orde, hoe beter de filter*

Filtertype

- *Laagdoorlaat, hoogdoorlaat, banddoorlaat, bandsper*

Filter design procedure

- *Butterworth, Bessel, Chebycheff, …*

Sampling theorema en genormaliseerde frequenties

Gebruik van een bepaalde samplefrequentie f_s

- *kan frequenties evalueren tot aan* $\frac{f_s}{2}$ $(Nyquistfrequentie)$

In DSP f_s *dikwijls genormaliseerd naar frequentie* $1\ Hz$

- *Te evalueren frequenties tussen* $0\ Hz$ *en* $0,5\ Hz$

Sampling theorema en genormaliseerde frequenties

Voorbeeld:

- $f_s = 500\ Hz \Rightarrow fnyquist = 250\ Hz$
- $f_s = 10\ Hz \Rightarrow fnyquist = 5\ Hz$
- $f_s = 1\ Hz \Rightarrow fnyquist = 0,5\ Hz$

Sampling theorema en genormaliseerde frequenties

Voorbeeld:

- $f_s = 500\ Hz \Rightarrow fnyquist = 250\ Hz$
- $f_s = 10\ Hz \Rightarrow fnyquist = 5\ Hz$
- $f_s = 1\ Hz \Rightarrow fnyquist = 0,5\ Hz$

In digitale signaalprocessing wordt de samplefrequentie dikwijls genormaliseerd naar 1,0 Hz zodat de te samplen frequenties liggen tussen 0,0 fs en $0,5\ f_s$. (Dit betekent dat de afsnijfrequentie moet liggen tussen 0,0 fs en $0,5\ f_s$)

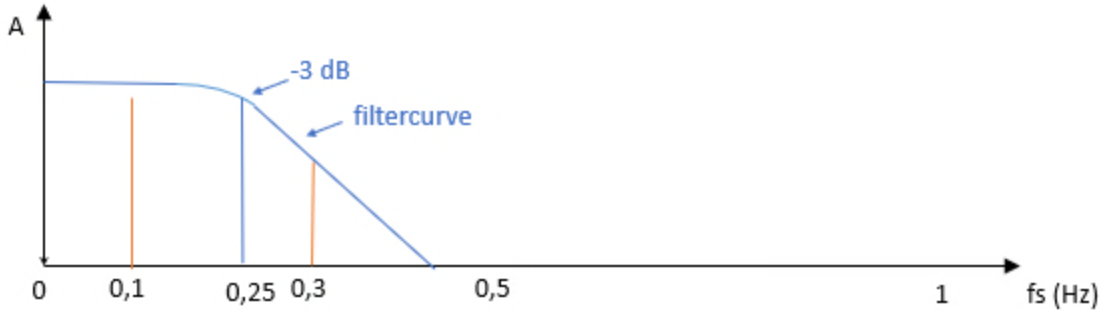
Sampling theorema en genormaliseerde frequenties

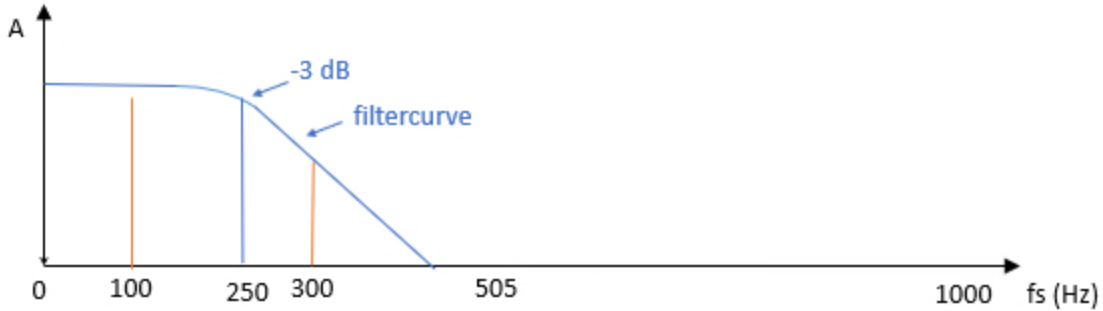
Voorbeeld:

Stel dat de kritische frequentie (afsnijfrequentie) ligt op 0,25 f_s (of 0,25 Hz)

- $f_s = 1\ kHz \Rightarrow f_c = 0,25 \times 1\ kHz = 250\ Hz$

- *Scilab-functies werken met genormaliseerde frequenties*





Eventjes overlopen

- Een ***frequentieselectieve filter*** wordt gebruikt om bepaalde delen van een frequentiespectrum door te laten ofwel te blokkeren.
  - Laagdoorlaat, hoogdoorlaat, banddoorlaat, bandsper

- Een ***digitale transfertfunctie*** wordt geschreven aan de hand van verschilvergelijkingen
  - Maakt gebruik van de huidige waarde van de output sequence y_n, en de huidige waarde van de input x_n en alle vorige waarden van de input- en output sequence

  - $\boldsymbol{voorbeeld\ FIR: G_z = b_0 + b_1 z^{-1} + \ ... + b_n z^{-n}}$

  - $\boldsymbol{voorbeeld\ IIR: G_z = \frac{b_0 + b_1 z^{-1} + ... + b_n z^{-n}}{1 + a_1 z^{-1} + ... + a_n z^{-n}}}$

Eventjes overlopen

- ***Eigenschappen van digitale filters***

  - Afsnijfrequentie (-3 dB)
  - Doorlaatband (zo vlak mogelijk)
  - Transitionband (zo klein mogelijk)
  - Sperband (zo veel mogelijk demping)
  - Filterorde (hoe hoger hoe beter)
  - Filtertype (LD, HD, BD, BS)
  - Filter design procedure (Butterworth, Bessel, Chebyscheff)

Eventjes overlopen

- ***Normalisatie***

  - ***In digitale signaalprocessing wordt de samplefrequentie dikwijls genormaliseerd naar 1,0 Hz zodat de te samplen frequenties liggen tussen 0,0 fs en*** $0,5\ f_s$.

  - ***Dit betekent dat de afsnijfrequentie moet liggen tussen 0,0 fs en*** $0,5\ f_s$

Voorbeeld : Toepassen van een filter op signalen

- Functie ***filter(num, dem, input)***
  - Hiermee kan je een digitale filter met een bepaalde transfertfunctie toevoegen
    - num : staat voor numerator (of teller van de transfertfunctie)
    - dem : staat voor denominator (of noemer van de transfertfunctie)
    - $input$: de te filteren samples

Voorbeeld : Toepassen van een filter op signalen

Stel een eenvoudig moving avarage FIR-filter waarbij de laatste twee ingangssignalen worden opgeteld en gedeeld door 2

$y_{[n]} = 0,5\ x[n] + 0,5\ x_{[n-1]}$ ***(hogere filterorden kunnen bekomen worden door meer coëfficiënten te gebruiken)***

- ***Coëfficiënten voor dit filter :***
  - ***Teller : 0,5 0,5***
  - ***Noemer : 1 (FIR heeft geen coëfficiënten in de noemer => enkel 1***

- ***Programmatie van de filter:***
  - ***Coëfficiënten ingeven als vectoren (array)***
  - ***Ingangssignaal ingeven als vector (array)***

Voorbeeld : Toepassen van een filter op signalen

Stel een eenvoudig moving avarage FIR-filter waarbij de laatste twee ingangssignalen worden opgeteld en gedeeld door 2

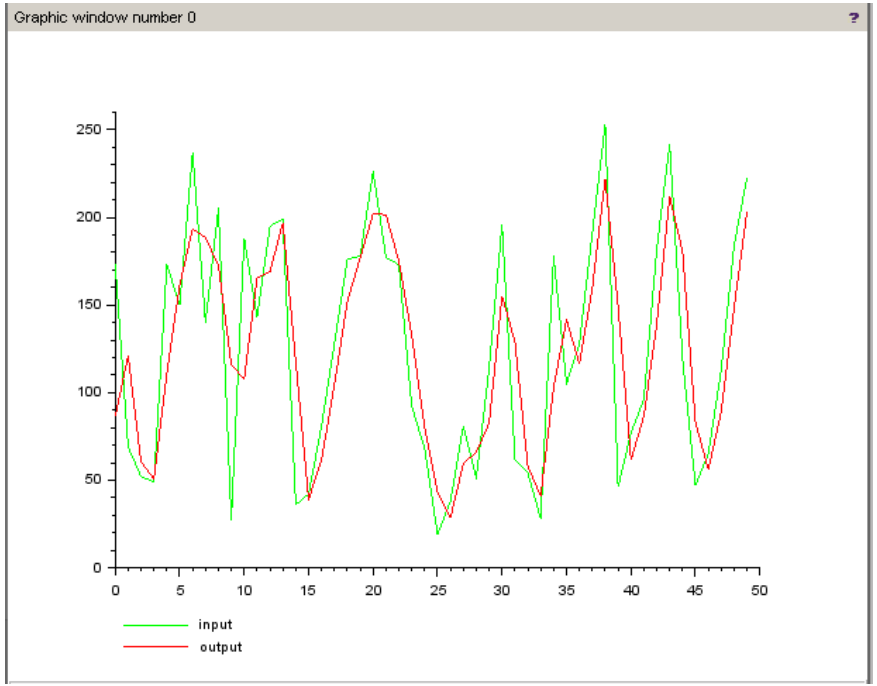
$y_{[n]} = 0,5\ x[n] + 0,5\ x_{[n-1]}$ ***(hogere filterorden kunnen bekomen worden door meer coëfficiënten te gebruiken)***

- ***Coëfficiënten voor dit filter :***
  - ***Teller : 0,5 0,5***
  - ***Noemer : 1 (FIR heeft geen coëfficiënten in de noemer => enkel 1***

- ***Programmatie van de filter:***
  - ***Coëfficiënten ingeven als vectoren (array)***
  - ***Ingangssignaal ingeven als vector (array)***

Voorbeeld : Toepassen van een filter op signalen

```
1  //·voorbeeld·van·FIR-filter:
2  //y(n))=0.5x(n)+0.5·x(n-1)
3  clc
4  clf
5
6  //·genereren·van·50·random·samples·(ADC-simulatie)
7  x_waarden·=·int(255*·rand(50,1))
8
9  //·definiëren·filter·met·twee·coëfficiënten·0.5·0.5
10 //·num·=·0.5·0.5;·dem·=·1·(FIR)·input·=·x_waarden
11
12 y_waarden·=·filter([0.5·0.5],[1],x_waarden)
13
14 //·plot·horizontaal·"n"·(aantal·waarden·-·samples)
15 n·=·[0·:·1·:·49]
16
17 //·vertikaal·amplitude·van·de·random·x_waarden
18 //·plot·met·legende·en·aanpassen·kleurstijl
19 plot2d·(n,[x_waarden,y_waarden],·leg·="input@output",...
20 ····style=[color("green"),·color("red")]);
21
```



Voorbeeld : Toepassen van een filter op signalen

```
// voorbeeld van FIR-filter:
//y(n))=0.5x(n)+0.5 x(n-1)
clc
clf

// genereren van 50 random samples (ADC-simulatie)
x_waarden = int(255* rand(50,1))

// definiëren filter met twee coëfficiënten 0.5 0.5
// num = 0.5 0.5; dem = 1 (FIR) input = x_waarden

y_waarden = filter([0.5 0.5],[1],x_waarden)

// plot horizontaal "n" (aantal waarden - samples)
n = [0 : 1 : 49]

// vertikaal amplitude van de random x_waarden
// plot met legende en aanpassen kleurstijl
plot2d (n,[x_waarden,y_waarden], leg ="input@output",...
        style=[color("green"), color("red")]);
```

Voorbeeld : Toepassen van een filter op signalen

```
1.  // voorbeeld van FIR-filter:
2.  //y(n))=0.5x(n)+0.5 x(n-1)
3.  clc
4.  clf

5.  // genereren van 50 random samples (ADC-simulatie)
6.  x_waarden = int(255* rand(50,1))

7.  // definiëren filter met twee coëfficiënten 0.5 0.5
8.  // num = 0.5 0.5; dem = 1 (FIR) input = x_waarden

9.  y1_waarden = int(filter([0.5 0.5],[1],x_waarden))

10. //definiëren filter met 4 coëfficiënten
11. Y2_waarden = int(filter([0.25 0.25 0.25 0.25 ],[1],x_waarden))
12. // plot horizontaal "n" (aantal waarden - samples)
13. n = [0 : 1 : 49]

14. // verticaal amplitude van de random x_waarden
15. // plot met legende en aanpassen kleurstijl
16. plot2d (n,[x_waarden,y_waarden], leg ="input@output",...
17. style=[color("green"), color("red")]);
```

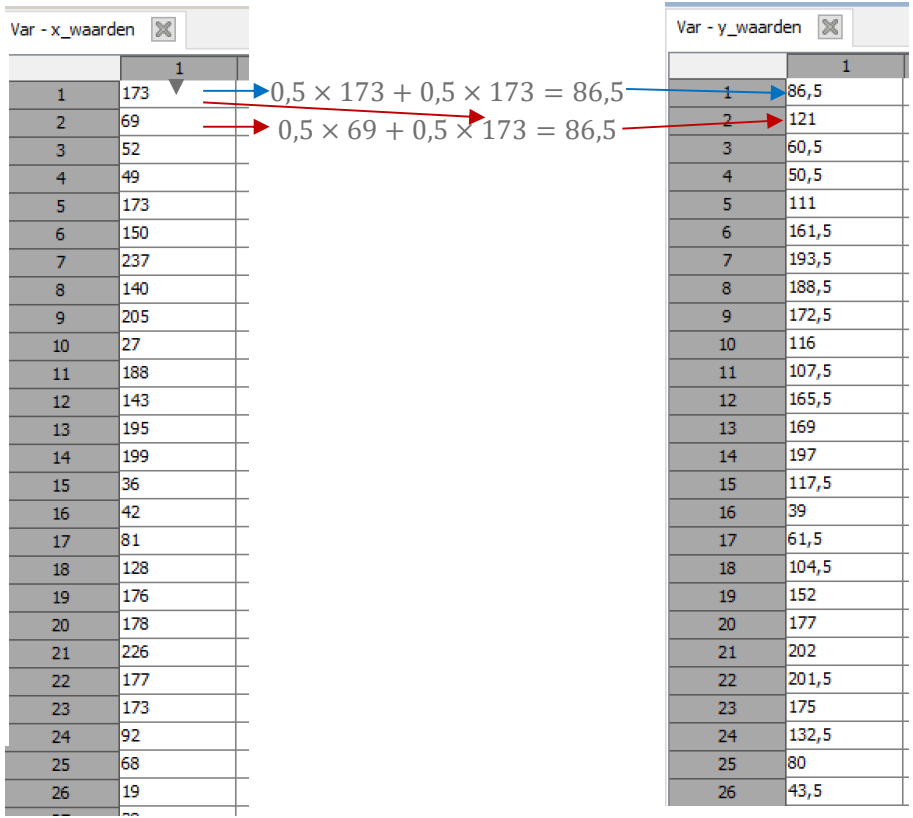
Var - x_waarden

	1
1	173
2	69
3	52
4	49
5	173
6	150
7	237
8	140
9	205
10	27
11	188
12	143
13	195
14	199
15	36
16	42
17	81
18	128
19	176
20	178
21	226
22	177
23	173
24	92
25	68
26	19

