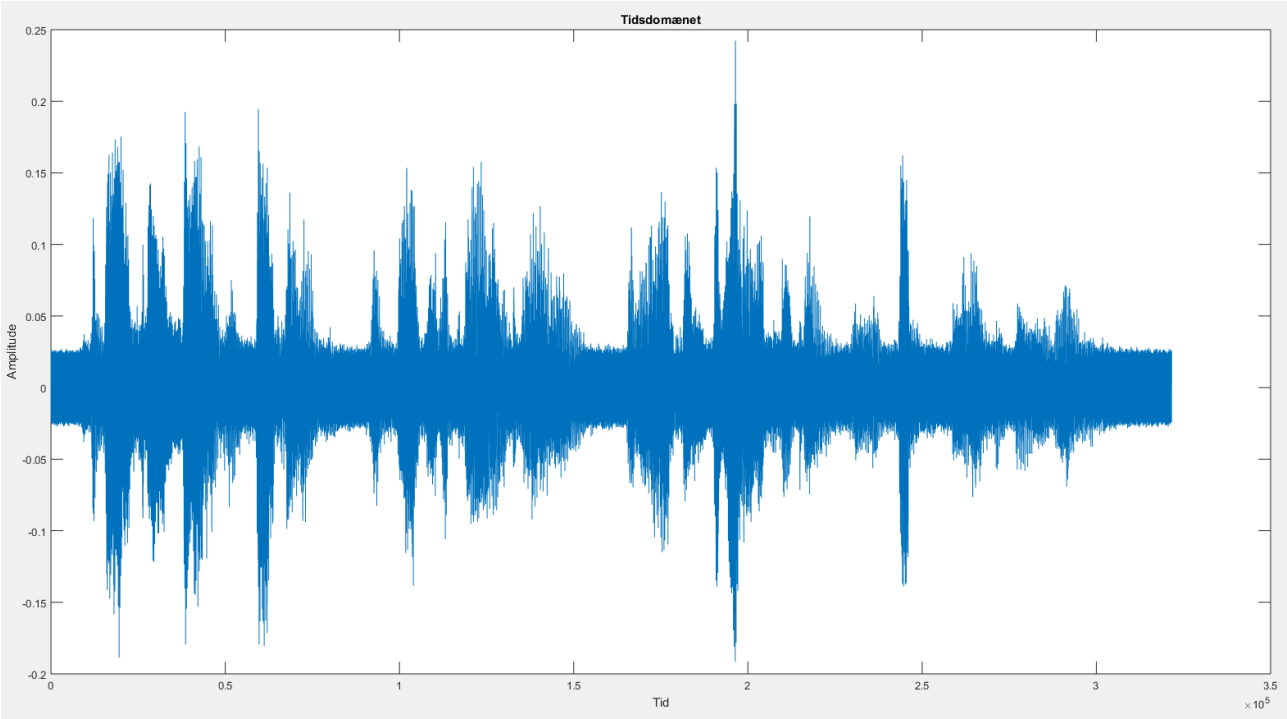
# Case Project 2 – Audio filter


| Navn | Studienummer |
|---|---|
| Naiyun Wu | 201405716 |
| Frederik Gadegaard | 201405625 |
| Maciej Lech | 20107672 |
| Jacob Aagaard | 201404442 |

# Indhold

# Opgave 1 – Analyse af inputsignal

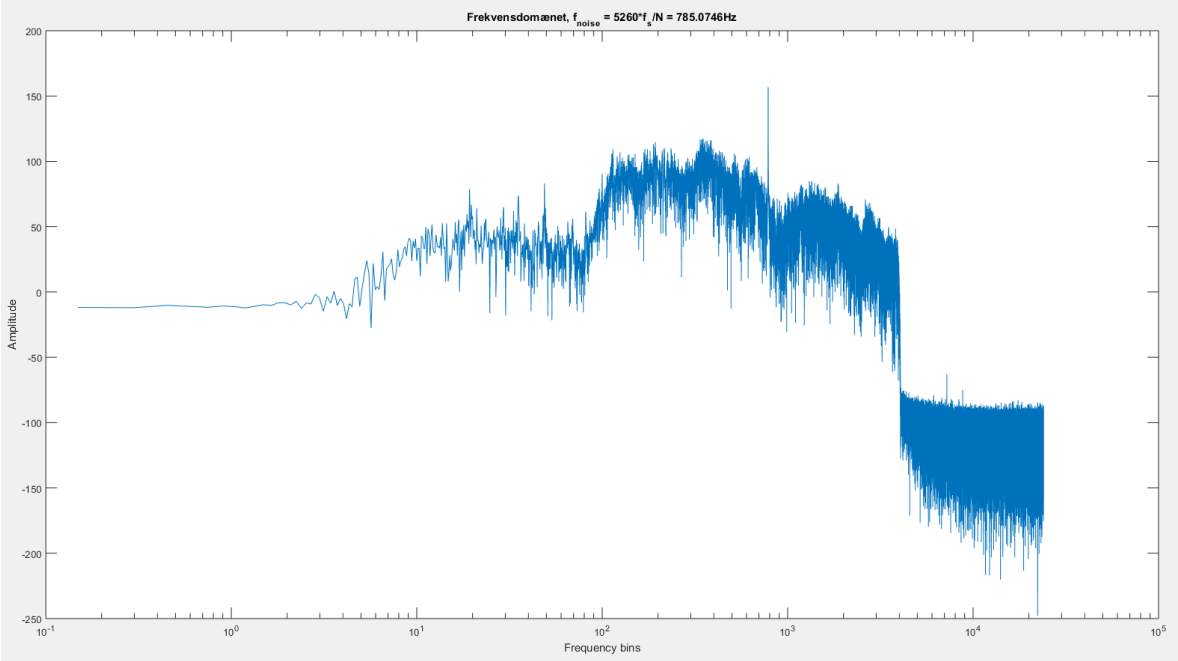
På Figur 1 ses signalet i tidsdomænet og det er tydeligt at se den forstyrrende tone hele tiden ligger i støjgulv neden under talesignalet.




