|  |
| --- |
| E4DSA – Introduktion til Digital Signal Analyse |
| Case 1 - Vejecelle |
| Reducering af støj på vejecelle-data |

|  |
| --- |
| Gruppe 3  Morten Haahr Kristensen – 201807664  Mojtaba Turabi – 201500178  Peer Weidemann – 201807779 |
| 19.2.2021 |

Indholdsfortegnelse

[Formål 3](#_Toc64652680)

[Opgave 1 – Data analyse (Morten) 3](#_Toc64652681)

[Indledning 3](#_Toc64652682)

[Opgave 1a (Morten) 3](#_Toc64652683)

[Teori 5](#_Toc64652684)

[Implementering 6](#_Toc64652685)

[Resultat 6](#_Toc64652686)

[Konklusion 6](#_Toc64652687)

[Opgave 1b (Morten) 6](#_Toc64652688)

[Teori 6](#_Toc64652689)

[Implementering 7](#_Toc64652690)

[Resultat 8](#_Toc64652691)

[Konklusion 8](#_Toc64652692)

[Opgave 1c (Morten) 8](#_Toc64652693)

[Teori 8](#_Toc64652694)

[Implementering 9](#_Toc64652695)

[Resultat 10](#_Toc64652696)

[Konklusion 10](#_Toc64652697)

[Opgave 1d (Morten) 11](#_Toc64652698)

[Teori 11](#_Toc64652699)

[Implementering 11](#_Toc64652700)

[Resultat 11](#_Toc64652701)

[Konklusion 11](#_Toc64652702)

[Opgave 2 – Implementering af filtre og analyse af filtrerede data (Peer & Mojtaba) 12](#_Toc64652703)

[Opgave 2.a (Peer) 12](#_Toc64652704)

[Teori 12](#_Toc64652705)

[Implementering 12](#_Toc64652706)

[Resultat 13](#_Toc64652707)

[Konklusion 14](#_Toc64652708)

[Opgave 2.b (Peer) 14](#_Toc64652709)

[Resultater 14](#_Toc64652710)

[Konklusion 17](#_Toc64652711)

[Opgave 2.C (Peer) 17](#_Toc64652712)

[Teori 17](#_Toc64652713)

[Resultat 17](#_Toc64652714)

[Opgave 2.d (Mojtaba) 18](#_Toc64652715)

[Teori 18](#_Toc64652716)

[Implementering 18](#_Toc64652717)

[Resultat 19](#_Toc64652718)

[Konklusion 20](#_Toc64652719)

[Opgave 2.e (Mojtaba) 20](#_Toc64652720)

[Teori 20](#_Toc64652721)

[Implementering 20](#_Toc64652722)

[Resultat 21](#_Toc64652723)

[Konklusion 22](#_Toc64652724)

[Opgave 2.f (Mojtaba) 22](#_Toc64652725)

[Teori 22](#_Toc64652726)

[Implementering 22](#_Toc64652727)

[Resultat 23](#_Toc64652728)

[Konklusion 24](#_Toc64652729)

[Opgave 3 (Morten) 25](#_Toc64652730)

[Teori 25](#_Toc64652731)

[Implementering 25](#_Toc64652732)

[Resultat 25](#_Toc64652733)

[Konklusion 25](#_Toc64652734)

[Appendix 26](#_Toc64652735)

[References 26](#_Toc64652736)

[Matlab Kode Implementering 26](#_Toc64652737)

[Main Kode 26](#_Toc64652738)

[Rekursivt Midlingsfilter (Fra Øvelse 1) 33](#_Toc64652739)

[Eksponentielt Midlingsfilter (Fra Øvelse 2) 33](#_Toc64652740)

# Formål

I denne Case skal der ses på forskellige typer af midlingsfiltre, hvilke egenskaber de har og hvordan de implementeres. Disse filtre vil anvendes på måledata optaget af en vægt, hvorpå der ligger støj. Arbejdsfordelingen kan ses for hver enkelt opgave, idet der bag hver opgave overskrift står navnet af den person, der har lavet opgaven.