27	38
28	81
29	51
30	114
31	196
32	62
33	54
34	28
35	178
36	105
37	128
38	191
39	253
40	46
41	77
42	96
43	182
44	242
45	119
46	47
47	65
48	113
49	184
50	223

Voorbeeld : Toepassen van een filter op signalen

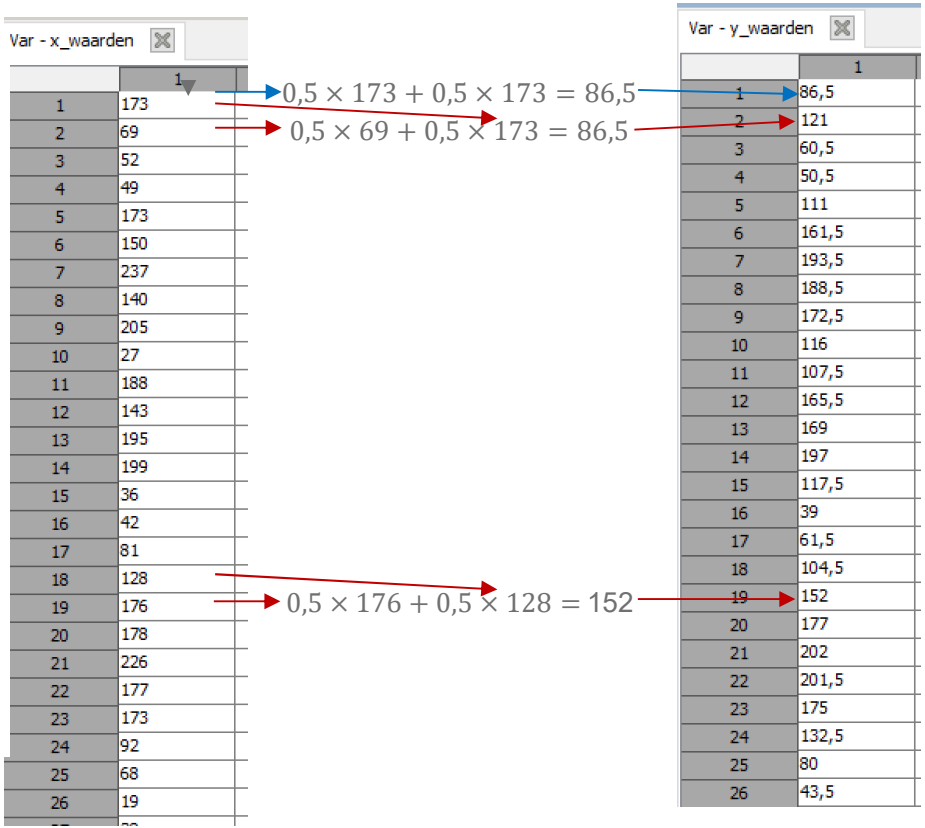
```
1.  // voorbeeld van FIR-filter:
2.  //y(n))=0.5x(n)+0.5 x(n-1)
3.  clc
4.  clf

5.  // genereren van 50 random samples (ADC-simulatie)
6.  x_waarden = int(255* rand(50,1))

7.  // definiëren filter met twee coëfficiënten 0.5 0.5
8.  // num = 0.5 0.5; dem = 1 (FIR) input = x_waarden

9.  y1_waarden = int(filter([0.5 0.5],[1],x_waarden))

10. //definiëren filter met 4 coëfficiënten
11. Y2_waarden = int(filter([0.25 0.25 0.25 0.25 ],[1],x_waarden))
12. // plot horizontaal "n" (aantal waarden - samples)
13. n = [0 : 1 : 49]

14. // verticaal amplitude van de random x_waarden
15. // plot met legende en aanpassen kleurstijl
16. plot2d (n,[x_waarden,y_waarden], leg ="input@output",...
17. style=[color("green"), color("red")]);
```



	x_waarden	y_waarden
1	173	86,5
2	69	121
3	52	60,5
4	49	50,5
5	173	111
6	150	161,5
7	237	193,5
8	140	188,5
9	205	172,5
10	27	116
11	188	107,5
12	143	165,5
13	195	169
14	199	197
15	36	117,5
16	42	39
17	81	61,5
18	128	104,5
19	176	152
20	178	177
21	226	202
22	177	201,5
23	173	175
24	92	132,5
25	68	80
26	19	43,5

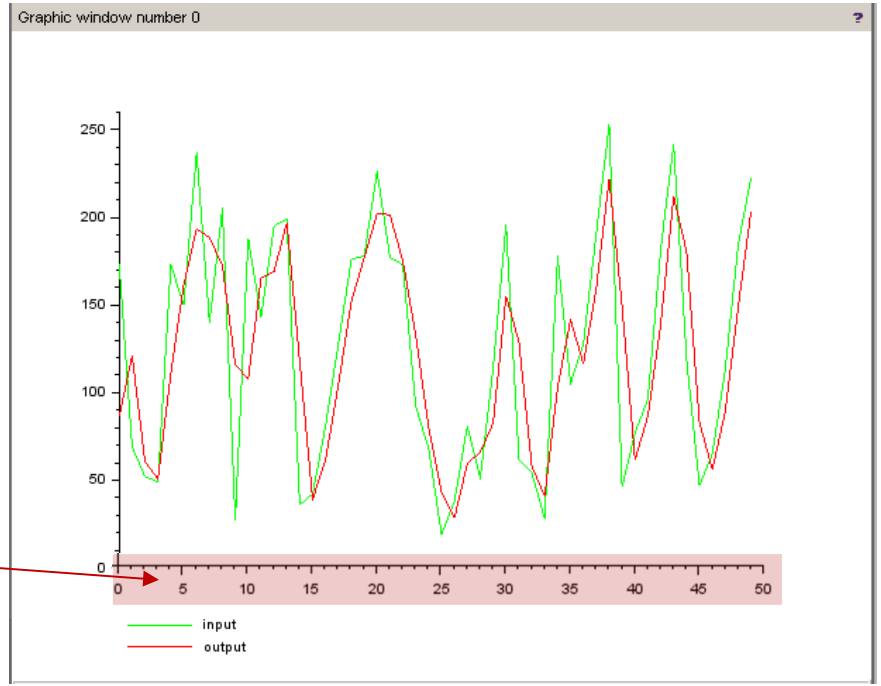
Voorbeeld : Toepassen van een filter op signalen

```
1.  // voorbeeld van FIR-filter:
2.  //y(n))=0.5x(n)+0.5 x(n-1)
3.  clc
4.  clf

5.  // genereren van 50 random samples (ADC-simulatie)
6.  x_waarden = int(255* rand(50,1))

7.  // definiëren filter met twee coëfficiënten 0.5 0.5
8.  // num = 0.5 0.5; dem = 1 (FIR) input = x_waarden

9.  y1_waarden = int(filter([0.5 0.5],[1],x_waarden))

10. //definiëren filter met 4 coëfficiënten
11. Y2_waarden = int(filter([0.25 0.25 0.25 0.25 ],[1],x_waarden))
12. // plot horizontaal "n" (aantal waarden - samples)
13. n = [0 : 1 : 49]

14. // verticaal amplitude van de random x_waarden
15. // plot met legende en aanpassen kleurstijl
16. plot2d (n,[x_waarden,y_waarden], leg ="input@output",...
17. style=[color("green"), color("red")]);
```



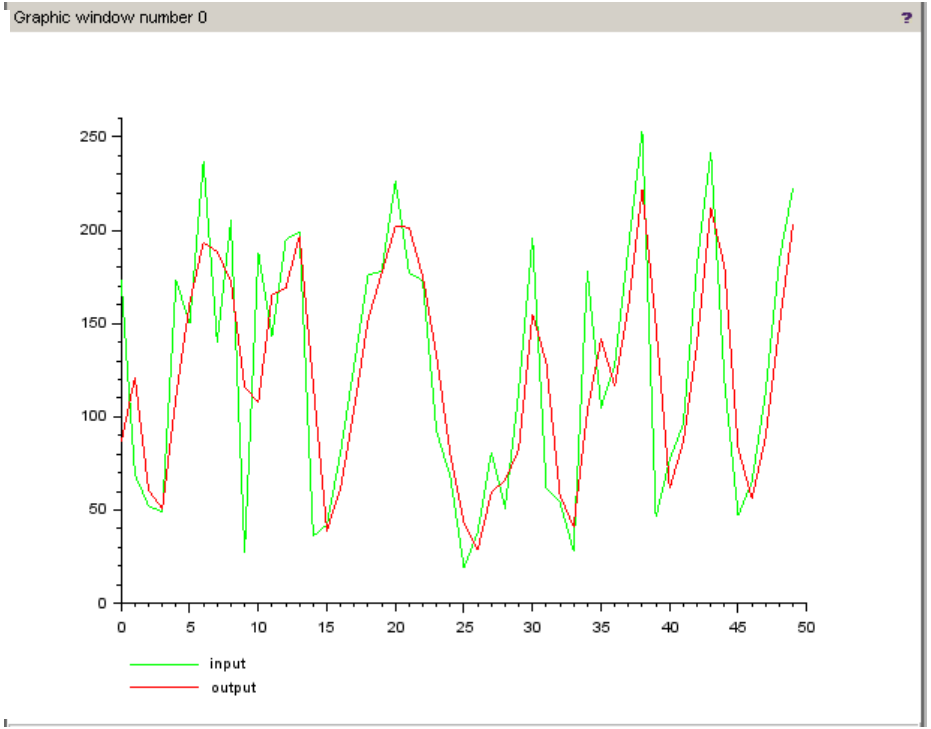
Voorbeeld : Toepassen van een filter op signalen

```
1.  // voorbeeld van FIR-filter:
2.  //y(n))=0.5x(n)+0.5 x(n-1)
3.  clc
4.  clf

5.  // genereren van 50 random samples (ADC-simulatie)
6.  x_waarden = int(255* rand(50,1))

7.  // definiëren filter met twee coëfficiënten 0.5 0.5
8.  // num = 0.5 0.5; dem = 1 (FIR) input = x_waarden

9.  y1_waarden = int(filter([0.5 0.5],[1],x_waarden))

10. //definiëren filter met 4 coëfficiënten
11. Y2_waarden = int(filter([0.25 0.25 0.25 0.25 ],[1],x_waarden))
12. // plot horizontaal "n" (aantal waarden - samples)
13. n = [0 : 1 : 49]

14. // verticaal amplitude van de random x_waarden
15. // plot met legende en aanpassen kleurstijl
16. plot2d (n,[x_waarden,y_waarden], leg ="input@output",...
17. style=[color("green"), color("red")]);
```



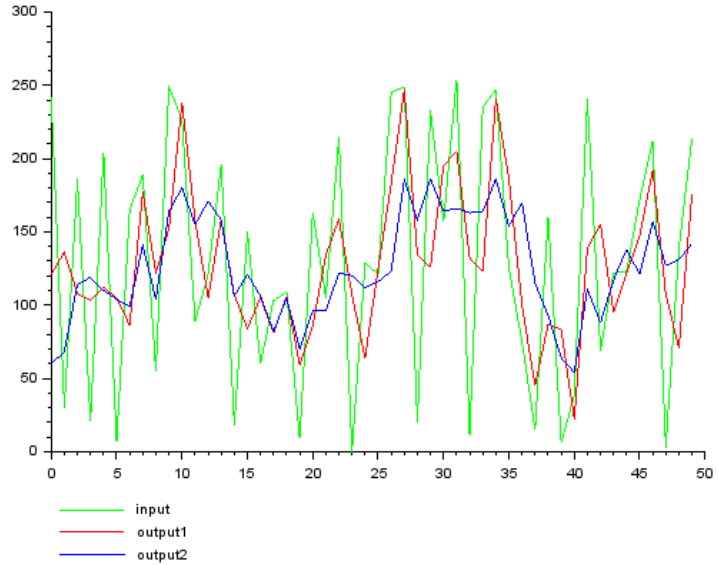
Voorbeeld : Vergelijk met een hogere orde filter

```
1.  // voorbeeld van FIR-filter:
2.  //y(n))=0.5x(n)+0.5 x(n-1)
3.  clc
4.  clf

5.  // genereren van 50 random samples (ADC-simulatie)
6.  x_waarden = int(255* rand(50,1))

7.  // definiëren filter met twee coëfficiënten 0.5 0.5
8.  // num = 0.5 0.5; dem = 1 (FIR) input = x_waarden

9.  y1_waarden = int(filter([0.5 0.5],[1],x_waarden))

10. //definiëren filter met 4 coëfficiënten
11. Y2_waarden = int(filter([0.25 0.25 0.25 0.25 ],[1],x_waarden))
12. // plot horizontaal "n" (aantal waarden - samples)
13. n = [0 : 1 : 49]

14. // verticaal amplitude van de random x_waarden
15. // plot met legende en aanpassen kleurstijl
16. plot2d (n,[x_waarden,y_waarden], leg ="input@output",...
17. style=[color("green"), color("red")]);
```