*Figur 1 tale_tone i tidsdomænet*

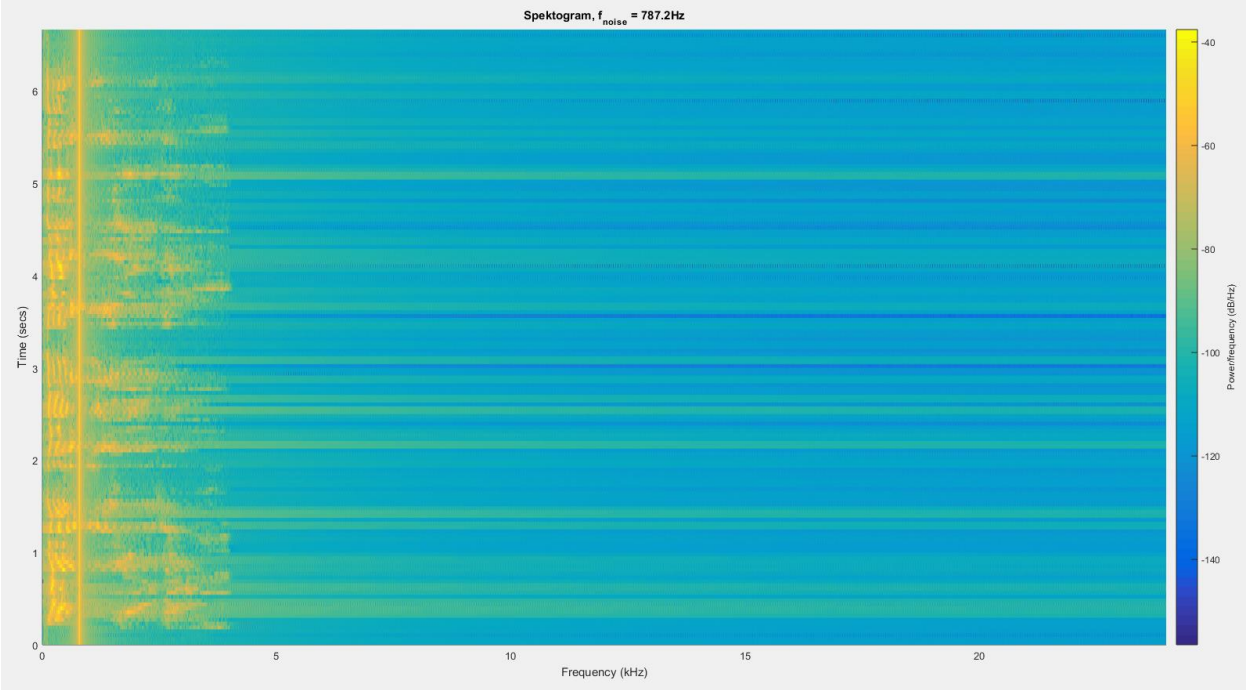
Når vi bevæger os over i frekvensdomænet, på Figur 2, ses den tydelige frekvens af støjen, der ligger ved bin nummer 5260, og svarer til frekvensen 785Hz, jf. nedenstående udregning:

$$f_{noise} = n_{bin} \cdot \frac{f_s}{N} = 5260 \cdot \frac{48000}{321600} = 785{,}0746\ Hz$$




*Figur 2 tale_tone i frekvensdomænet*

På Figur 3, spektogrammet, kan frekvensen bestemmes der hvor intensiteten er mest gul (og dermed, den frekvens, der har den højeste forstærkning). Det ses også tydeligt at det er en konstant tone, der ligger som baggrundsstøj igennem hele talesignalet - meget lig den oplevelse man får når man afspiller lydsignalet. Med den specifikke opløsning af spektogrammet, bestemmes frekvensen til $f_{noise} = 787{,}2Hz$. Meget tæt på frekvensen fundet i Figur 2.