# Opgave 1 – Data analyse (Morten)

## Indledning

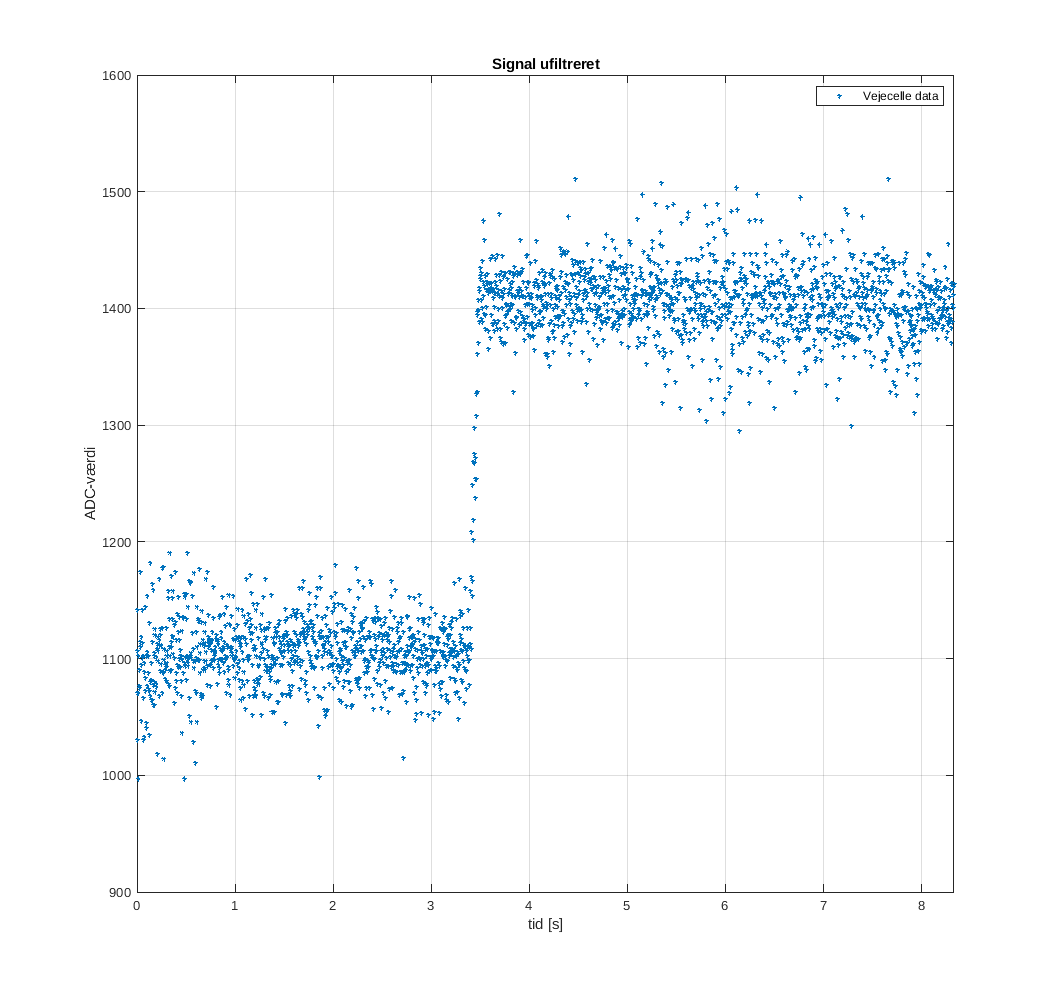
Til case 1 bliver der givet en datafil “vejecelle\_data.mat”, som vil blive analyseret i denne opgave. Datafilen indeholder data fra en vejecelle, hvor der først er blevet samplet uden et lod på vejecellen og derefter med et lod på 1 kg. Som det ofte er med signaler fra den virkelige verden, indeholder dataene fra vejecellen støj. Det er primært støjen, som vil blive analyseret i denne opgave, hvortil målet i de kommende opgaver er at minimere støjen gennem forskellige filtre.

## Opgave 1a (Morten)

Denne delopgave består af at finde middelværdien, spredningen og variansen af signalet. Før der kigges på disse statistiske egenskaber af signalet, giver det dog godt mening at tegne hele signalet.

load('vejecelle\_data.mat');  
N = length(vejecelle\_data);  
t = [0:1/fs:N/fs-1/fs]; % tidsakse   
   
figure(10);  
plot(t,vejecelle\_data, '\*', 'MarkerSize', 3)  
title('Signal ufiltreret')  
ylabel('ADC-værdi')  
xlabel('tid [s]')  
legend('Vejecelle data');  
xlim([0 t(**end**)]);  
grid on;

Ovenfor ses hvordan datafilen først er loadet ind i Matlab og data-punkterne derefter er tegnet. Der er blevet lavet en tidsakse for at kunne lave x-aksen om til tid i sekunder i stedet for samples. Dataet er tegnet i diskret tid.



Figur 1 - Vejecelle data tegnet ufiltreret i tidsdomænet

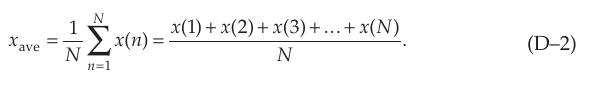
På Figur 1 ses det, hvordan dataet fra vejecellen er fordelt. Det ses at der er en del støj på signalet, men at det ligner et step, hvor der forekommer en ændring omkring 3,3 sekunder inde. Det er her der bliver påført et lod.

For at løse opgaven giver det bedst mening at opdele signalet i to grupper. Den ene med det ubelastede signal og den anden med det belastede signal.

Dataet bliver derfor opdelt i ubelastet og belastet, som svarer til når vejecellen er ubelastet og belastet. Det ubelastede signal er defineret fra sample 0...999 og det belastede signal fra signal N-999...N, hvor N er længden af det samlede signal (numerisk værdi for N = 2500).

### Teori

En estimering af middelværdien af det ubelastede signal kan findes ved hjælp af formel D-2 fra bogen [1]:



Ligning 1 - Formel til estimering af middelværdi

Ligning 1 siger at gennemsnittet af en talsekvens i tidsdomænet kan findes ved summen af elementer divideret men antallet af elementer. Det er værd at notere at gennemsnittet i Ligning 1 er en estimering af middelværdien. For at finde den korrekte middelværdi, kræves det, at der haves uendeligt mange samples.