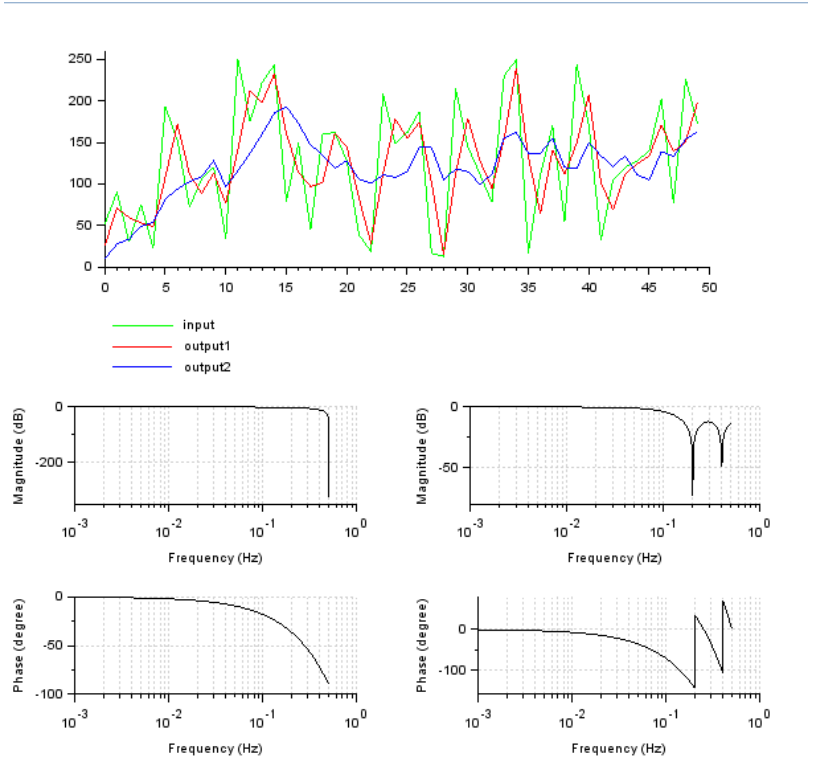
Voorbeeld : Vergelijk met een hogere orde filter

```
1  //·voorbeeld·van·FIR-filter:
2  //y(n))=0.5x(n)+0.5·x(n-1)
3  clc
4  clf
5
6  //·genereren·van·50·random·samples·(ADC-simulatie)
7  x_waarden·=·int(255*·rand(50,1))
8
9  //·definiëren·filter·met·twee·coëfficiënten·0.5·0.5
10 //·num·=·0.5·0.5;·dem·=·1·(FIR)·input·=·x_waarden
11
12 y1_waarden·=·int(filter([0.5·0.5],[1],x_waarden))
13
14 //definiëren·filter·met·4·coëfficiënten
15 y2_waarden·=·int(filter([0.25·0.25·0.25·0.25·],[1],x_waarden))
16 //·plot·horizontaal·"n"·(aantal·waarden·-·samples)
17 n·=·[0·:·1·:·49]
18
19 //·vertikaal·amplitude·van·de·random·x_waarden
20 //·plot·met·legende·en·aanpassen·kleurstijl
21 plot2d·(n,[x_waarden,y1_waarden,·y2_waarden],·leg·="input@output1@output2",...
22 ····style=[color("green"),·color("red"),·color("blue")]);
```



Voorbeeld : Vergelijk met een hogere orde filter

```
//-genereren-van-50-random-samples-(ADC-simulatie)
x_waarden-=-int(255*-rand(50,1))
//-definiëren-filter-met-twee-coëfficiënten-0.5-0.5
//-num-=-0.5-0.5;-dem-=-1-(FIR)-input-=-x_waarden
y1_waarden-=-int(filter([0.5-0.5],[1],x_waarden))
//definiëren-filter-met-4-coëfficiënten
y2_waarden-=-int(filter([0.2-0.2-0.2-0.2-0.2],[1],x_waarden))
//-plot-horizontaal-"n"-(aantal-waarden-=-samples)
n-=-[0-:-1-:-49]
subplot-(2,1,1)
//-vertikaal-amplitude-van-de-random-x_waarden
//-plot-met-legende-en-aanpassen-kleurstijl
plot2d-(n,[x_waarden,y1_waarden,-y2_waarden],-leg-="input@output1@output2",...
----style=[color("green"),-color("red"),-color("blue")]);
para_filter1-=-[0.5-0.5];
para_filter2-=-[0.2-0.2-0.2-0.2-0.2];
filter1_p-=-poly(para_filter1,-'z',-'coeff');
filter2_p-=-poly(para_filter2,-'z',-'coeff');
filter1_h-=-horner(filter1_p,(1/%z));
filter2_h-=-horner(filter2_p,(1/%z));
filter1_s-=-syslin('d',filter1_h);
filter2_s-=-syslin('d',filter2_h);
subplot(2,2,3)
bode-(filter1_s)
subplot(2,2,4)
bode-(filter2_s)
```



Filterontwerp volgens de “window”-methode

Waarvoor wordt een window gebruikt binnen het ontwerp van een digitale filter?

Filterontwerp volgens de “window”-methode

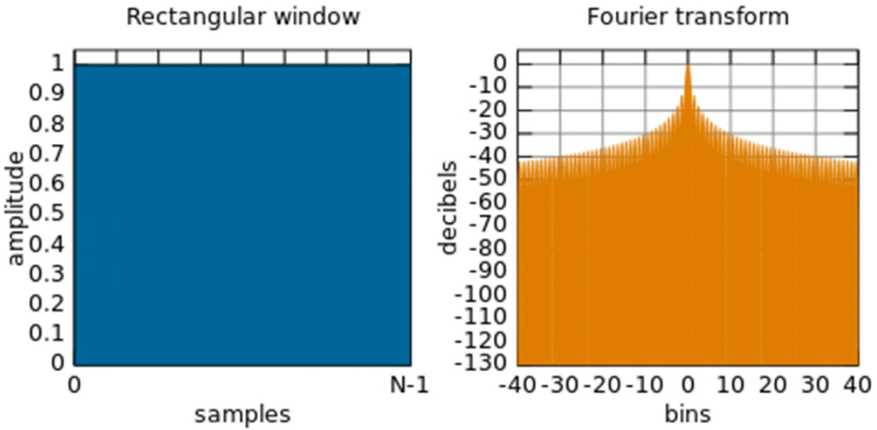
Waarvoor wordt een window gebruikt binnen het ontwerp van een digitale filter?

Om binnen het ontwerp met een ‘ideale’ impulsresponse (bv een sincfunctie die oneindig lang is) deze om te vormen naar een finitive impulse response

Voorbeelden van windows

Rectangular window *(boxcar of Dirichiet)*

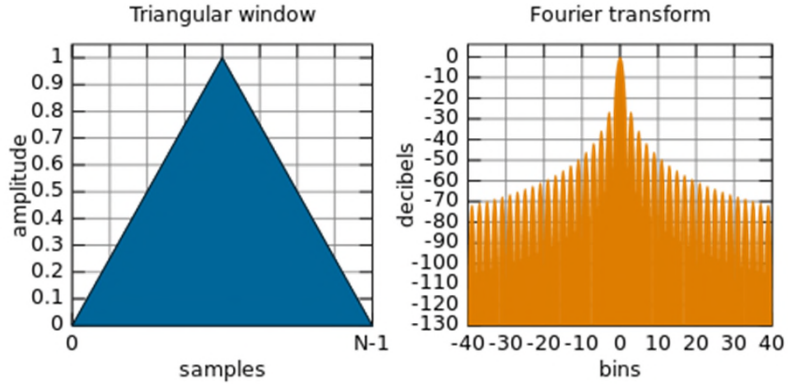
- *Eenvoudigste window-equivalent om alle waarden op nul te krijgen*
- *Alle waarden op nul behalve de waarden 'n' die doorgelaten worden*
- $w(n) = 1$*; met* $w(n)$ *de windowfunctie*
- *Nadeel filter:*
  - *Plotselinge overgang tussen wel doorgelaten en niet doorgelaten ➔ ontstaan ongewenste effecten in de DTFT (discrete time fourier transformatie)*



Voorbeelden van windows

Triangular window

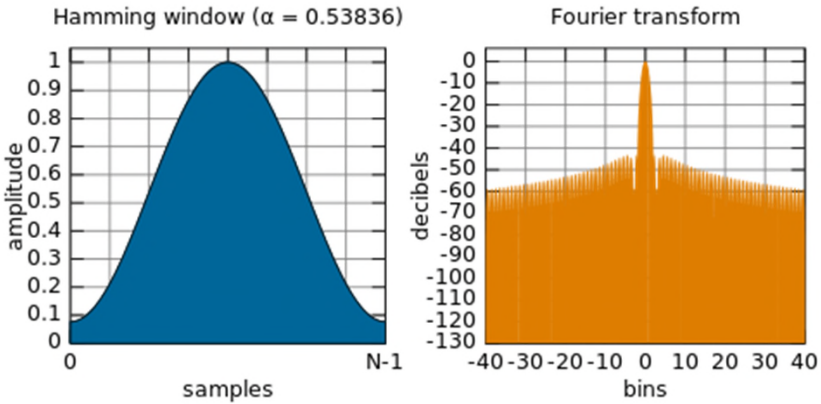
- $w(n) = 1 - n - \frac{\frac{N-1}{2}}{\frac{L}{2}}$
- ***met L gelijk aan $N, N+1$ of $N-1$***
- ***Kan gezien worden als de convolutie van twee $N/2$ brede reactangular windows***



Voorbeelden van 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**

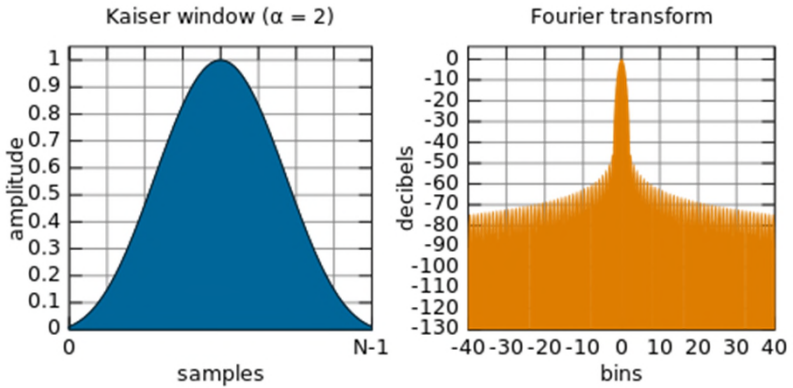


Hamming window, α = 0.53836 and β = 0.46164; B = 1.37. The original Hamming window would have α = 0.54 and β = 0.46; B = 1.3628.

Voorbeelden van windows

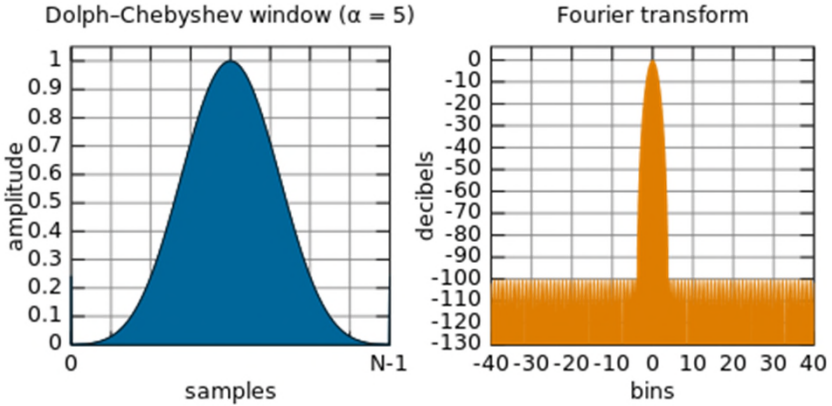
Kaiser window *(Kaiser-Bessel window)*

- 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.



Voorbeelden van windows

- **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 $W_0(k)$.



Waarvoor worden windows weer gebruikt bij digitale filters?

- Om binnen het ontwerp met een ‘ideale’ impulsresponse (bv een sincfunctie die oneindig lang is) deze om te vormen naar een finitive impulse response

Filterontwerp met de functie 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

Filterontwerp met de functie 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])

- De returnwaarden van de functie wfir() zijn:
  - ***coëfficiënten :*** 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

Voorbeeld filterontwerp met wfir()

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

Voorbeeld filterontwerp met 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*

Voorbeeld filterontwerp met 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]);

- De returnwaarden van de functie wfir() moeten vervolgens omgezet worden naar het scilab-formaat zodat de functie flts() gebruik kan maken van de filterparameters.
- ***flts() is een functie die de filter in zijn tijdsdomeinresponse kan weergeven***

Hoe creëer je een filter via scilab met wfir()?

Via wfir() kan je de coëffiiënten bepalen die nodig zijn om het filter te vormen.

Met deze coëfficiënten wordt de filter kernel gevormd. (vermenigvuldigd met de wiskundige formule van het window)

Via een for-loop wordt vervolgens een convolutie uitgevoerd tussen de samples van het ingangssignaal en de filterkernel (zie hoofdstuk windowed sync filters)

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

Hoe weet je nu zeker dat de filter doet wat je er van verwacht?

Hoe weet je nu zeker dat de filter doet wat je er van verwacht?

Door het maken van een testsignaal om de filter uit te testen

De filter heeft een $f_c = 40\ Hz$

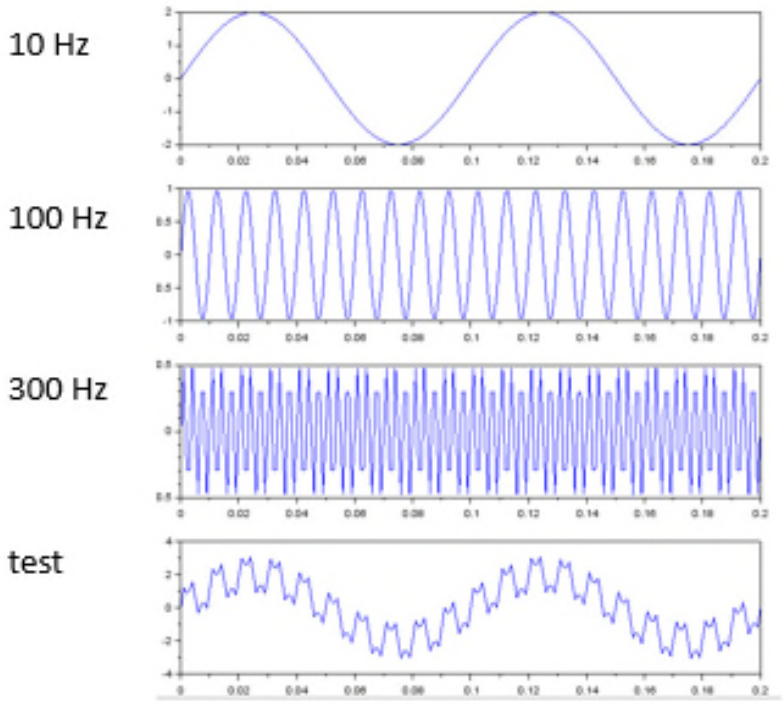
Via een gecombineerd signaal, een in de doorlaatband, een relatief dicht aan f_c en een iets verder in het spergebied kan je de filter uittesten

Hoe weet je nu zeker dat de filter doet wat je er van verwacht?