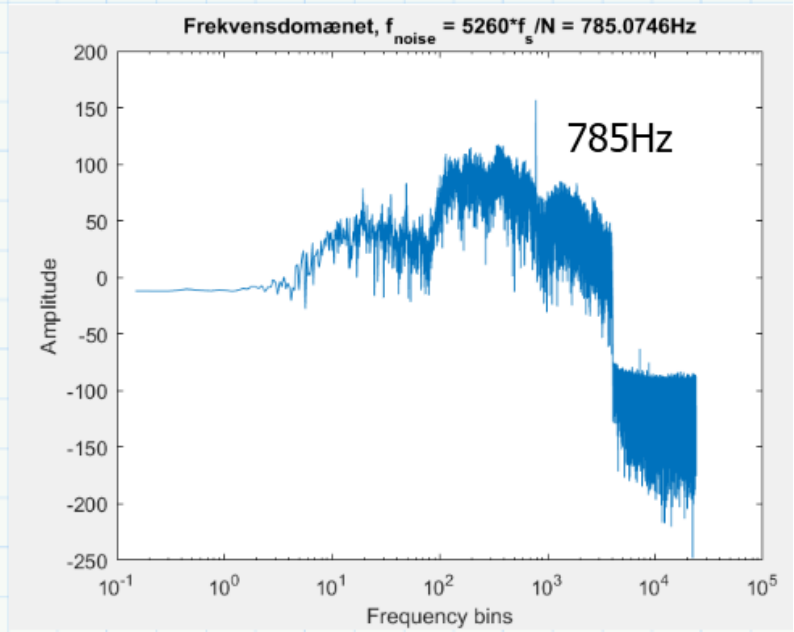

*Figur 3 tale_tone i spectrogram*

# Opgave 2 – IIR-notch filter design

## 1. Skitsér pol-nul punkts diagram for filteret

Herunder i Figur 4 har vi i matcad beregnet os frem til den nomeret vinkelfrekvens som vores pol og nul punkt skal have i pol-nul punkts diagrammet i Figur 5.

$fs := 48\ kHz$

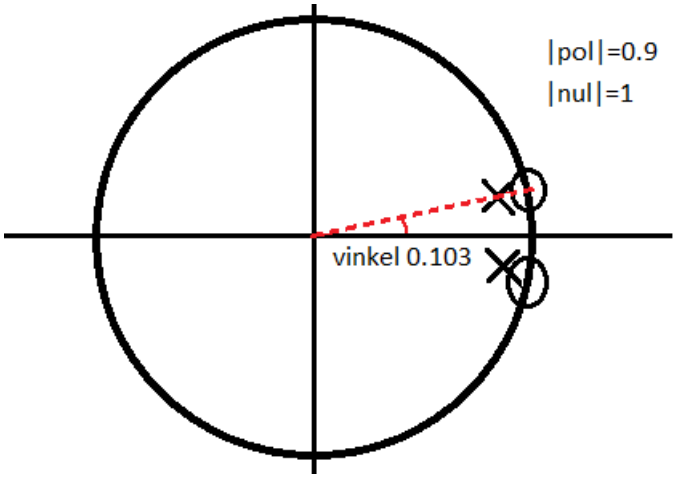



$peak := 785\ Hz$

$$\theta := \frac{peak}{fs} \cdot 360 = 5.888$$

$$\phi := \frac{peak}{fs} \cdot 2\ \pi = 0.103$$

*Figur 4 udregning af den nomeret vinkelfrekvens i pol-nul punkts diagram*




*Figur 5 Illustration af vores valgte poler og nul-punkter i et pol-nul punkts diagram*

## 2. Udregn systemfunktion H(z) og opskriv differensligningen herudfra

### Systemfunktion H(z):

$$nul := e^{1i \cdot 0.103} = 0.995 + 0.103i$$

$$|nul| = 1$$

$$\arg(nul) = 0.103$$

$$pol := 0.9 \cdot e^{1i \cdot 0.103} = 0.895 + 0.093i$$

$$|pol| = 0.9$$

$$\arg(pol) = 0.103$$

*Figur 6 Udregning af poler og nul punkter på sumform*

Herunder ses udregningen af overføringsfunktionen ud fra poler og nul punkter. Denne er implementeret i linje 13 i Figur 7.

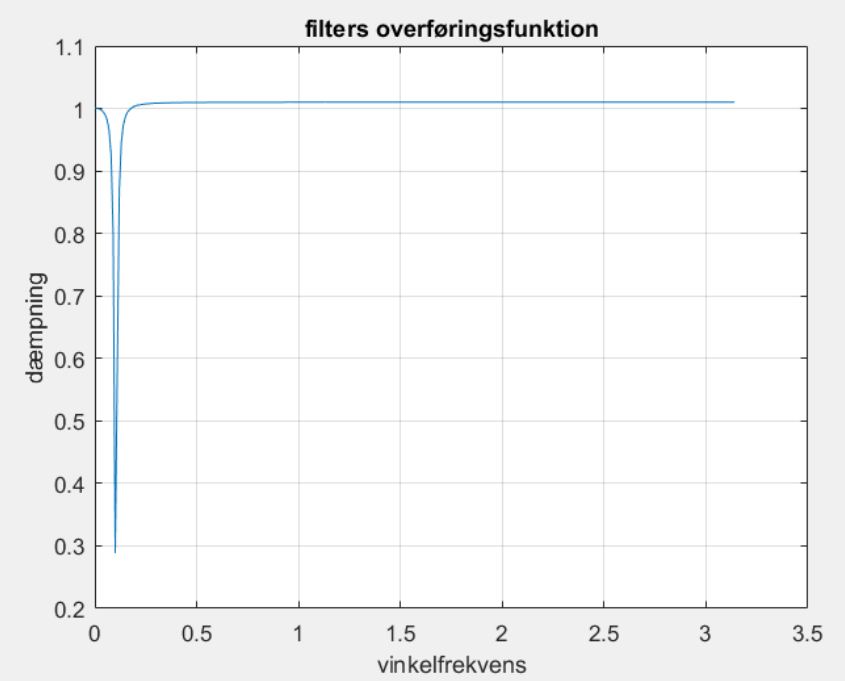
$$H(z) = \frac{Y(z)}{X(z)} = \frac{(z + z0) \cdot (z - z0)}{(z + p0) \cdot (z - p0)} = \frac{\left(z - (0.995 - 0.103i)\right) * \left(z - (0.995 + 0.103i)\right)}{\left(z - (0.895 - 0.093i)\right) * \left(z - (0.895 + 0.093i)\right)}$$