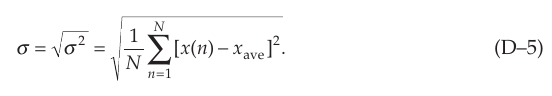
For at finde variansen, kan formel D-3 fra bogen [2] anvendes:



Ligning 2 - Formel til estimering af varians

Ligning 2 siger at variansen af en talsekvens kan findes ved summen af forskellen mellem de enkelte elementer og gennemsnittet i anden. Til sidst normaliseres med længden af talsekvensen. Ligesom ved ligning 2 er ligning 3 en estimering af variansen.

For at finde standardafvigelsen (spredningen), kan ligning 3 fra bogen [3] anvendes:



Ligning 3 - Formel til estimering af standardafvigelse

Ligning 3 siger at standardafvigelsen er kvadratroden af variansen. Da der variansen er en estimering, bliver standardafvigelsen også en estimering.

### Implementering

Middelværdien, gennemsnittet og variansen er blevet estimeret i Matlab gennem nedenstående kode. Bemærk at det i eksemplet kun er for det ubelastede signal, men implementeringen er gentaget for det belastede signal.

% Estimeret middelværdi:  
est\_middel\_ubelastet = 1/N\_ubelastet \* sum(signal\_ubelastet);  
% Estimeret varians:  
est\_var\_ubelastet = 1/(N\_ubelastet-1)\*sum((signal\_ubelastet-est\_middel\_ubelastet).^2);  
% Estimeret standardafvigelse:  
est\_std\_ubelastet = sqrt(est\_var\_ubelastet);

### Resultat

|  |  |  |
| --- | --- | --- |
|  | Ubelastet signal | Belastet signal |
| Middelværdi | 1106,4 | 1404,4 |
| Varians | 764,9 | 905,8 |
| Standardafvigelse | 27,7 | 30,1 |

Tabel 1 - Resultater fra delopgave 1a

Resultaterne fra delopgave 1a ses i Tabel 1.

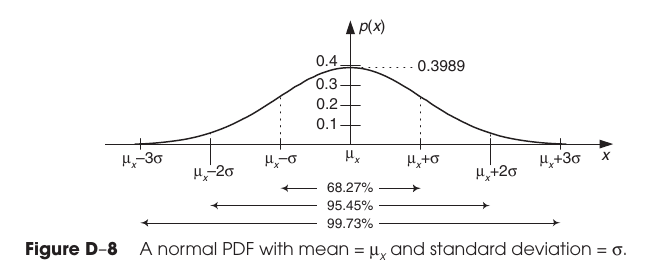
### Konklusion

Det kan konkluderes at de en estimering af middelværdien, variansen og standardafvigelsen blev udregnet. Resultaterne kan ses af Tabel 1.

## Opgave 1b (Morten)

Den næste delopgave handler om at plotte histogrammer af dataet og vurdere, hvorvidt dataet ser normalfordelt ud.

### Teori



Figur 2 - Normalfordelingskurve. På engelsk PDF (“probability density function”) [4]

Figur 2 viser normalfordelingskurven, der vil blive taget udgangspunkt i ved denne delopgave. På figur D-8 er kurven beskrevet gennem standardafvigelser, men den kan også beskrives gennem numeriske værdier. Normalfordelingskurven beskriver sandsynligheden for at et givent datapunkt ligger indenfor et interval. Eksempelvis ses det på figur 2, at der er 68,27 % sandsynlighed for at et datapunkt ligger indenfor middelværdien µ +/- standardafvigelsen.