Door het maken van een testsignaal om de filter uit te testen

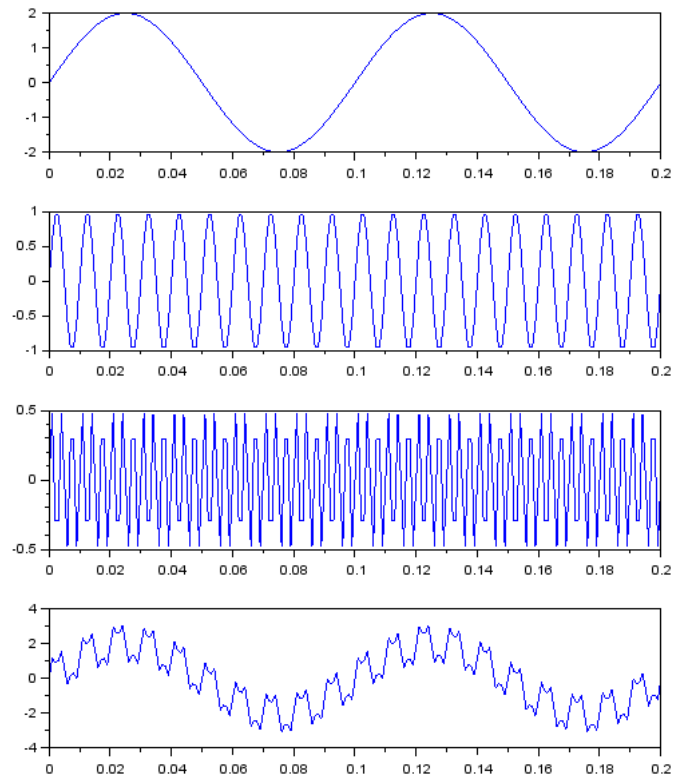
De filter heeft een $f_c = 40\ Hz$

Via een gecombineerd signaal, een in de doorlaatband, een relatief dicht aan f_c en een iets verder in het spergebied kan je de filter uittesten



Hoe weet je nu zeker dat de filter doet wat je er van verwacht?

```
clc
clf
//genereren van een testsignaal
// 10 Hz, 100 Hz, 300 Hz
// fs = 1 kHz
// tijdsduur testsignaal : 200 ms; interval 1 µs
t = [0 : 0.001 : 0.2];
// genereren sinus 10 HZ; Vp = 2 V
sin_10Hz = 2*sin(2*%pi*10*t);
// genereren sin 100 Hz Vp = 1 V
sin_100Hz = 1*sin(2*%pi*100*t);
// genereren sin 300 Hz Vp = 0,5 V
sin_300Hz = 0.5*sin(2*%pi*300*t);
// genereren testsignaal
testsign = sin_10Hz + sin_100Hz + sin_300Hz;
// weergeven alle signalen
subplot(411)
plot(t,sin_10Hz);
subplot(412)
plot(t,sin_100Hz);
subplot (413)
plot(t,sin_300Hz);
subplot(414)
plot(t,testsign);
```



Hoe weet je nu zeker dat de filter doet wat je er van verwacht?

- Via flts() kan de filter worden doorlopen met het testsignaal

LD_output = flts(testsign, LD_linear_system)

- testsign : vector dat het te filteren signaal bevat
- LD_linear_system : filterkernel gevormd uit de coëfficiënten bepaald via wfir()

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. //filterontwerp lp orde 40 fc 40 Hz, fs/1 kHz BW 40 Hz
12. [LD_coeff, amplitude, frequentie]  = ...
13.       wfir('lp', 40, [0.04 0], 'hm', [0 0]);
14. LD_polynoom = poly(LD_coeff, 'z', 'coeff');
15. LD_functie = horner(LD_polynoom, (1/%z));
16. LD_lineair_system = syslin('d', LD_functie);
17. // testen werking van de filter
18. //filteren via scilab met functie flts()
19. LD_output = flts(testsign, LD_lineair_system)
20. // weergeven van testsignaal
21. subplot (311)
22. plot(t,testsign)
23. //weergeven filteroutput
24. subplot (312)
25. plot(t, LD_output, 'r');
26. subplot(313)
27. plot(t, testsign)
28. plot (t, LD_output, 'r')
29. plot (t, sin_10Hz, 'k')
```

$f_s = 1\ kHz; tijdsduur\ 200ms$

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. //filterontwerp lp orde 40 fc 40 Hz, fs/1 kHz BW 40 Hz
12. [LD_coeff, amplitude, frequentie]  = ...
13.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
14. LD_polynoom = poly(LD_coeff, 'z', 'coeff');
15. LD_functie = horner(LD_polynoom, (1/%z));
16. LD_lineair_system = syslin('d', LD_functie);
17. // testen werking van de filter
18. //filteren via scilab met functie flts()
19. LD_output = flts(testsign, LD_lineair_system)
20. // weergeven van testsignaal
21. subplot (311)
22. plot(t,testsign)
23. //weergeven filteroutput
24. subplot (312)
25. plot(t, LD_output, 'r');
26. subplot(313)
27. plot(t, testsign)
28. plot (t, LD_output, 'r')
29. plot (t, sin_10Hz, 'k')
```

Bepalen van filtercoëfficiënten (LD_coeff), amplitudes en frequenties voor het maken van een 40ste orde LD-filter met fc = 40 Hz en fs = 1 kHz (waarden zijn genormaliseerd in code)

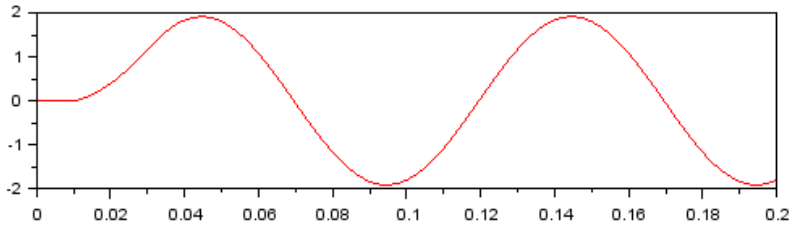
```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. //filterontwerp lp orde 40 fc 40 Hz, fs/1 kHz BW 40 Hz
12. [LD_coeff, amplitude, frequentie]  = ...
13.       wfir('lp', 40, [0.04 0], 'hm', [0 0]);
14. LD_polynoom = poly(LD_coeff, 'z', 'coeff');
15. LD_functie = horner(LD_polynoom, (1/%z));
16. LD_lineair_system = syslin('d', LD_functie);
17. // testen werking van de filter
18. //filteren via scilab met functie flts()
19. LD_output = flts(testsign, LD_lineair_system)
20. // weergeven van testsignaal
21. subplot (311)
22. plot(t,testsign)
23. //weergeven filteroutput
24. subplot (312)
25. plot(t, LD_output, 'r');
26. subplot(313)
27. plot(t, testsign)
28. plot (t, LD_output, 'r')
29. plot (t, sin_10Hz, 'k')
```

Omzetten van berekende filtercoëfficiënten naar waarden die voor scilab herkenbaar zijn als filtercoëfficiënten

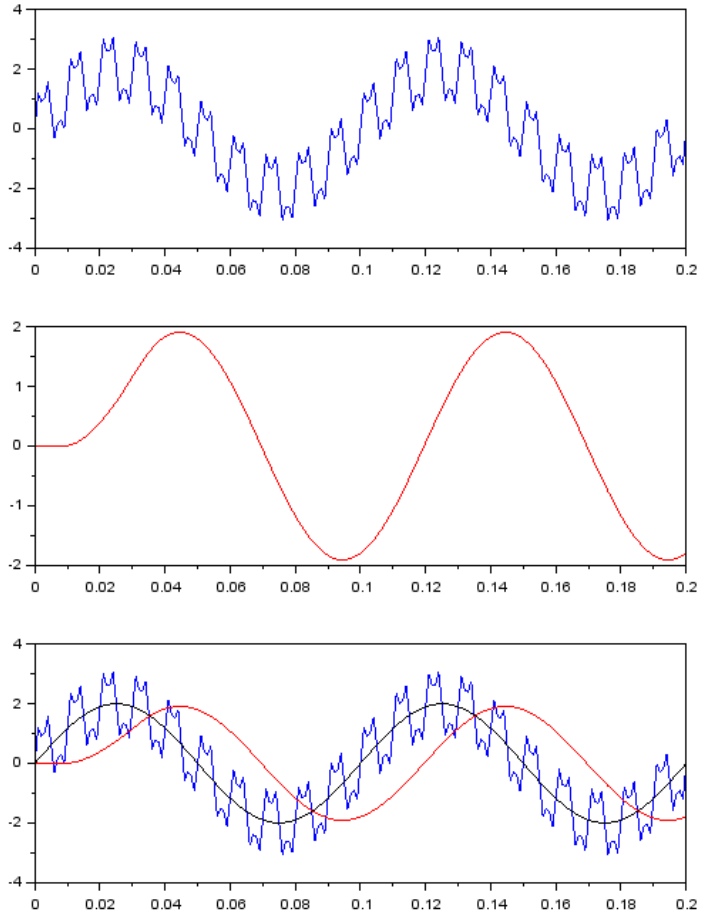
AP HOGESCHOOL ANTWERPEN

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. //filterontwerp lp orde 40 fc 40 Hz, fs/1 kHz BW 40 Hz
12. [LD_coeff, amplitude, frequentie]  = ...
13.       wfir('lp', 40, [0.04 0], 'hm', [0 0]);
14. LD_polynoom = poly(LD_coeff, 'z', 'coeff');
15. LD_functie = horner(LD_polynoom, (1/%z));
16. LD_lineair_system = syslin('d', LD_functie);
17. // testen werking van de filter
18. //filteren via scilab met functie flts()
19. LD_output = flts(testsign, LD_lineair_system)
20. // weergeven van testsignaal
21. subplot (311)
22. plot(t,testsign)
23. //weergeven filteroutput
24. subplot (312)
25. plot(t, LD_output, 'r');
26. subplot(313)
27. plot(t, testsign)
28. plot (t, LD_output, 'r')
29. plot (t, sin_10Hz, 'k')
```

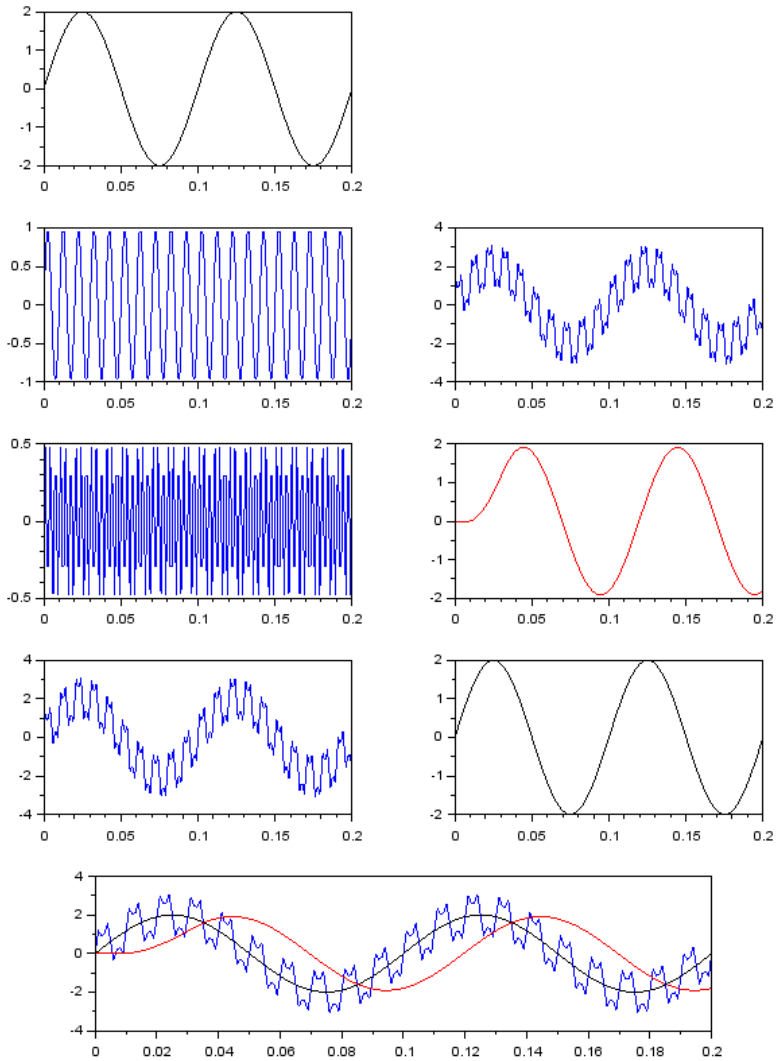
Het uitgangssignaal van de filter, weergegeven in het tijdsdomein



```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. //filterontwerp lp orde 40 fc 40 Hz, fs/1 kHz BW 40 Hz
12. [LD_coeff, amplitude, frequentie]  = ...
13.       wfir('lp', 40, [0.04 0], 'hm', [0 0]);
14. LD_polynoom = poly(LD_coeff, 'z', 'coeff');
15. LD_functie = horner(LD_polynoom, (1/%z));
16. LD_lineair_system = syslin('d', LD_functie);
17. // testen werking van de filter
18. //filteren via scilab met functie flts()
19. LD_output = flts(testsign, LD_lineair_system)
20. // weergeven van testsignaal
21. subplot (311)
22. plot(t,testsign)
23. //weergeven filteroutput
24. subplot (312)
25. plot(t, LD_output, 'r');
26. subplot(313)
27. plot(t, testsign)
28. plot (t, LD_output, 'r')
29. plot (t, sin_10Hz, 'k')
```

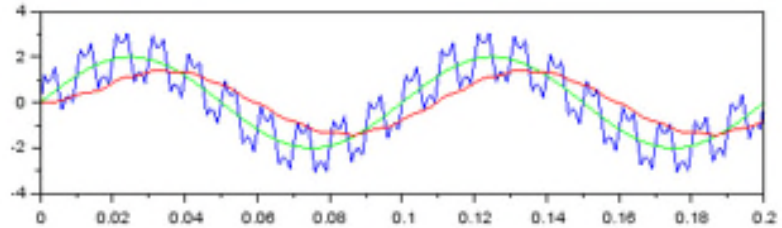