Kodeudsnittet herunder viser hvordan vi i linje 10 har oprettet en vektor gående fra 0 til pi svarende til 0 til fs/2. Dette gøres for ikke at kigge på det spejlede billede også. Derefter defineres $z = e^{1j \cdot w}$ og overføringsfunktionen H plottes herefter i linje 16 som kan ses i Figur 8.

```
1 -    clear;
2 -    clc;
3
4 -    s=tf('s');          %transfer function s
5
6 -    z0=1*exp(1i*0.103); %nul punkt
7
8 -    zp=0.99*exp(1i*0.103);%pol
9
10 -   w=[0:0.01:pi];      %make x_axis up to pi
11 -   z=exp(1i*w);        %z domain
12
13 -   H=((z-z0)).*(z-conj(z0))./((z-zp).*(z-conj(zp)));   %transfer function
14
15 -   figure
16 -   plot(w,abs(H))      %plotting transfer function
17 -   title('filters overføringsfunktion')
18 -   xlabel('vinkelfrekvens')
19 -   ylabel('dæmpning')
20 -   grid on
```

*Figur 7 Kodeudsnit af beregningen af overføringsfunktionen*

I Figur 8 ses vores IIR-notch filter som dæmper frekvenser ved $f = \frac{0.103 \cdot fs}{2 \cdot \pi} \rightarrow 785\ Hz = \frac{0.103\ \frac{rad}{s} \cdot 48\ kHz}{2 \cdot \pi}$.




*Figur 8 Overføringsfunktion H(z) plottet i linje 16 i Figur 7*

## Differensligning:

Herunder gennemgås udledningen af differensligningen ud fra overføringsfunktionen H(z):

$$H(z)=\frac{Y(z)}{X(z)}=\frac{-1.99\cdot z+z^2+1.0}{-1.79\cdot z+z^2+0.81}=\frac{-1.99\cdot z^{-1}+1+1\ z^{-2}}{-1.79\cdot z^{-1}+1+0.81\ z^{-2}}=\frac{z^{-2}-1.99\cdot z^{-1}+1}{0.81\ z^{-2}-1.79\cdot z^{-1}+1}$$

*Figur 9 H(z) overføringsfunktion*

$$Y(z)\cdot\left(0.81\ z^{-2}-1.79\cdot z^{-1}+1\right)=X(z)\cdot\left(z^{-2}-1.99\cdot z^{-1}+1\right)$$

*Figur 10 X(z) er ganget på begge sider af lighedstegnet*

$$Y(z)=X(z)+X(z)\cdot z^{-2}-0.81\cdot Y(z)\cdot z^{-2}+1.79\cdot Y(z)\cdot z^{-1}-1.99\cdot X(z)\cdot z^{-1}$$

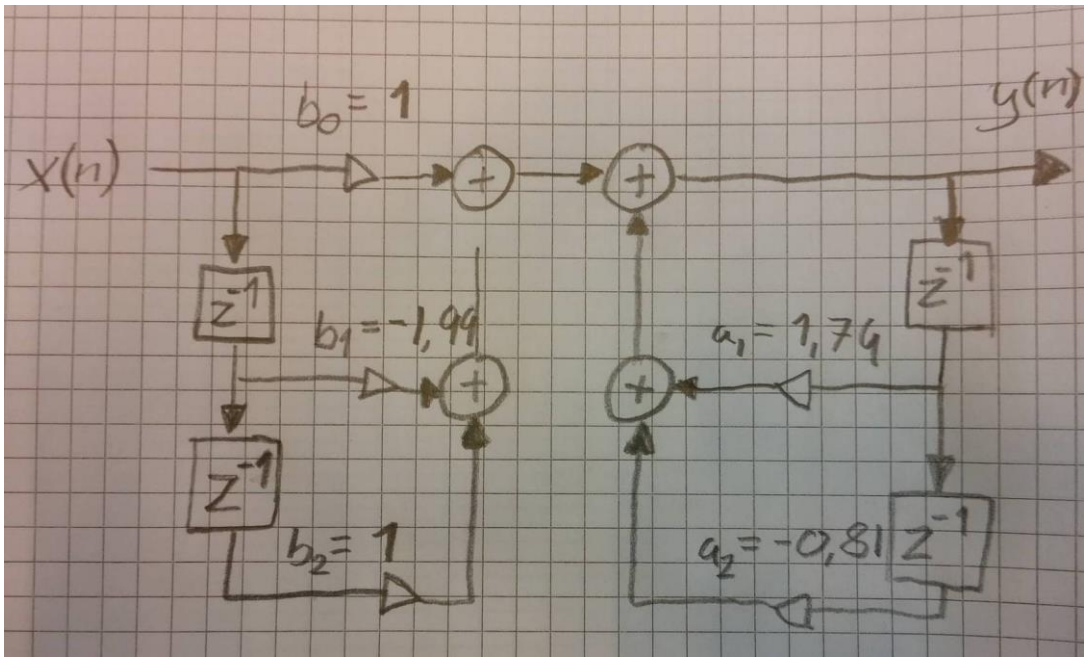
*Figur 11 Y(z) og X(z) er ganget ind i parentesen og Y(z) er isoleret*