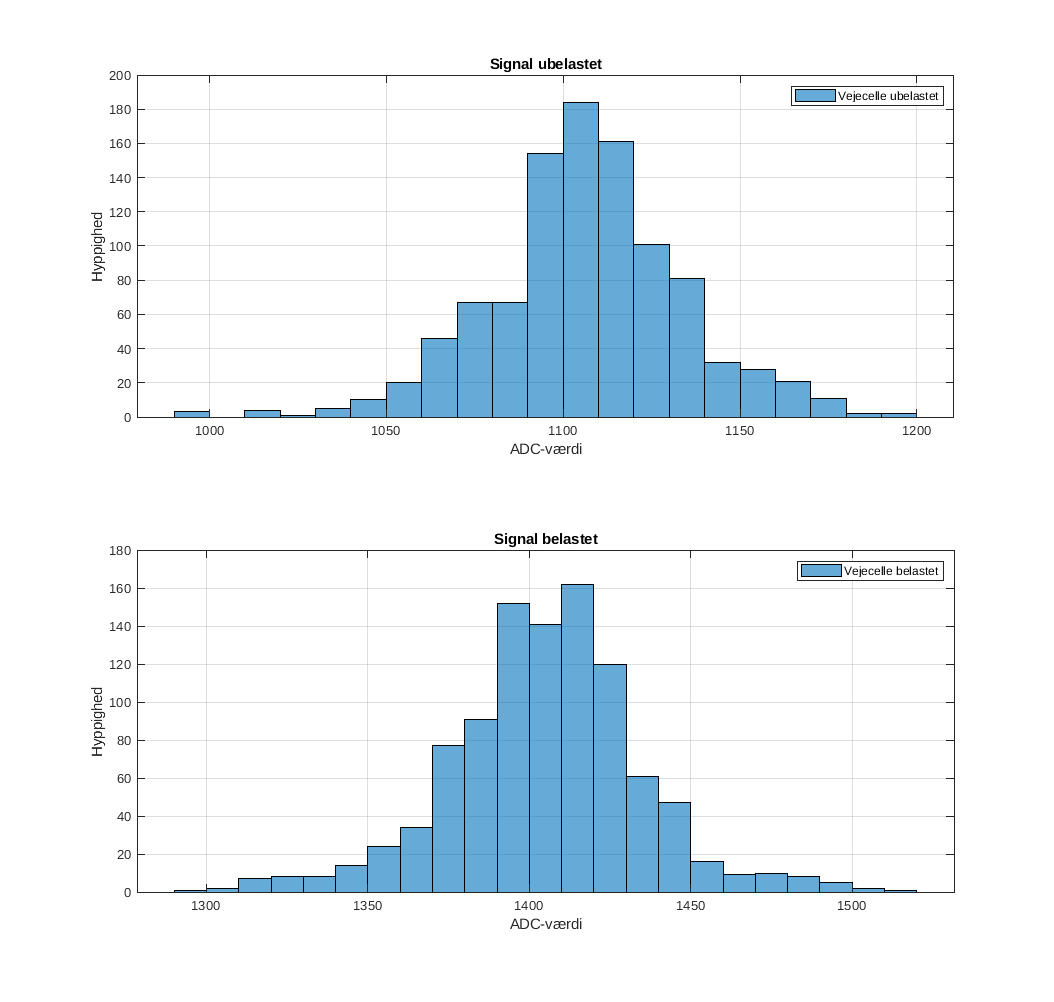
Hvis dataet fra vejecellen ser normalfordelt ud, svarer det til at de tegnede histogrammer har en form som minder om den på figur 2.

### Implementering

Dataet er plottet som histogrammer gennem kodeudsnippet nedenfor. I kodeudsnippet er det kun det ubelastede signal, som tegnes, men dette gentages for det belastede signal.

subplot(2,1,1);  
histogram(signal\_ubelastet);  
title('Signal ubelastet')  
ylabel('Hyppighed')  
xlabel('ADC-værdi')  
legend('Vejecelle ubelastet');  
grid on;

### Resultat



Figur 3 - Histogrammer over data for vejecellen. Belastet og ubelastet.

På Figur 3 ses de to histogrammer for vejecellen.

### Konklusion

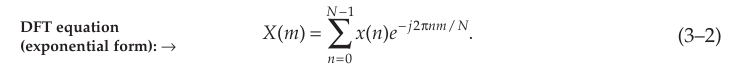
Ud fra figur 3 kan det konkluderes, at dataet ser ud til at være normalfordelt. Dette kan baseres på at dataets form i histogrammerne minder om klokkeformen på figur 2.

## Opgave 1c (Morten)

Den næste delopgave til opgave 1 går ud på at plottefrekvensspektret for signalet og vurdere, hvorvidt det ligner at der fremtræder hvidt støj.

### Teori

For at plotte frekvensspektret, skal dataen først konverteres til frekvensdomænet gennem en Fourier Transformation. I den idéelle verden kunne der anvendes en Diskret Tids Fourier Transformation (DTFT), hvor der samples over et input-signal i diskret tid, men resultatet er et kontinuert signal i frekvensdomænet. Dette kan dog ikke lade sig gøre, da det kræves at der samples fra tiden minus uendeligt til uendeligt. I stedet kan den Diskrete Fourier Transformation (DFT) anvendes. Formlen til DFT kan ses i Ligning 4 [5]. DTF tager et diskret signal fra tidsdomænet og transformerer det til et diskret signal i frekvensdomænet. Ulempen ved DTF frem for DTFT er, at man ikke kan se energien for en vilkårlig frekvens, men er bundet til de diskrete værdier i form af frekvens bins.



Ligning 4 - Den Diskrete Fourier Transformation i eksponentiel form.

Det ses af Ligning 4, at DFT resulterer i et signal i frekvensdomænet, som er en funktion af m, hvor m repræsenterer frekvens bins.

### Implementering

bin\_ubelastet\_res = fs/N\_ubelastet;  
freq\_akse\_ubelastet = 0:bin\_ubelastet\_res:fs-bin\_ubelastet\_res;  
fft\_ubelastet = fft(signal\_ubelastet - est\_middel\_ubelastet);