```
t = [0:0.001:0.2];
sin_10Hz = 2*sin(2*%pi*10*t);
sin_100Hz = 1*sin(2*%pi*100*t);
sin_300Hz = 0.5*sin(2*%pi*300*t);
testsign = sin_10Hz + sin_100Hz + sin_300Hz
//filterontwerp lp orde 40 fc 40 Hz, fs 1 kHz BW 40 Hz
[LD_coeff, amplitude, frequentie] = ...
    wfir('lp', 40, [0.04 0], 'hm', [0 0]);
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(521)
plot(t, sin_10Hz,'k')
subplot(523)
plot(t, sin_100Hz)
subplot(525)
plot(t, sin_300Hz)
subplot(524)
plot(t, testsign)
subplot(527)
plot(t, testsign)
subplot(526)
plot(t,LD_output, 'r')
subplot (528)
plot(t, sin_10Hz,'k')
subplot (515)
plot(t, testsign)
plot (t, LD_output, 'r')
plot (t, sin_10Hz, 'k')
```

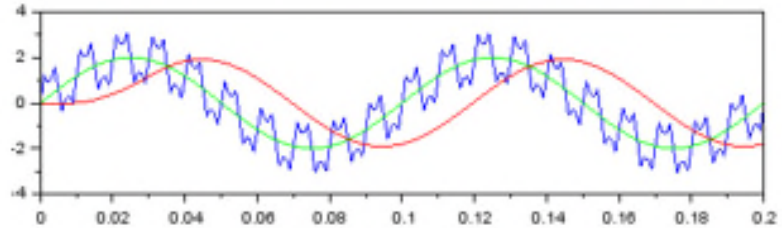


Invloed van de orde op de filter

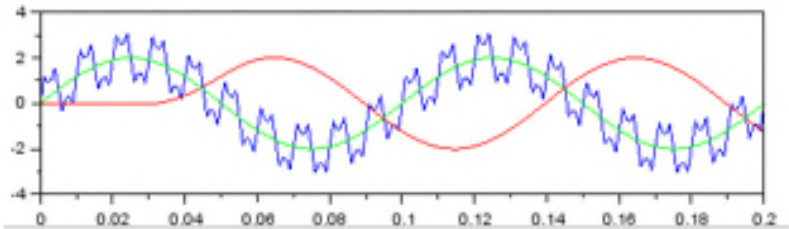
Orde 20

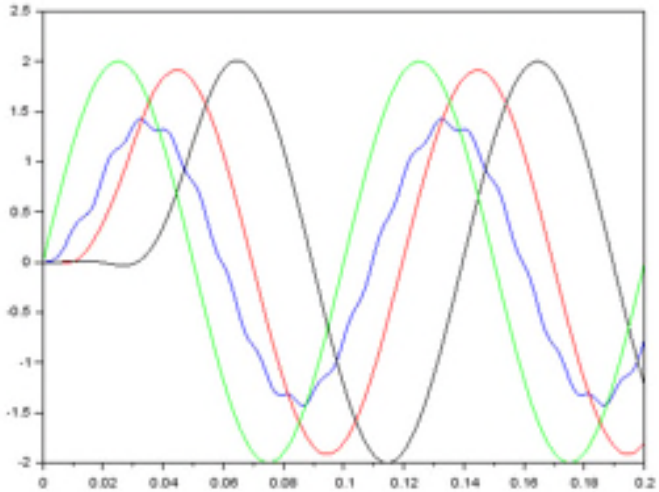


Orde 40



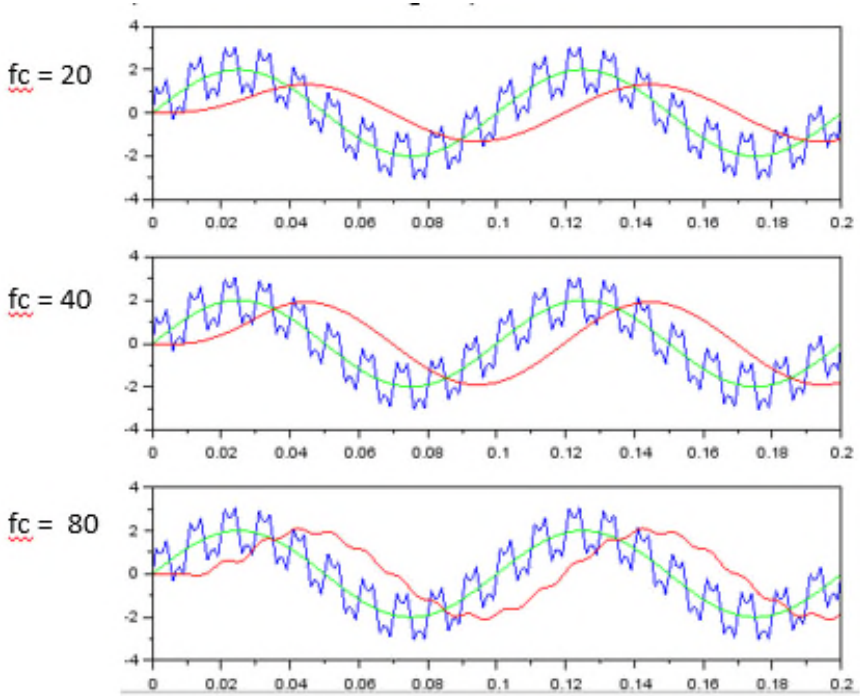
Orde 80

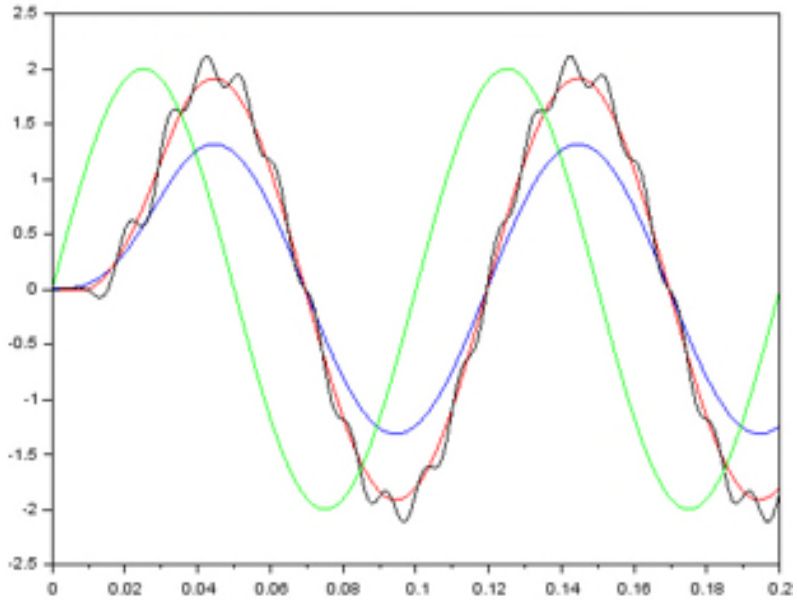




Groen: originele sinus 10 Hz
Blauw : filteroutput 20ste orde filter
Rood : filteroutput 40ste orde filter
Zwart : filteroutput 80ste orde filter

Invloed van de afsnijffrequentie op de filter

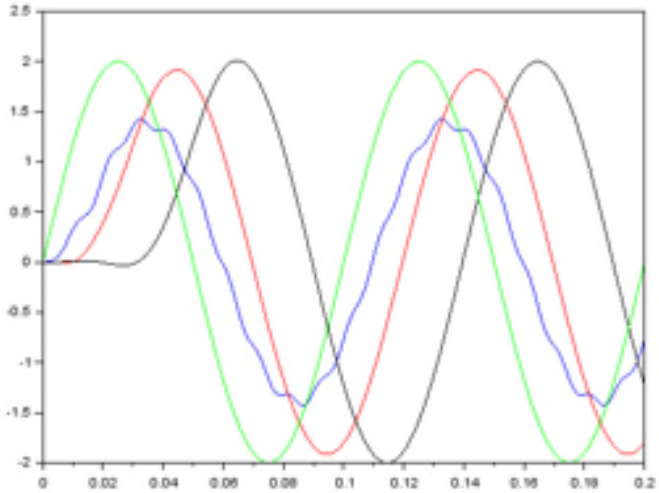




Groen: originele sinus 10 Hz
Blauw : fc = 20 Hz
Rood : fc = 40 Hz
Zwart: fc = 80 Hz

Invloed van de orde op de filter

- ***Hoe lager de filterorde bij dezelfde afsnijfrequentie (40 Hz), hoe meer gedempt 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

Even overlopen …

- ***Hoe creëer je nu weer een FIR-filter met lineaire faseresponse in scilab?***

Even overlopen …

- ***Hoe creëer je nu weer een FIR-filter met lineaire faseresponse in scilab?***

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

Wfir heeft volgende parameters nodig:

filter-type

Filter-orde

[fg1 fg2]

window-type : re, tr, hm, kr, ch
(rectangular, triangular, hamming, kaiser, chebyshev)

[par1 par2]

Even overlopen …

- ***Hoe creëer je nu weer een FIR-filter met lineaire faseresponse in scilab?***

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

Wfir heeft volgende parameters nodig:

filter-type

Filter-orde

[fg1 fg2]

window-type : re, tr, hm, kr, ch (rectangular, triangular, hamming, kaiser, chebyshev)

[par1 par2]

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

Even overlopen …

- ***Hoe maak je scilab duidelijk dat het een filter is?***

```
[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);
```

Even overlopen …

- ***Hoe maak je het resultaat van de filter zichtbaar in het tijdsdomein?***

- Via flts() kan de filter worden doorlopen met het testsignaal
  - LD_output = flts(testsign, LD_linear_system)

Even overlopen …

- ***Wat is de invloed van de orde en afsnijfrequentie op het uitgangssignaal van een FIR-filter met lineaire response?***

Even overlopen …

- ***Wat is de invloed van de orde en afsnijfrequentie op het uitgangssignaal van een FIR-filter met lineaire response?***

- **FIR-filter heeft een relatief hoge filterorde nodig om tot goede resultaten te bekomen**
  ➔Hoe hoger de orde, hoe groter de faseverschuiving

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

Hoe kan je het spectrum van de filter zichtbaar maken in scilab?

- Met de functie fft() kan je een signaal decomposeren in de sinuscompenten waarmee dit signaal is samengesteld.

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot Nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
```

Geeft weer welke frequenties vervat zitten in het singaal "testsign"

AP HOGESCHOOL ANTWERPEN

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot Nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
```

Nagaan uit hoeveel tijdsamples het testsignaal "testsign" bestaat

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot Nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
```

De tijdsas omvormen naar frequentie-as (0 Hz tot Nyquistfrequentie

$f_s = 1000\ Hz$

$\Rightarrow as\ gaat\ tot\ 500\ Hz$

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot Nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
```

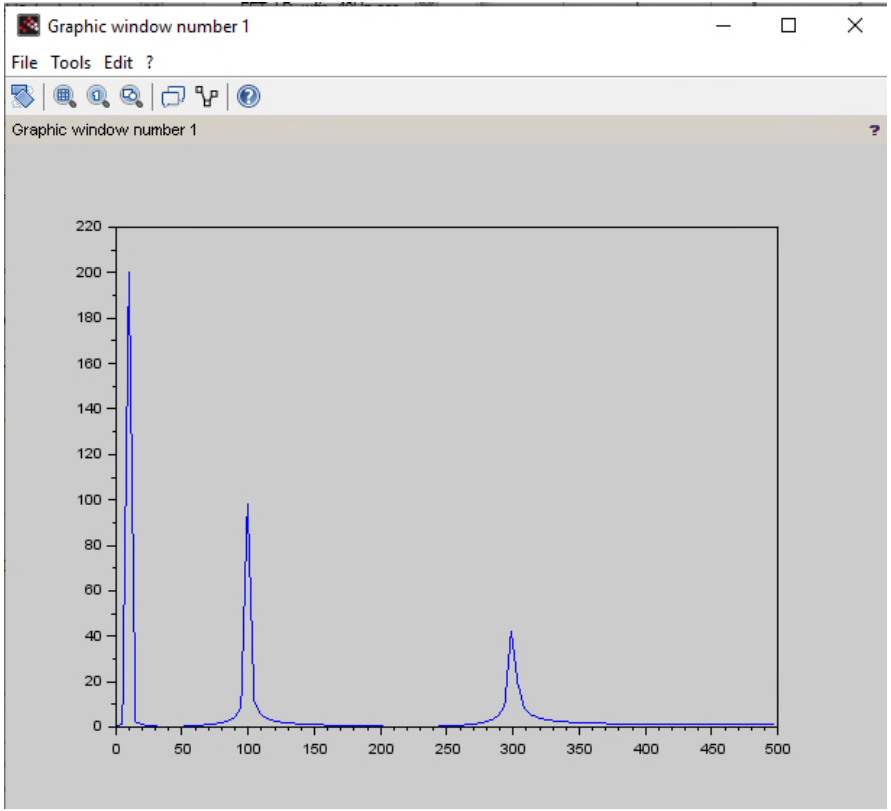
Nagaan hoe lang de vector f is

```
1.  clc
2.  clf
3.  //filteren testsignaal met FIR
4.  //fc filter is 40 Hz
5.  //testsignaal
6.  t = [0:0.001:0.2];
7.  sin_10Hz = 2*sin(2*%pi*10*t);
8.  sin_100Hz = 1*sin(2*%pi*100*t);
9.  sin_300Hz = 0.5*sin(2*%pi*300*t);
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot Nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
```

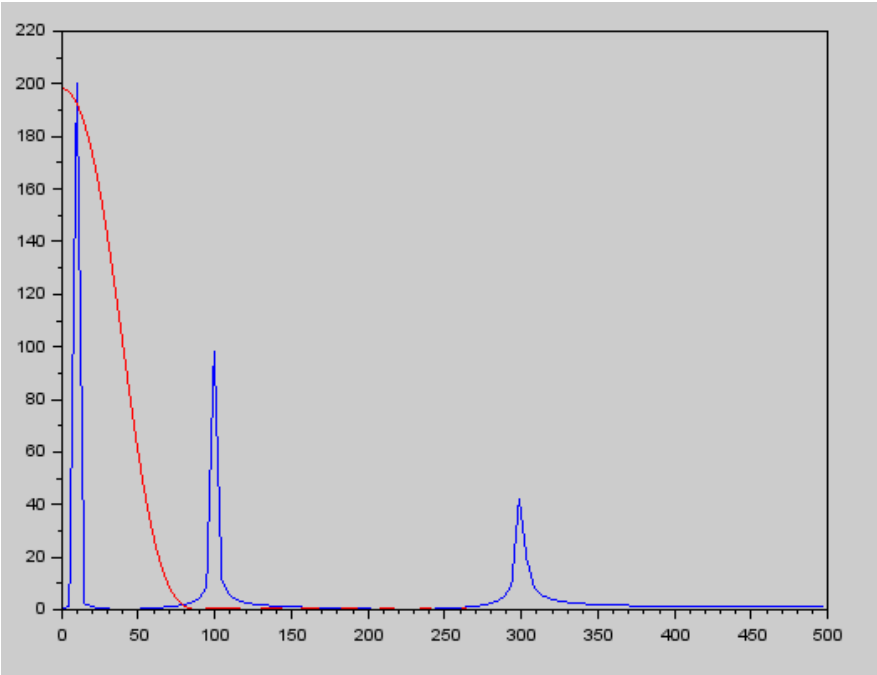
Een nieuw graphic window openen

Zichtbaar maken van de sinuscomponenten in het testsignaal