$$y(n)=x(n)-1.99\cdot x(n-1)+x(n-2)+1.79\cdot y(n-1)-0.81\cdot y(n-2)$$

*Figur 12 Endelige differensligning*

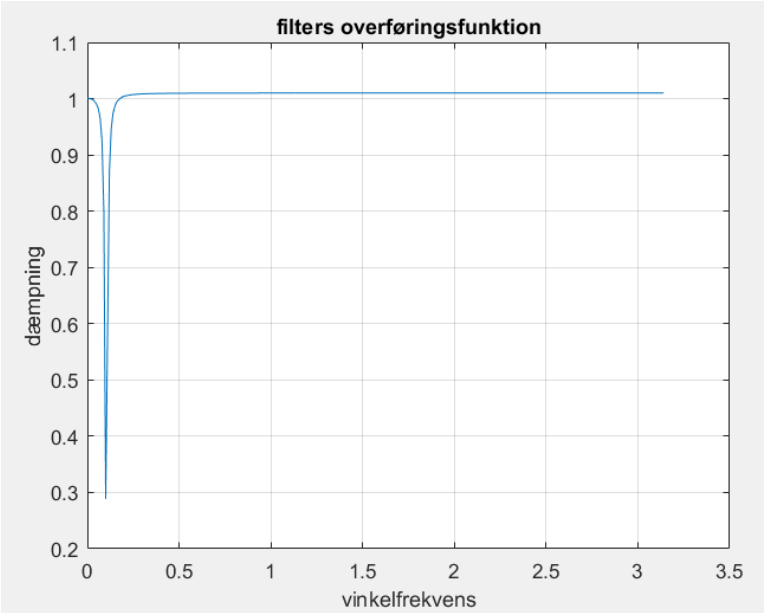
## 3. Tegn signalgrafen for filteret på direkte form I.

Ved brug af filterkoefficienterne kan vi opstille signalgrafen på direkte form I, som ses på Figur 13. Den giver et overblik over selve udregningen i z-domænet fra input til output. Det tydeliggøre de forskellige delays og hvilken tilhørende koefficient som bliver ganget på.




*Figur 13 signalgraf skitseret på direkt form I*

## 4. Bestem og plot frekvens-responset $|H(ej\omega)|$ for filteret udfra H(z) – Er det et ideelt filter?




*Figur 14 Overføringsfunktion H(z)*

Det er ikke en ideelt filter. Et ideelt filter vil have en uendelig orden og lodret knæk ved bandstop frekvensen. Dette ville vi aldrig kunne opnå i praksis på et digitalt filter da vi ikke kan have en uendelig orden.

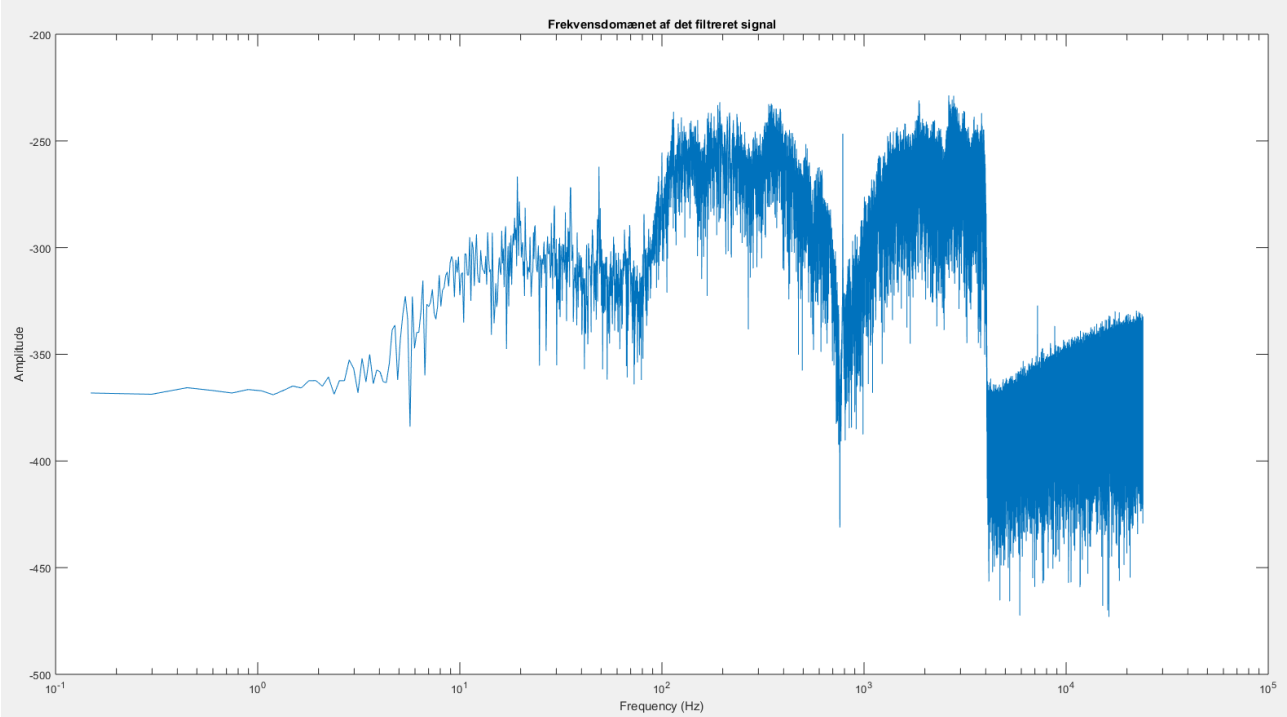
## 5. Test at filteret virker ved at filtrere lydfilen i Matlab