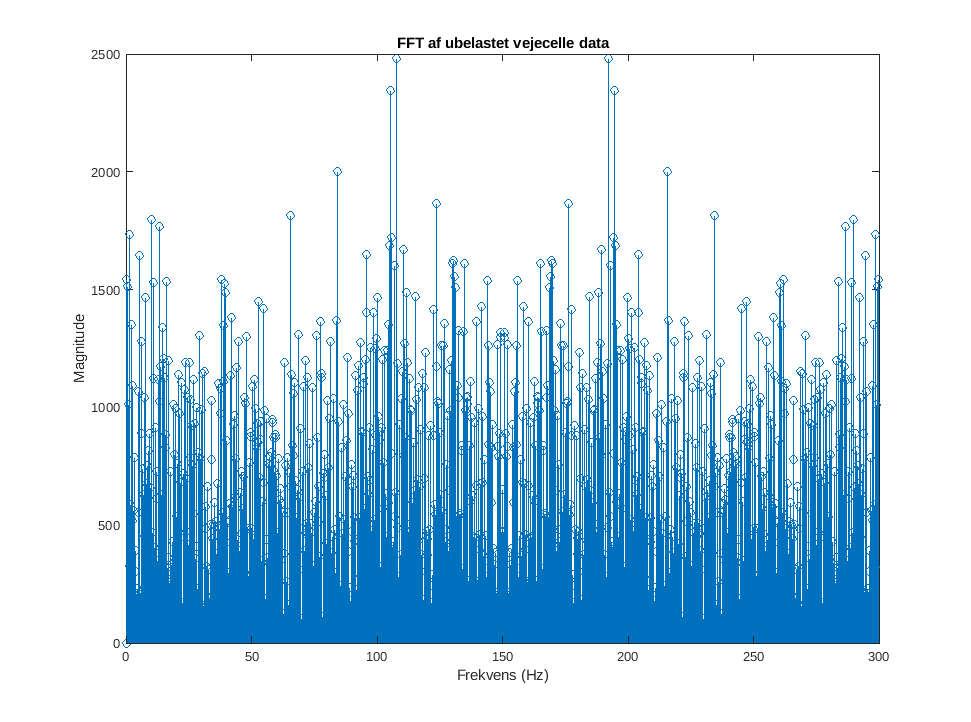
I ovenstående kodeudsnip bliver opløsningen på frekvens bins udregnet. Denne findes til 0,3 Hz. Derefter oprettes der en frekvens-akse, som bliver anvendt i plottet. Til sidst kaldes fft() funktionen, som laver en DFT på signalet. Bemærk at DC-værdien (den estimerede middelværdi) er blevet trukket fra signalet, hvilket er lidt atypisk. Dette skyldes at der i opgaven ønskes at kigge efter hvid støj, hvilket har en DC-værdi på 0.

stem(freq\_akse\_ubelastet, abs(fft\_ubelastet));  
title('FFT af ubelastet vejecelle data');  
ylabel('Magnitude');  
xlabel('Frekvens (Hz)');

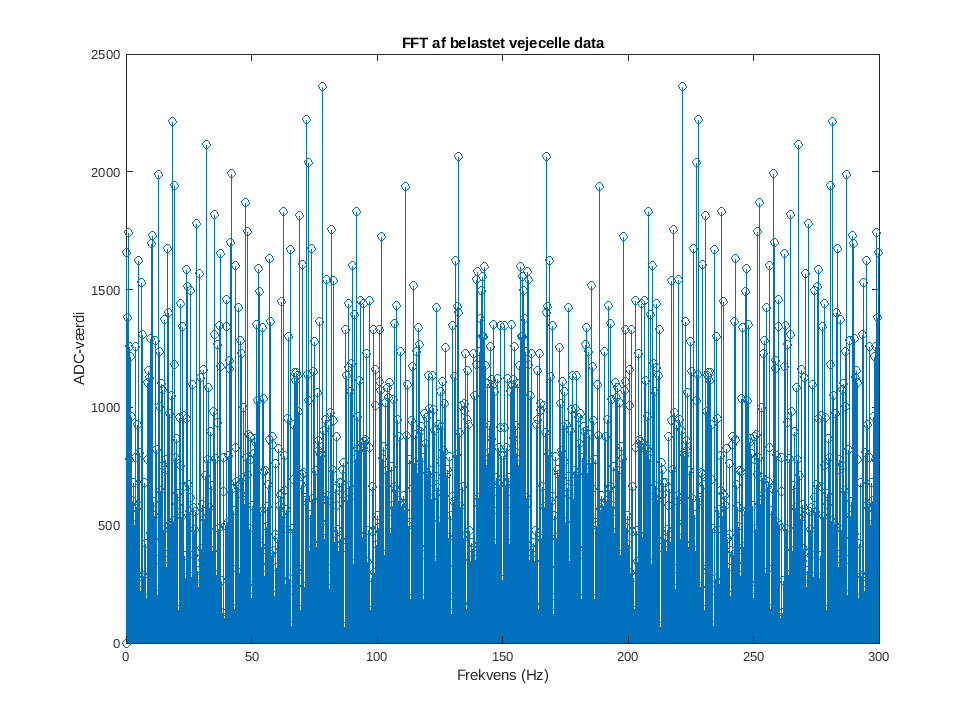
De forskellige plots kan genereres ved at kalde stem() funktionen i Matlab, hvor argumenterne er frekvens-aksen og det ubelastede signal i frekvens-domænet.

Det hele gentages for det belastede signal.

### Resultat



Figur 4 - Magnitude plot for det ubelastede signal



Figur 5 - Magnitude plot for det belastede signal.

På figur 4 og 5 ses de to magnitude plots.

### Konklusion

Ud fra frekvensspektret på Figur 4 og Figur 5 kan det ses, at der fremtræder støj på signalet, som er næsten ligeligt fordelt udover alle frekvenser. Dette er karakteristika for hvid støj, at det er fordelt tilfældigt over alle frekvenser. Det kan dermed konkluderes at der fremtræder hvid støj på signalerne.