```
clc
clf
//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
//weergeven van frequentiespectrum
Frequentie_FFT = (abs(fft(testsign)))
//aantal samples
N = size(t, '*');
//fs = 1000 Hz
//geassocieerde frequentievector (tot nyquist)
f = 1000*(0:(N/2))/N;
//bepalen lengte f-vector
n=size(f, '*');
//weergeven figuur
figure;
plot (f, Frequentie_FFT(1:n));
```



Zichtbaar maken van de frequentie filterkarakteristiek tezamen met de frequenties van het testsignaal



Zichtbaar maken van de frequentie filterkarakteristiek tezamen met de frequenties van het testsignaal

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

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

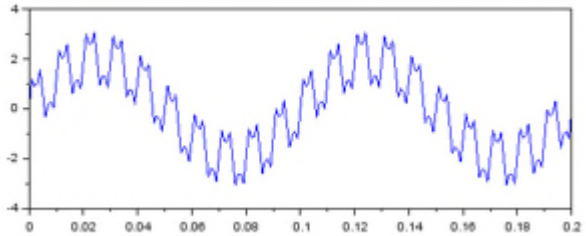
De returnwaarden van de functie wfir() zijn:

coefficients : filtercoëfficiënten

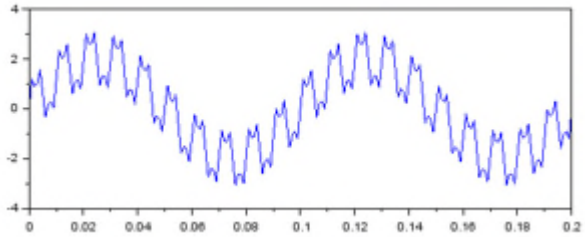
amplitude : vector met lengte 256 met de amplitudewaarden

frequency : vector met lengte 256 met de frequenties in het gebied tussen 0 tot 0,5

```
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
23. // gebruik maken van de frequentie en amplitudewaarde
24. // om frequentiekarakteristiek LD-filter weer te geven
25. [LD_coeff, amplitude, frequentie]  = ...
26.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
27. plot (frequentie*1000, amplitude*N,'r')
```

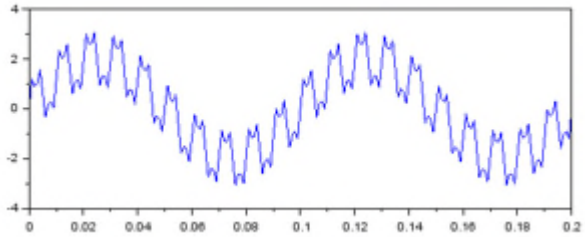