Herunder i Figur 15 ses koden for testen af filteret. Vi anvender den indbyggede filter – funktion i matlab til at konstruere filteret ud fra filterkoefficienterne a og b. Vi lyttede til det filtreret lydsignal og kunne verificere at den støjende tone var filtreret væk.

```
32      % load signalet ind i x-variable
33 -    [x, fs] = audioread('tale_tone_48000.wav');
34
35      % a- og b-koeffienter
36 -    a = [1.79 -0.81];
37 -    b = [1 -1.99 1];
38
39      %filtrere signal x
40 -    y = filter(b,a,x);
41
42      %test af filtreret signal y
43 -    soundsc(y,fs)
```

*Figur 15 kodeudsnit af test af det filtreret lydsignal*

Resultat af filterets påvirkning på signalet ses i Figur 16 herunder. Som det ses i forhold til frekvens-analysen i ##, er tonen på de 785 Hz dæmpet betydeligt. Her ses det også at vores filter ikke er ideelt og at det dæmper nærtliggende frekvenser en smule også.




*Figur 16 FIlteret lydsignal i frekvensdomænet*

## Opgave 3 – Implementering på blackfin processor

Nedenunder ses koden for vores implementeret filter i funktionen myVolume.

```
1   #include "Talkthrough.h"
2   // Modify and insert your notch filter here!!!!
3
4   short xn=0, x1=0, x2=0;
5   short y1=0, y2=0;
6   short out;
7   int tmp_out;
8
9   short myVolume(short x)
10  {
11  static short b0 = 8192;         // Values from MATLAB, times 2^13
12  static short b1 = -16302;
13  static short b2 = 8192;
14  static short a1 = 14664;
15  static short a2 = -6636;
16
17  //Differensligning
18  tmp_out = (int) b0*x + (int) b1*x1 + (int) b2*x2 + (int) a2*y2 + (int) a1*y1;
19
20  out = tmp_out >> 13;            // Divide output by 2^13
21  y2 = y1;                        // y(n-2)
22  y1 = out;                       // y(n-1)
23
24  x2 = x1;                        // x(n-2)
25  x1 = x;                         // x(n-1)
26
27  return out;
28  }
```

For at implementere IIR notch filteret på Blackfin processoren, er det nødvendigt at overveje type konvertering nøje. Da det ikke er muligt for os at arbejde med floating-point variable, har vi på forhånd skaleret filterets koefficienter med en faktor 2^13. Disse værdier oprettes som short variable, da 16-bit er rigeligt til at opbevare disse i.

Differensligningens output lagres midlertidigt i en integer i linje 18, og selve beregningerne typecastes også til integers, for at sikre at der ikke sker overflow, når to short variable ganges sammen. Denne værdi shiftes derefter 13 pladser til højre, i linje 20, og lægges ind i variablen `out` af typen `short`. Derefter lægges den sidste output-værdi `y1` ind i `y2` for at implementere delay-funktionaliteten, y(n-2) og out lægges tilsvarende i `y1`, for at implementere y(n-1). Det samme gælder i linje 24 og 25 for delay af x-variablene.

Til slut returneres `out` variablen.