## Opgave 1d (Morten)

I den sidste delopgave skal det vurderes hvad opløsningen på vejecellen er, dvs. hvor meget én bit svarer til i gram.

### Teori

For at finde opløsningen på vejecellen, bør blikket rettes mod forskellen mellem det belastede signal og det ubelastede signal. Her kan den estimerede middelværdi bruges, for at fjerne bidraget fra støjen. Da vægten på det påførte lod er kendt, kan man tage differencen mellem gennemsnittet for det belastede signal og det ubelastede signal. Man kan derefter dividere ændringen i vægt med forskellen på de to estimerede middelværdier.

### Implementering

Implementeringen af opløsningen ses i kodeudsnippet nedenfor. Først findes forskellen mellem de to middelværdier, og derefter findes opløsningen.

% Forskel mellem de to middelværdier:  
diff = est\_middel\_belastet - est\_middel\_ubelastet;  
% Opløsning (vi ved at loddet vejer 1000 g):  
res = 1000/diff;

### Resultat

Resultatet giver 3,36, hvilket svarer til en opløsning på 3,36 g pr. bit.

### Konklusion

Det kan konkluderes at vægten har en opløsning på 3,36 g. Det vil sige, at hver gang ADC-værdien stiger med 1, svarer det til at vægten er steget med 3,36 g.

# Opgave 2 – Implementering af filtre og analyse af filtrerede data (Peer & Mojtaba)

I denne opgave skal der implementeres et normalt og et eksponentielt midlingsfilter. Der vil ses på effekten af filtrene, hvis de anvendes og analyseres, hvordan effektiviteten af disse ændres ved at ændre deres parametre.

## Opgave 2.a (Peer)

I denne opgave skal der implementeres midlingsfiltre med orden 10, 50 og 100, der skal filtrere støjen fra måledataene fra vejecellen.

### Teori

Der findes to forskellige typer midlingsfiltre man kan anvende for at filtrere støj væk fra måledate. Den første type er et ikke-rekursivt filter og anden type er et rekursivt filter. Hovedforskellen mellem de to typer filtre er, hvor effektive de er i forhold til antallet af regneoperationer der skal gennemføres. Her er det sådan, at det rekursive midlingsfilter kun har behov for to additioner for at generere et output uanset filterlængden. Ved det ikke-rekursive midlingsfilter er der behov for næsten samme antal additioner som filterlængden, hvor antallet af additioner svarer til M-1, hvor M er filterlængden. Det betyder, at et rekursivt midlingsfilter meget hurtigere kan beregne filteroutputtet og der ikke kræves så meget processor power. Ulempen ved et rekursivt filter er dog, at det kan komme til overflowfejl, da den tager sidste outputsample til beregning af næste output og tallet dermed kan vokse større og større. Det vil dog ikke være et problem for use-casen i denne opgave. Formlen for et rekursivt midlingsfilter kan ses nedenfor:

Herfra kan der ses, at for ethvert output sample vil differencen mellem den aktuelle sample-værdi og sample-værdien fra *M* samples før divideres med filterlængden *M*, der svarer til at beregne middelværdien. Dette vil så adderes med sidste output sample.

### Implementering

For implementeringen af det rekursive midlingsfilter anvendes samme kode/funktion som brugt til opgaveregning i Øvelse 1.

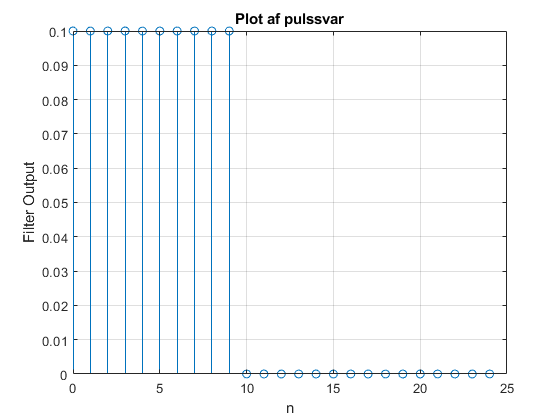
Implementeringen for det rekursive midlingsfilter kan ses i kodeudsnittet nedenfor:

 xmem = zeros(1, M); % initialize vector of last M samples memory  
ymem = 0; % Memory for last output  
y = zeros(1, N-1); % Vector containing all output values  
   
**for** n=1:N  
    y(n)= 1/M.\* (x(n) - xmem(**end**)) + ymem; % Calculation of output value  
    xmem = [x(n), xmem(1:**end**-1)]; %Pushing current sample value in sample memory  
    ymem = y(n); % Assigning last output memory variable to current output

Først initialiseres der en memory vektor *xmem*, der gemmer de sidste *M* sampleværdier, hvor *M* er filterlængden. Derefter bliver *ymem* variablen defineret, der holder sidste output sample pga. at filteret jo er rekursivt. Derefter initialiseres outputvektoren *y*, der vil indeholde de filtrerede outputsamples. Følgende bliver der i en for-loop iterativt beregnet outputsamples for alle inputsamples. Disse pushes i memory vektoren *xmem* og outputsamplen gemmes i output memory variablen *ymem*.