```
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
23. // gebruik maken van de frequentie en amplitudewaarde
24. // om frequentiekarakteristiek LD-filter weer te geven
25. [LD_coeff, amplitude, frequentie]  = ...
26.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
27. plot (frequentie*1000, amplitude*N,'r')
```



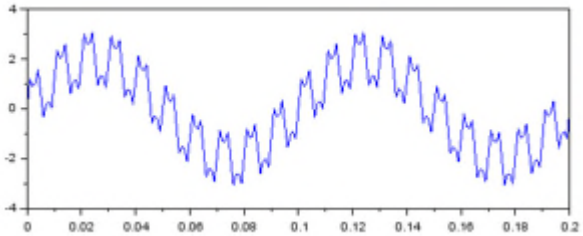
Geeft weer welke frequenties vervat zitten in het singaal "testsign"

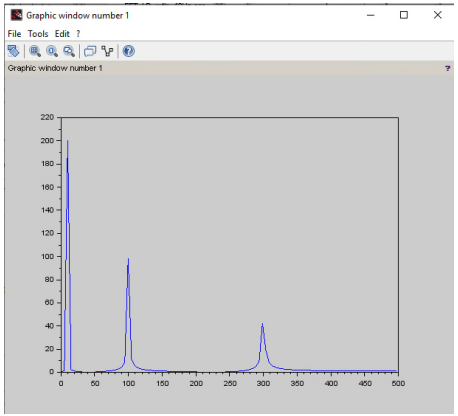
```
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
23. // gebruik maken van de frequentie en amplitudewaarde
24. // om frequentiekarakteristiek LD-filter weer te geven
25. [LD_coeff, amplitude, frequentie]  = ...
26.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
27. plot (frequentie*1000, amplitude*N,'r')
```



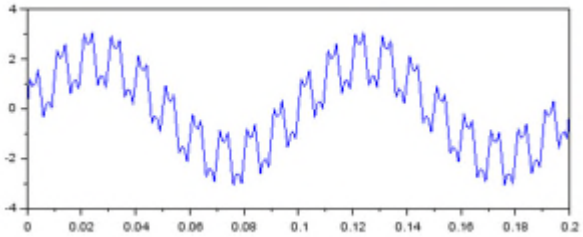
Nagaan uit hoeveel samples het testsignaal bestaat

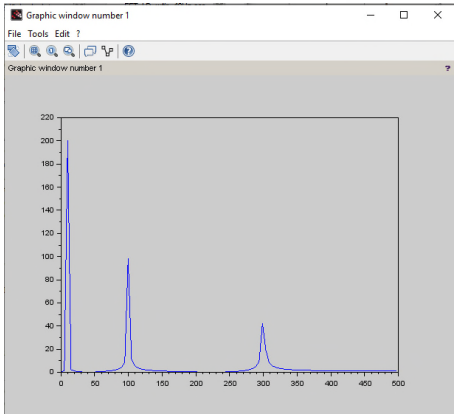
```
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
23. // gebruik maken van de frequentie en amplitudewaarde
24. // om frequentiekarakteristiek LD-filter weer te geven
25. [LD_coeff, amplitude, frequentie]  = ...
26.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
27. plot (frequentie*1000, amplitude*N,'r')
```





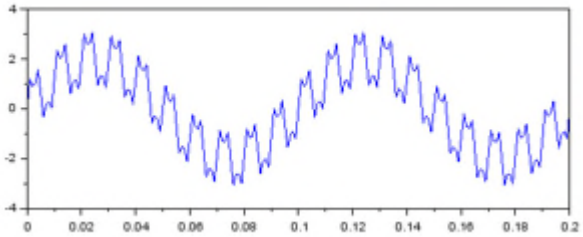
```
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
23. // gebruik maken van de frequentie en amplitudewaarde
24. // om frequentiekarakteristiek LD-filter weer te geven
25. [LD_coeff, amplitude, frequentie]  = ...
26.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
27. plot (frequentie*1000, amplitude*N,'r')
```

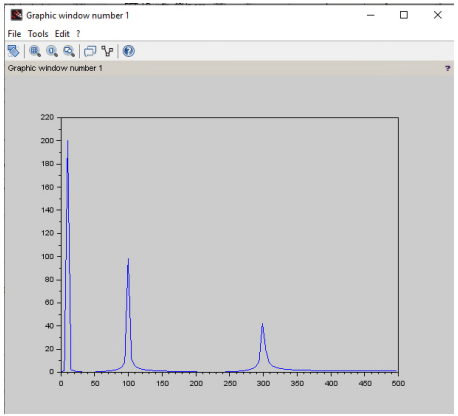


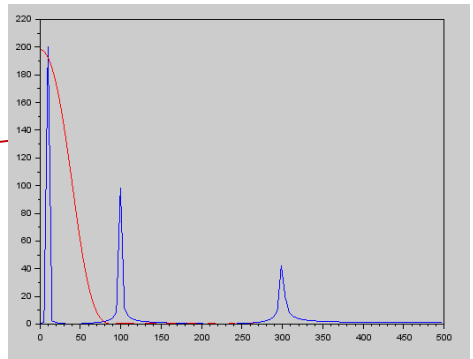


Bepalen van frequentie-componenten en amplitude-waarden van de filterkarakteristiek

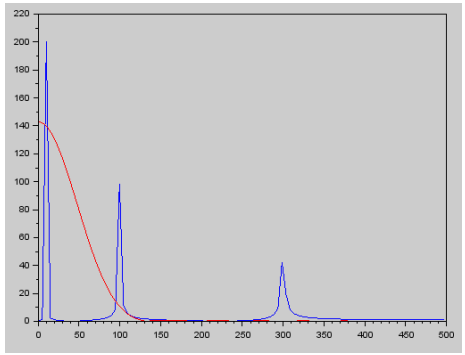
```
10. testsign = sin_10Hz + sin_100Hz + sin_300Hz
11. // weergeven van frequentiespectrum
12. Frequentie_FFT = (abs(fft(testsign)))
13. //aantal samples
14. N = size(t, '*');
15. // fs = 1000 Hz
16. // geassocieerde frequentievector (tot nyquist)
17. f = 1000*(0:(N/2))/N;
18. //bepalen lengte f-vector
19. n=size(f, '*');
20. // weergeven figuur
21. figure;
22. plot (f, Frequentie_FFT(1:n));
23. // gebruik maken van de frequentie en amplitudewaarde
24. // om frequentiekarakteristiek LD-filter weer te geven
25. [LD_coeff, amplitude, frequentie]  = ...
26.     wfir('lp', 40, [0.04 0], 'hm', [0 0]);
27. plot (frequentie*1000, amplitude*N,'r')
```



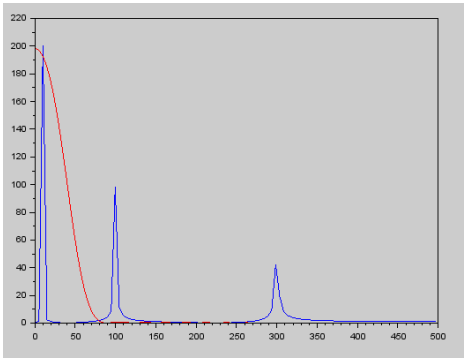




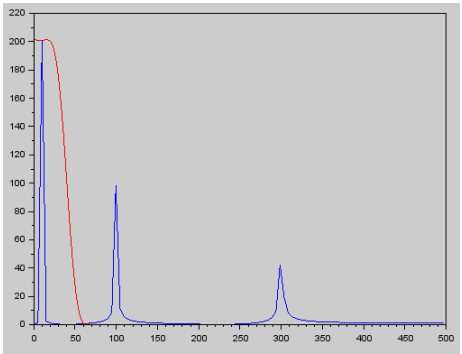
Invloed van de orde op de frequentiekarakteristiek



Orde = 20

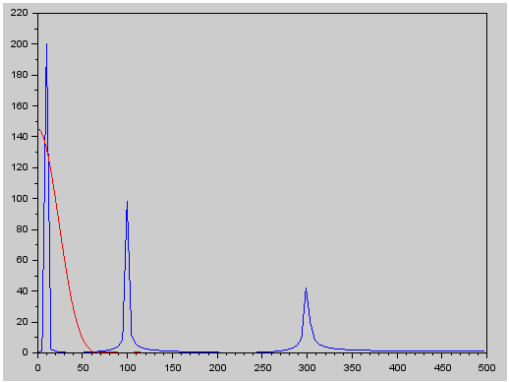


Orde = 40

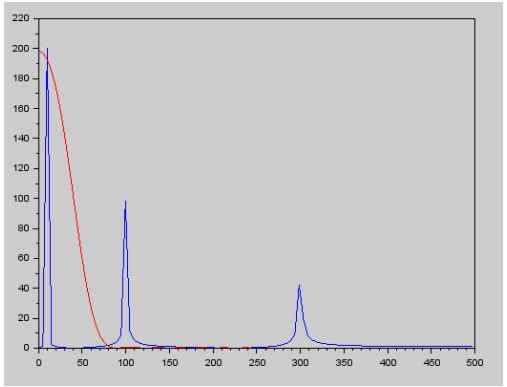


orde = 80

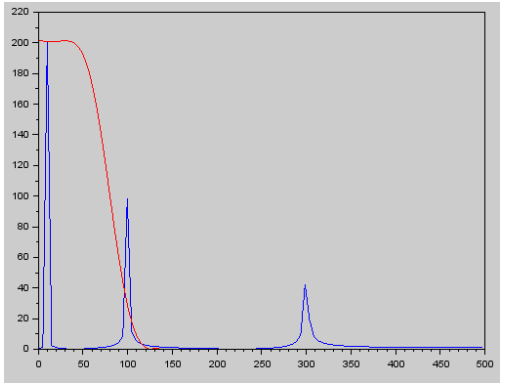
Invloed van de afsnijfrequentie op de frequentiekarakteristiek



$f_c = 20\ Hz$

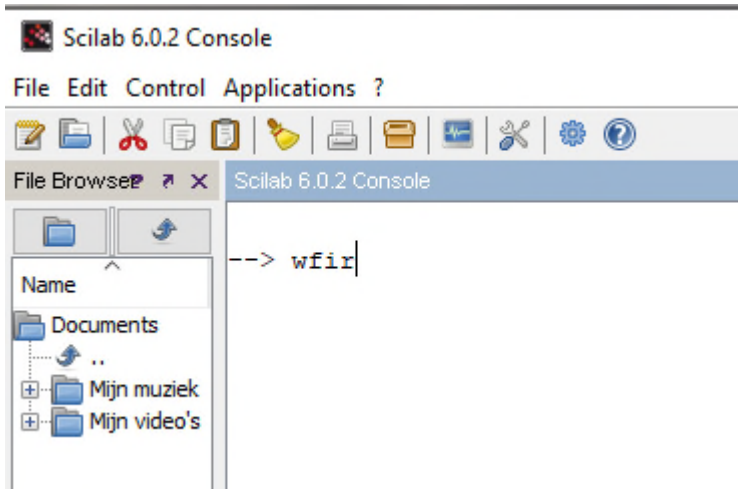


$f_c = 40\ Hz$

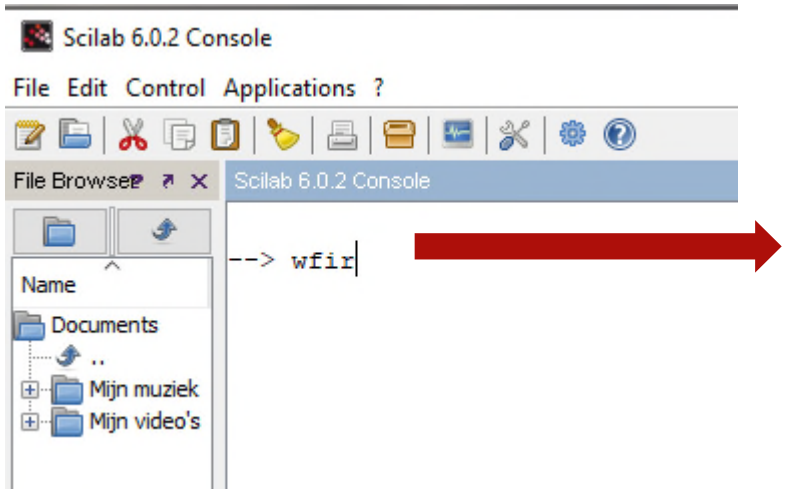


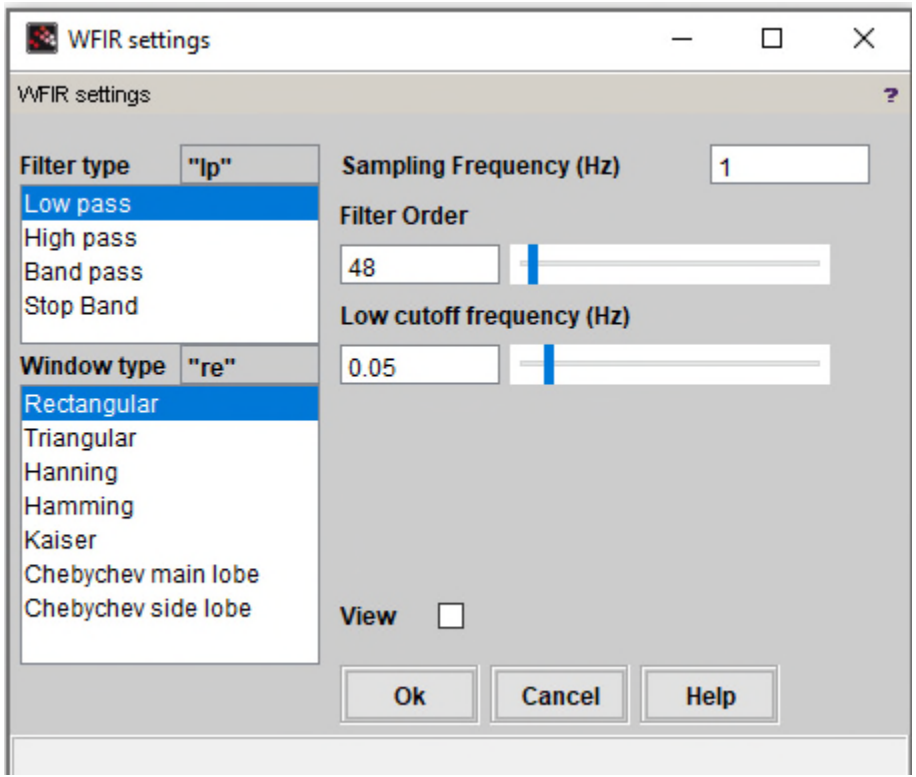
$f_c = 80\ Hz$

Snel en eenvoudig een FIR-filter ontwerpen via wfir in console



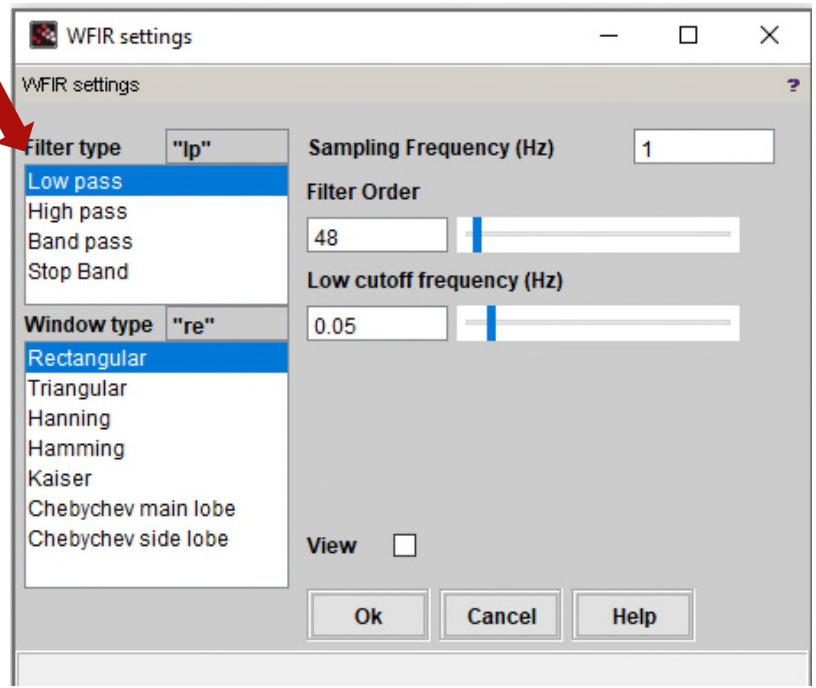
Snel en eenvoudig een FIR-filter ontwerpen via wfir in console





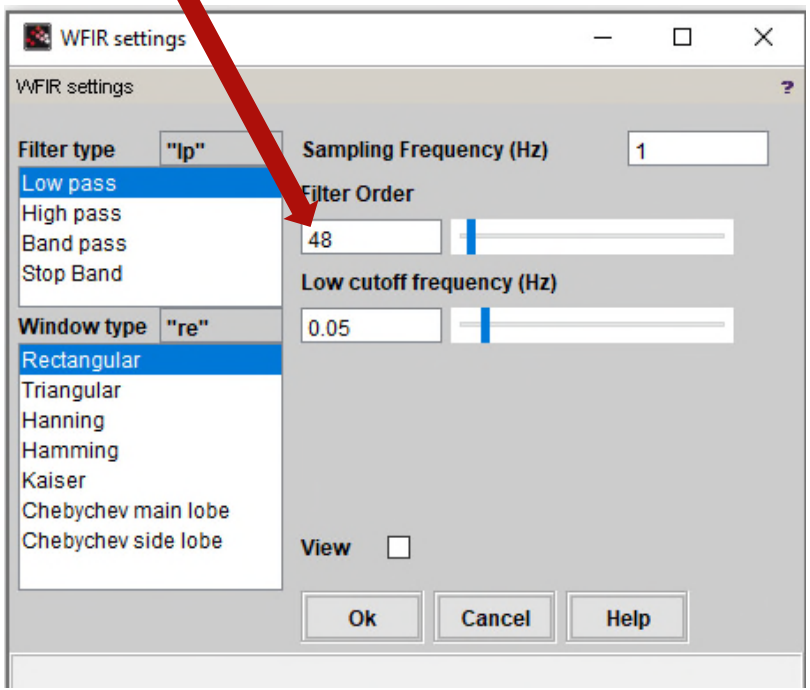
Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

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



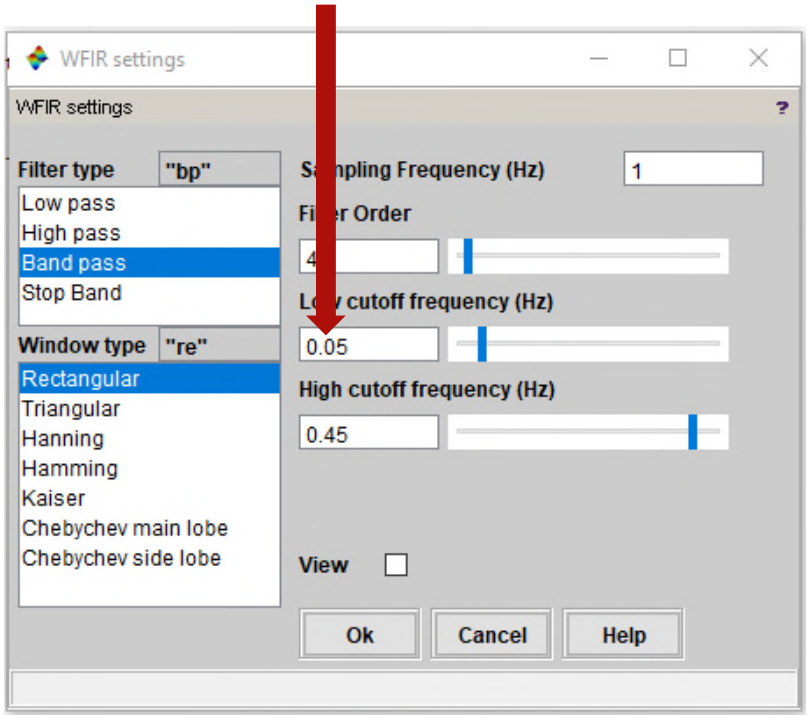
Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

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



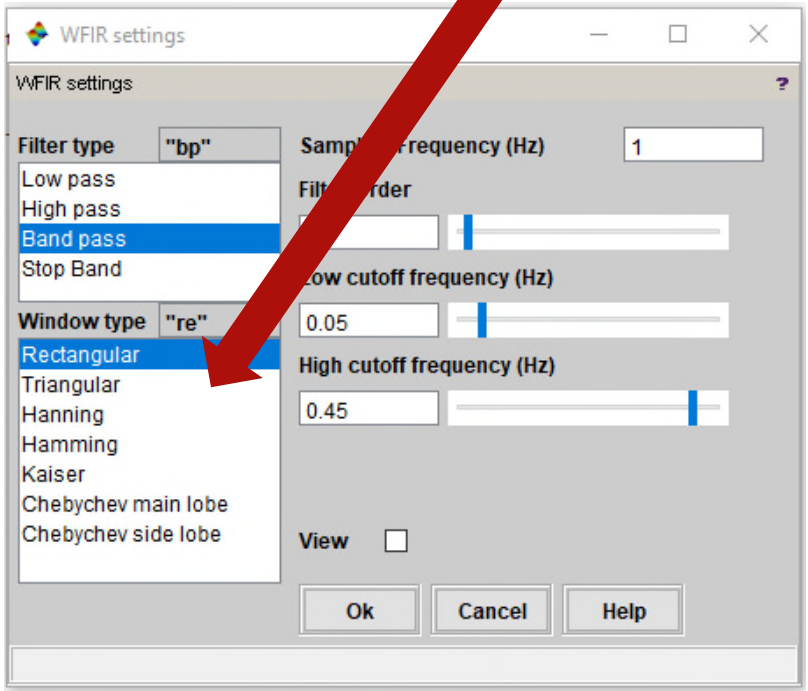
Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

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



Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

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



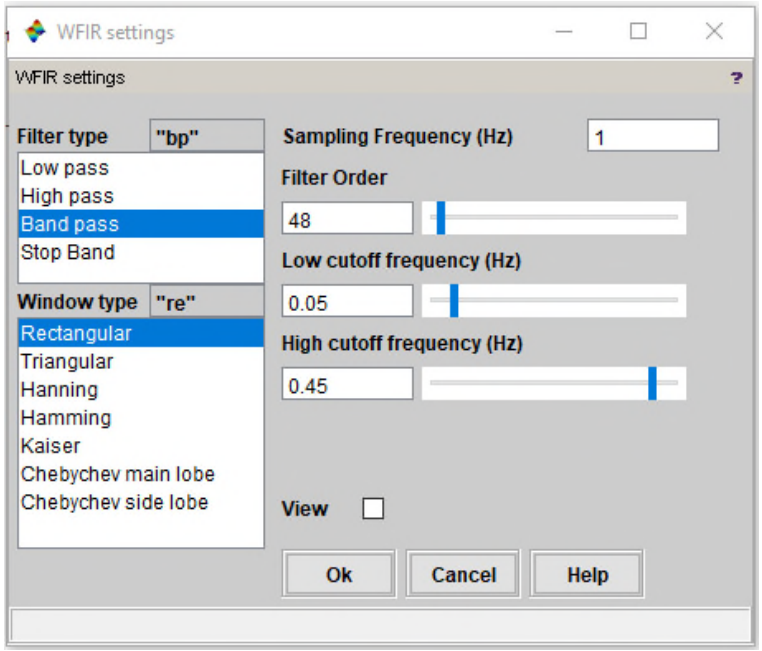
Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

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

Stel we wensen een banddoorlaatfilter voor spraak te bekomen (300 Hz – 3300 Hz)

De frequenties 100 Hz en 3600 Hz moeten minstens 30 dB verzwakt worden

$f_s = 8kHz$



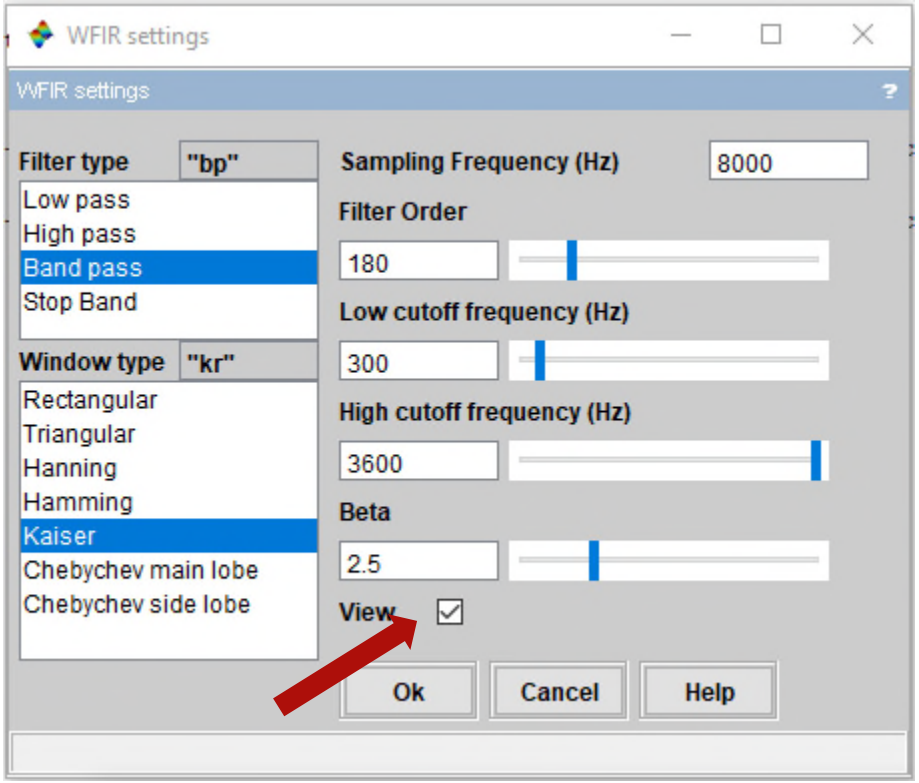
Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

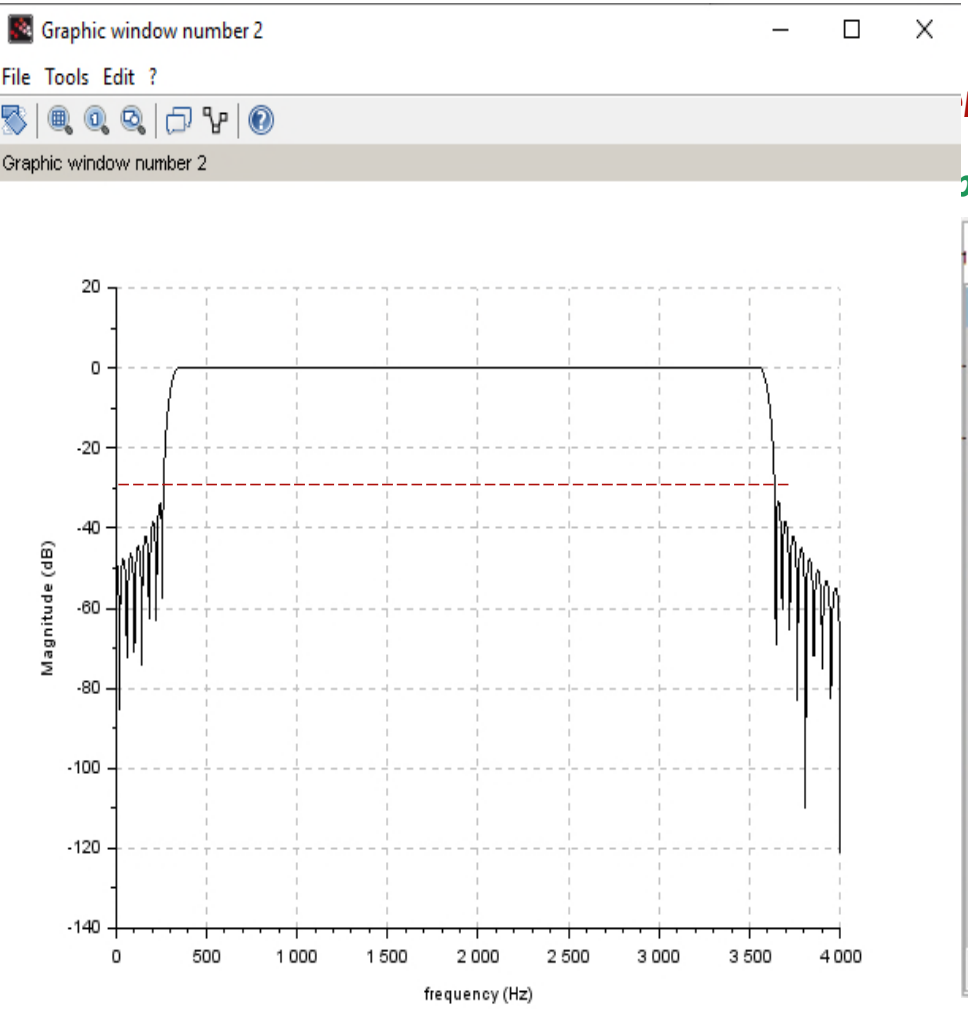
[coefficients, amplitude, frequency] = wfir(filter-type, filter-order, [fg1 fg2], windowtype, [par1 par2])

Stel we wensen een banddoorlaatfilter voor spraak te bekomen (300 Hz – 3300 Hz)

De frequenties 100 Hz en 3600 Hz moeten minstens 30 dB verzwakt worden

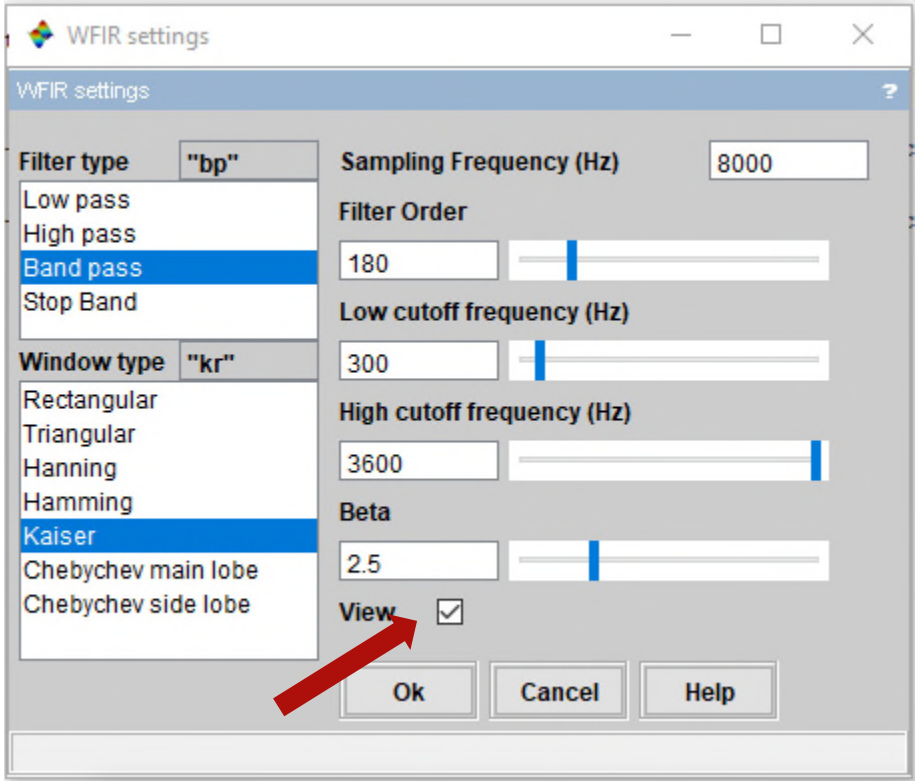
$f_s = 8kHz$





rpen via wfir in console

pe, filter-order, [fg1 fg2], windowtype, [par1 par2])



Snel en eenvoudig een FIR-filter ontwerpen via wfir in console

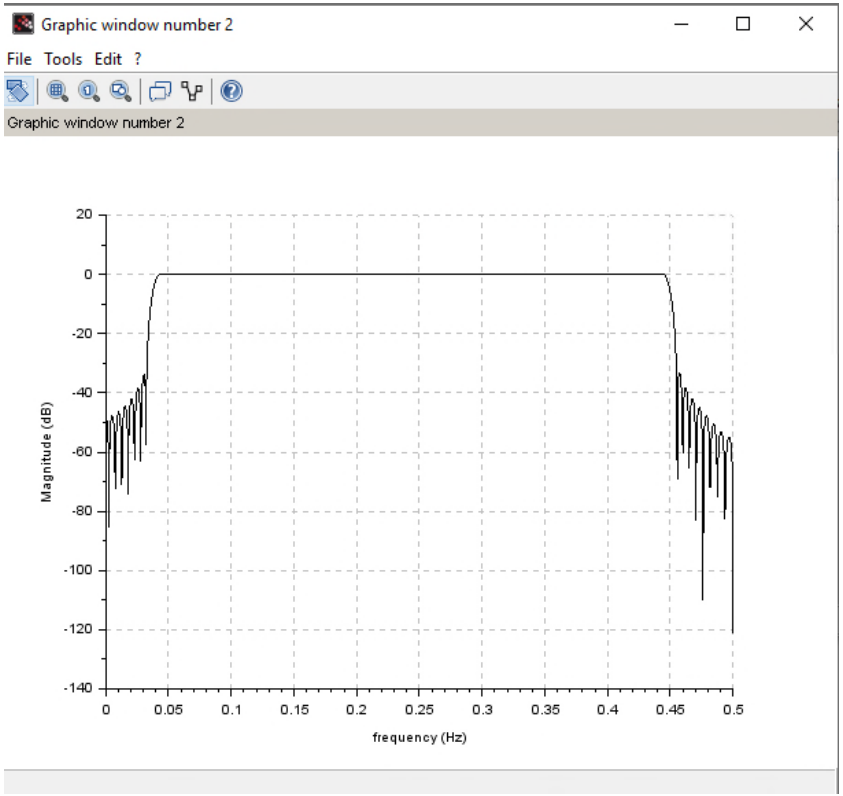
[coefficients, amplitude, frequency] = wfir(filter-type, filter-order, [fg1 fg2], windowtype, [par1 par2])

Stel we wensen een banddoorlaatfilter voor spraak te bekomen (300 Hz – 3300 Hz)

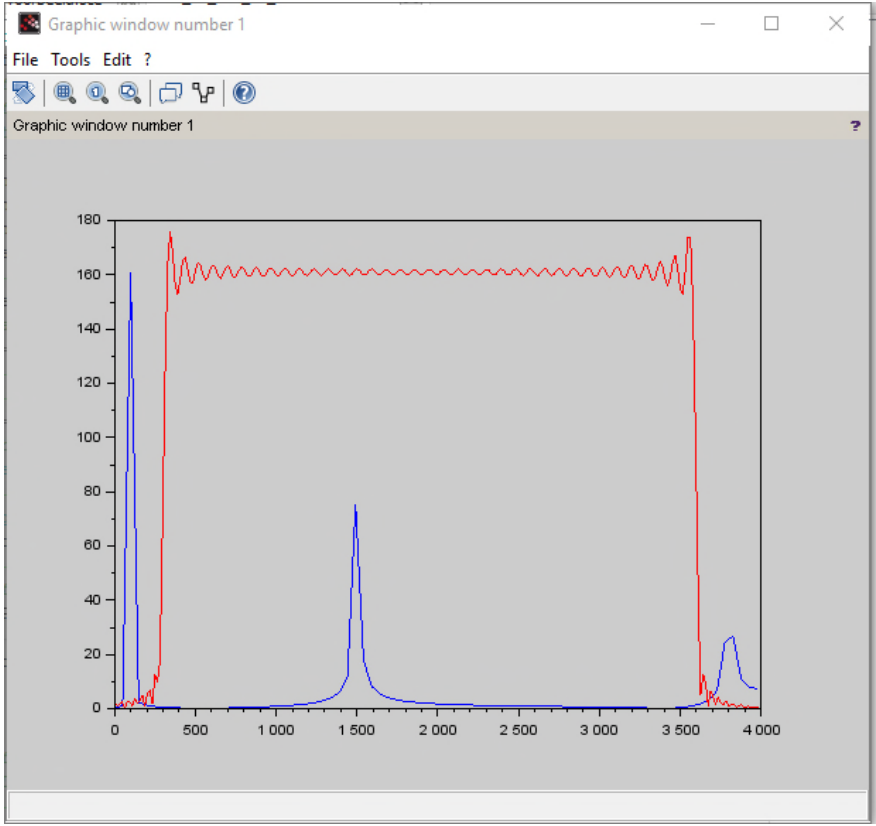
De frequenties 100 Hz en 3600 Hz moeten minstens 30 dB verzwakt worden

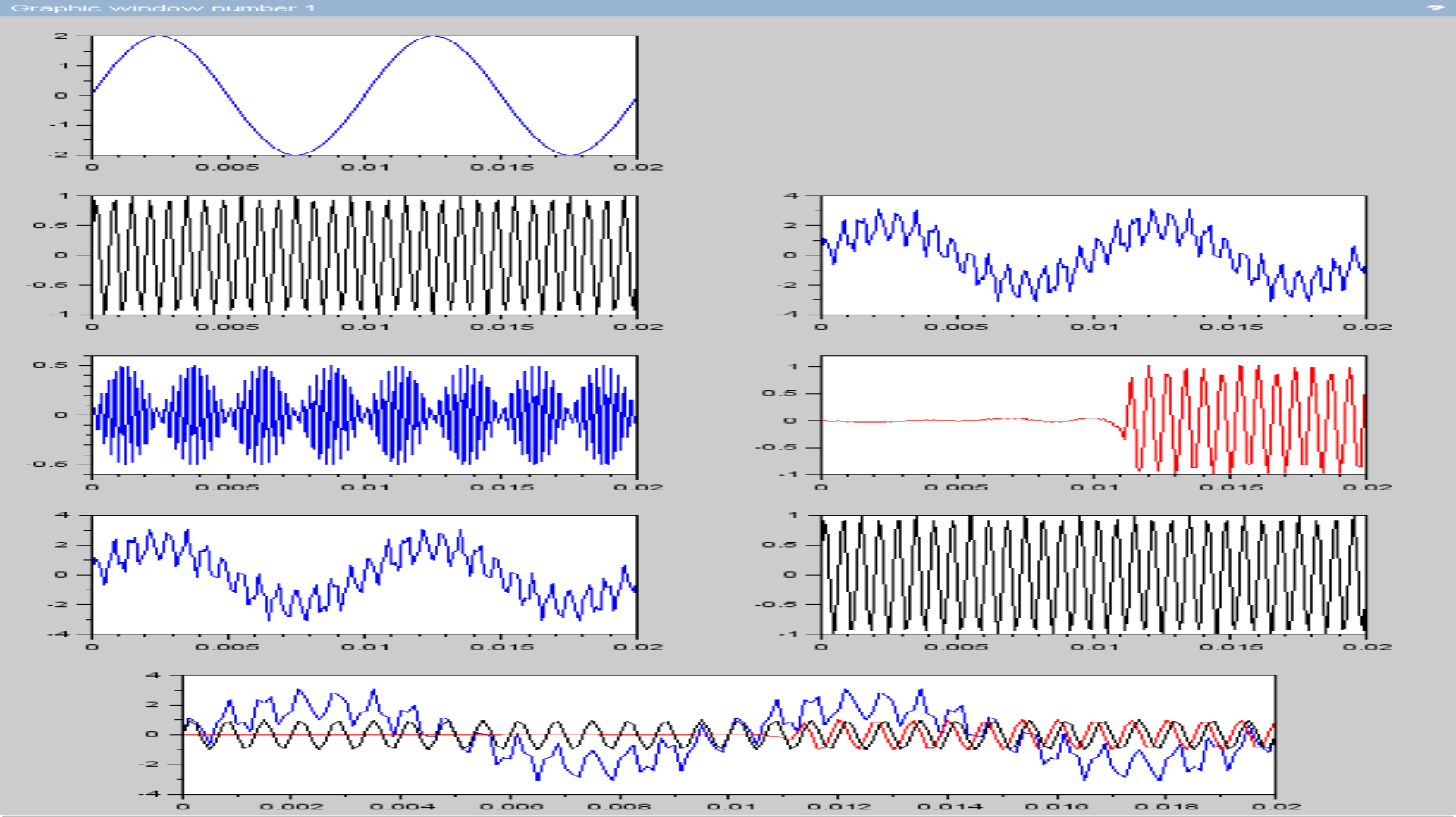
$f_s = 8kHz$ => normalisatie:

$$f_s = 1;$$
$$\mathrm{f_{cl}} = \frac{300}{8000} = 0.0375$$
$$\mathrm{f_{ch}} = \frac{3§00}{8000} = 0.45$$