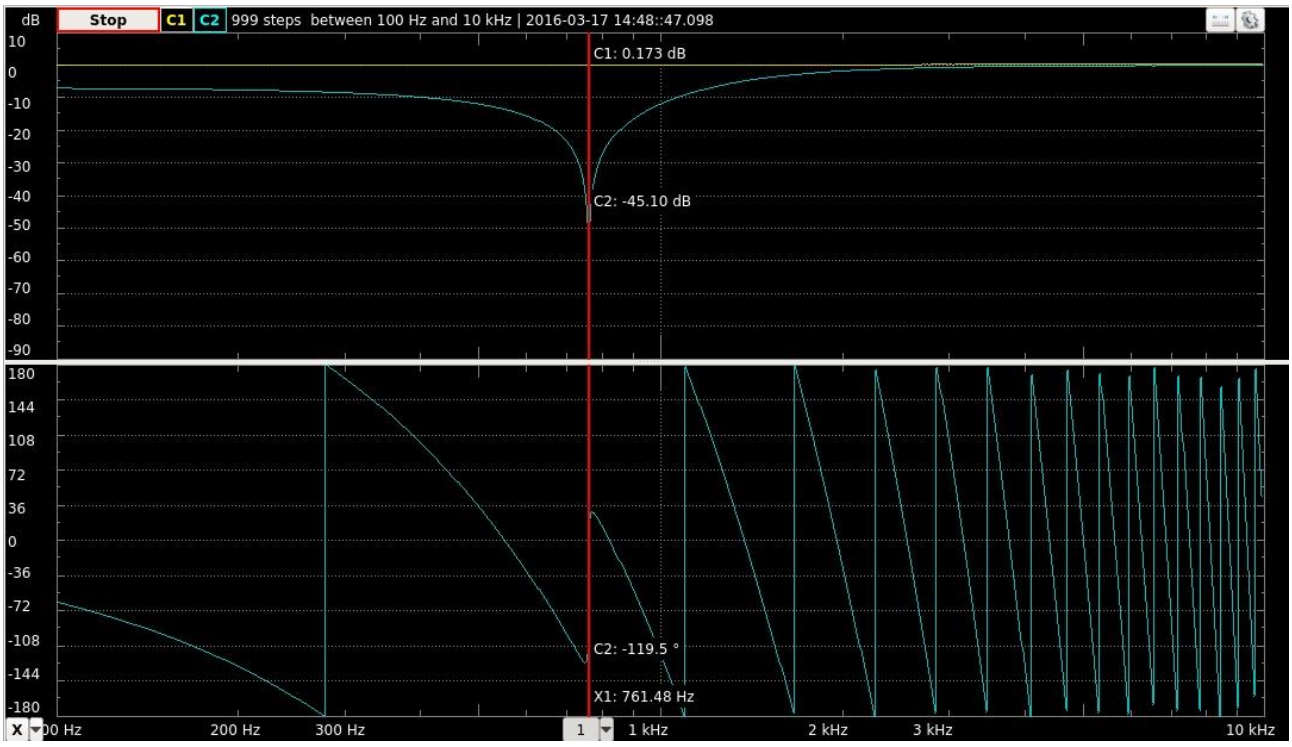
Undervejs havde vi problemer med at vi hørte støj når alle koefficienter indgik i differensligningen, men vi prøvede at fjerne y-leddene, for at se effekten af et FIR notch filter. Det resulterede i at hele lydsignalet blev dæmpet, og at noget af tonen blev fjernet. Dernæst måtte vi debugge koden, for at finde ud af hvor problemet lå med y-leddene. Det viste sig at vi havde implementeret delay-funktionaliteten i forkert

rækkefølge, hvilket resulterede i at outputtet fra differensligningen voksede konstant, altså havde vi fået lavet et ustabilt filter.

## Opgave 4 – Test af filteret

Vi lyttede til output-signalet direkte fra blackfin processoren og kunne verificere at den støjende tone var dæmpet væsentligt og næsten ikke længere hørbar.

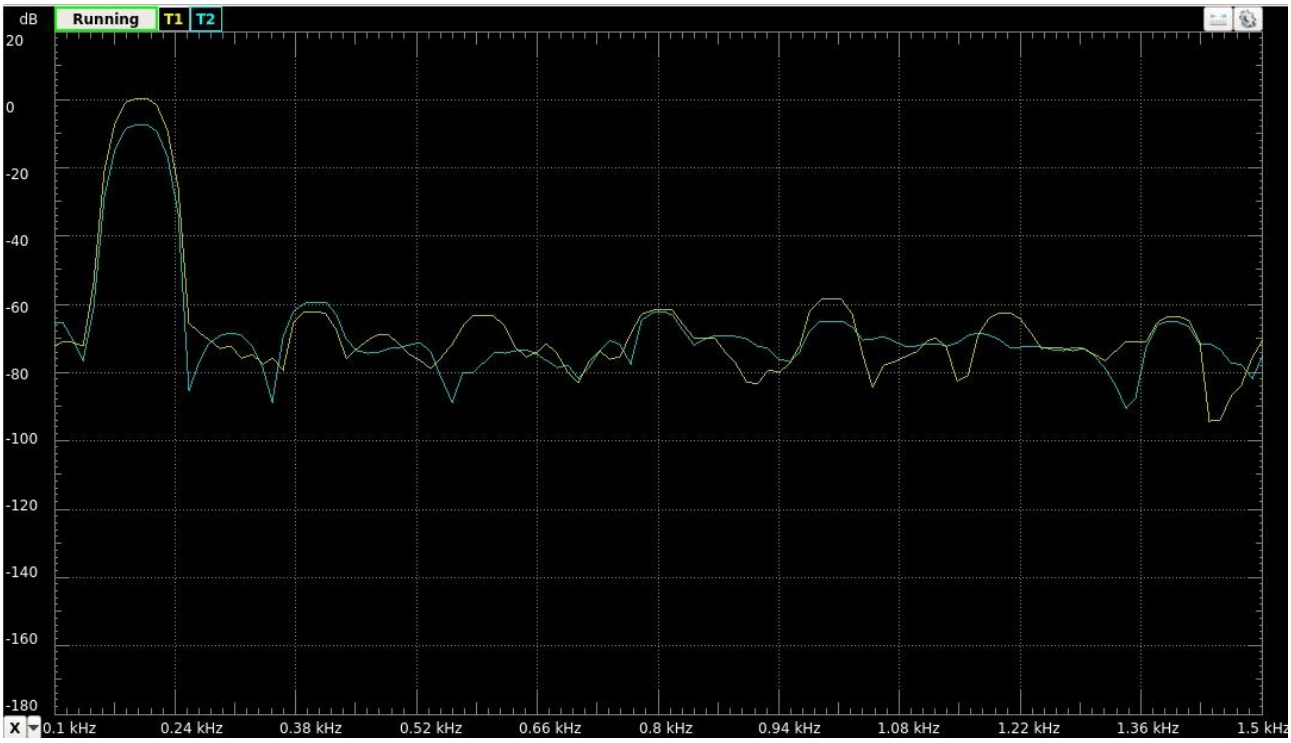
Her i Figur 17 ses en netværks analyse af det realiseret notch filter på blackfin processoren. Vi får en dæmpning på ca. 45 dB omkring det forventede område. Peaken er aflæst til 761 Hz hvor det teoretisk skulle være 785 Hz. Dette kan skyldes afrunding af filterkoefficienter eller usikkerhed i aflæsning på grafen.