### Resultat

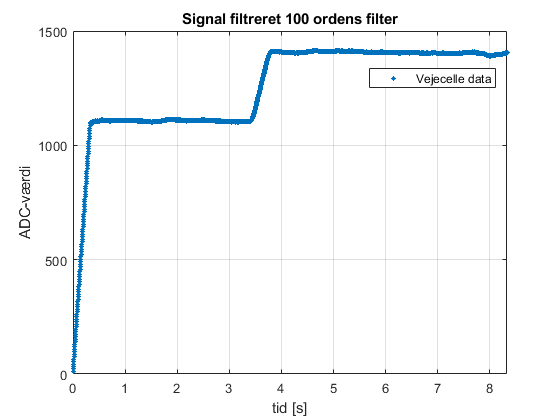
For en god ordens skyld er det første der ses på, impulsresponsen af det anvendte filter. Her vil der eksempelvis ses på filteret med en filterorden på 10.



Figur 6 - Impulssvar af 10. ordens rekursiv midlingsfilter

Som der kan ses, vil pulssvaret have samme output indtil vi kommer til sample 10. Dette skyldes, at pulssvaret genereres ved at første sample er et 1-tal og alle efterfølgende samples er 0-taller. Da filteret er et moving average filter med 10. orden betyder det, at der på hvert samples har en vægt på 1/10 eller bedre sagt ganges med tallet. Derfor ses at outputtet er 1/10, altså 0,1. Grunden til, at den ved 10. sample falder til 0 igen er, at alle forrige samples, der indeholdt tallet 1 eller 0,1, afhængigt om det er et ikke-rekursivt eller rekursivt filter, blev gennemløbet og nu består de sidste samples kun af 0-taller. Herfra kan der også ses, at pulssvaret for et 10. ordens rekursivt filter tager 10 samples. Denne information kan bruges senere hen.

Som det næste vil der ses på et plot af vejecelle dataene, men hvorpå der er anvendt et 100. ordens midlingsfilter.



Figur 7 - Plot af vejecelledata med 100. ordens filter anvendt

Sammenlignes Figur 7 med plottet af de ufiltrerede data, Figur 1, så kan man helt klart se, hvordan måledataene ligner en hel del mere en vandret linje.

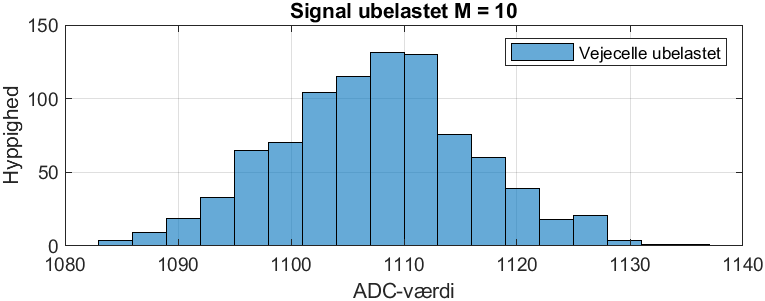
### Konklusion

Som der tydeligt kan ses på Figur 7 virker midlingsfilteret som ønsket. Næsten hele støjen i måledataene er blevet fjernet ved anvendelsen af filteret.

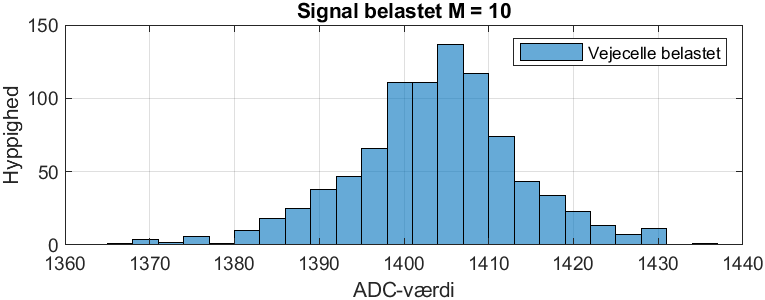
## Opgave 2.b (Peer)

I denne opgave ses der på histogrammer over vejecelle-dataene, hvorpå der blev anvendt midlingsfiltre med forskellige ordener. Der vil ses på, om den teoretiske og praktiske ændring i variansen passer sammen. Der skal også nævnes, at dataene for histogrammerne kun består af 900 samples, for at der kun ses på måledataenes steady-state, da der i begyndelses af vejecelle-dataene ikke hersker steady-state endnu.