```
//filteren testsignaal met FIR
//fc filter is 300 Hz en 3600 Hz
//testsignaal
t = [0:0.000125:0.02];
sin_100Hz = 2*sin(2*%pi*100*t);
sin_1500Hz = 1*sin(2*%pi*1500*t);
sin_3800Hz = 0.5*sin(2*%pi*3800*t);
testsign = sin_100Hz + sin_1500Hz + sin_3800Hz
// weergeven van frequentiespectrum
Frequentie_FFT = (abs(fft(testsign)))
//aantal samples
N = size(t, '*');
// fs = 8000 Hz
// geassocieerde frequentievector (tot nyquist)
f = 8000*(0:(N/2))/N;
//bepalen lengte f-vector
n=size(f, '*');
// weergeven figuur
figure;
plot (f, Frequentie_FFT(1:n));
// gebruik maken van de frequentie en amplitudewaarde
//filterontwerp bp orde 180 fc 300 en 3600 Hz, fs 8 kHz BW 3300 Hz
[BP_coeff, amplitude, frequentie] = ...
   wfir('bp', 180, [0.0375 0.45], 'kr', [0 0]);
plot (frequentie*8000, amplitude*N,'r')
```




Graphic window number 1

Einde

Frequentieselectieve filters

AP HOGESCHOOL ANTWERPEN

AP.BE