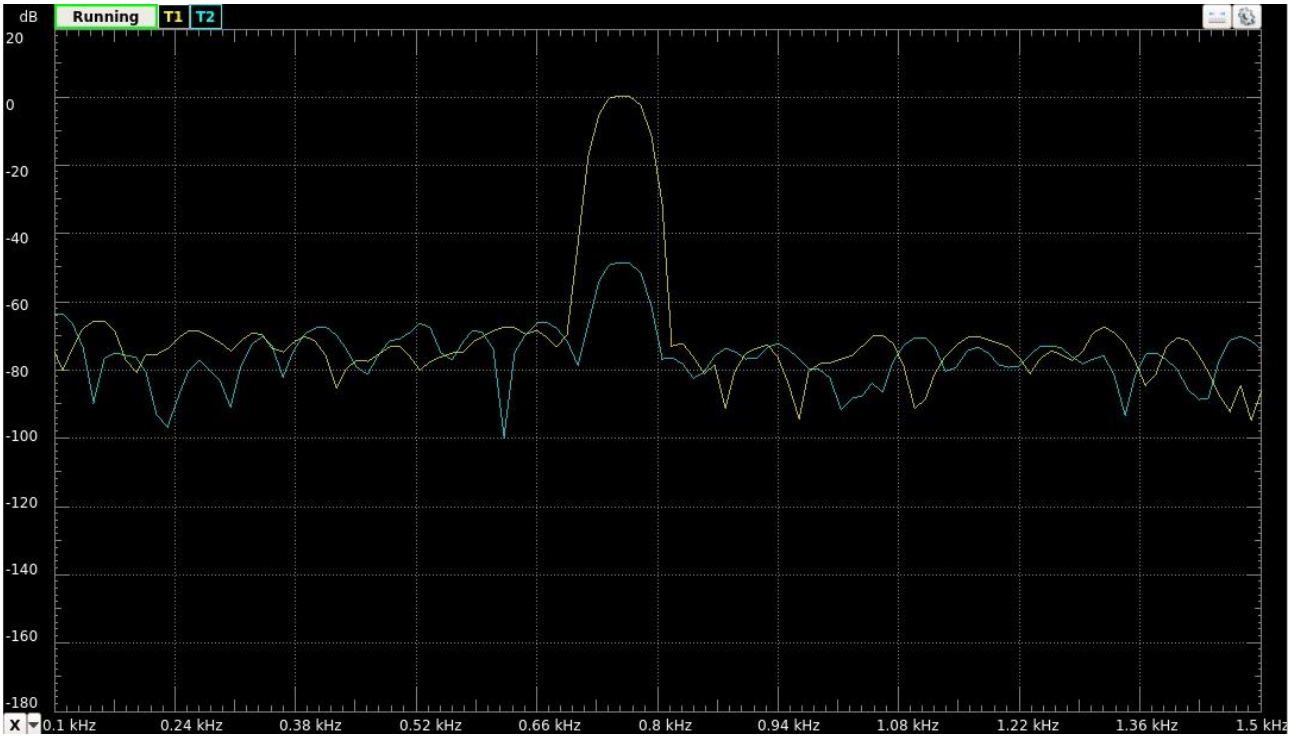

*Figur 17 Frekvens analyse filteret via Analog Discovery (Network Analyzer)*

For at sammenligne med input-signalet har vi lavet en spectrum-analyse også, for at se forholdet i mellem input og output ved to forskellige frekvenser. Begge grafer er normeret efter input-signalet.

I Figur 18 havde input-signalet en frekvens på 200 Hz og i Figur 19 nedenunder havde input signalet en frekvens på 785 Hz. Som forventede observeres en lille dæmpning ved de 200 Hz på ca. 7 dB og en meget større dæmpning ved de 785 Hz på ca. 45 dB. Dette bekræfter IIR-notch filterets virkning.




*Figur 18 Inputsignal = 200 Hz*




*Figur 19 Inputsignal = 785 Hz*