### Resultater

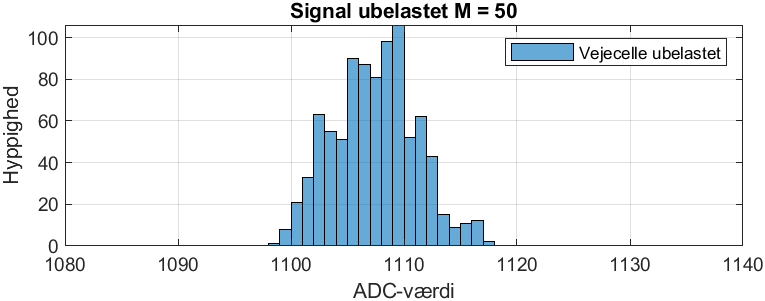


Figur 8 - Histogram over ubelastede måledate filtreret med 10. ordens filter

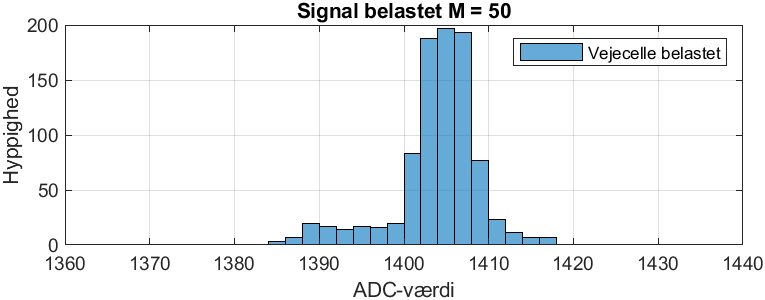


Figur 9 - Histogram over belastede måledate filtreret med 10. ordens filter

Hvis man ser på Figur 8 og Figur 9 og sammenligner dem Figur 3, så kan man tydeligt se, at spredningen mellem den maksimale og minimale værdi for de to histogrammer er blevet mindre. Der er stadig støj på, men reduceret i forhold til de ufiltrerede data.

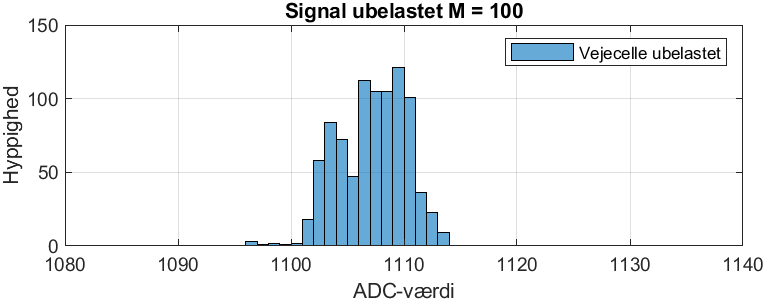


Figur 10 - Histogram over ubelastede måledate filtreret med 50. ordens filter

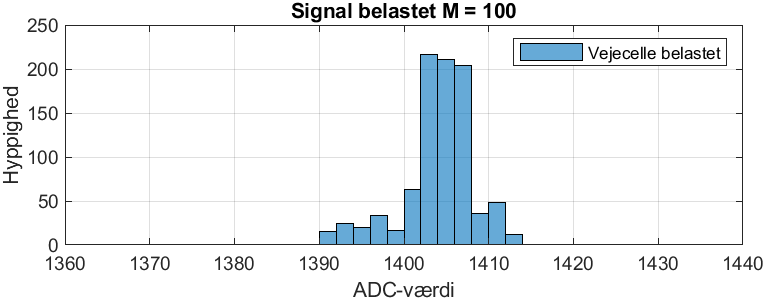


Figur 11 - Histogram over belastede måledate filtreret med 50. ordens filter

Sammenlignes Figur 10 og Figur 11 med de ufiltrerede data, Figur 3, kan man pludselig se en signifikant forbedring af spredningen mellem min og max værdier på histogrammerne af de filtrerede data. Dette skyldes, at filtreret nu er af 50. orden og dermed har en meget større effektivitet end det 10. ordens filter.



Figur 12 - Histogram over ubelastede måledate filtreret med 100. ordens filter



Figur 13 - Histogram over belastede måledate filtreret med 100. ordens filter

Til sidst sammenlignes histogrammerne Figur 12 og Figur 13 med de ufiltrerede data på Figur 3. Her kan der i forhold til 50. ordens filteret, Figur 10 og Figur 11, ikke ses et signifikant spring i effektiviteten, dog kan man stadig se en forbedring af effektiviteten af filteret.

Nu skal der ses på, om variansændringen i teorien og praksis passer sammen. Den teoretiske ændring kan beregnes med følgende formel:

Denne formel anvendes nu for at beregne den teoretiske ændring i variansen ud fra variansen fra de ufiltrerede data for belastet og ubelastet tilstand. Resultaterne kan ses i følgende tabel:

|  |  |  |
| --- | --- | --- |
| Type | Teoretisk ændring | Beregnet varians |
| Ufiltreret ubelastet |  | 764,93 |
| Ufiltreret belastet |  | 905,76 |
| 10. ordens filter ubelastet | 76,49 | 74,78 |
| 10. ordens filter belastet | 90,58 | 101,74 |
| 50. ordens filter ubelastet | 15,29 | 13,40 |
| 50. ordens filter belastet | 18,12 | 25,77 |
| 100. ordens filter ubelastet | 7,65 | 8,91 |
| 100. ordens filter belastet | 9,06 | 17,87 |

Tabel 2 - Teoretisk og praktisk varians-ændring

Hvis man nu ser på de teoretiske og praktiske værdier, så vil man kunne se, at værdierne for den teoretiske og praktiske ændring af den ubelastede tilstand stemmer meget godt overens med kun en lille afvigelse. Derimod kan man se, at der ikke gælder det samme for de belastede data. Her kan man se en større afvigelse. En af grundene kunne være, at der i den belastede tilstand sker et lille dip af værdierne i slutningen, se Figur 7.

### Konklusion

Ud fra de forskellige histogrammer kan der ses, at jo større filterordenen er, jo mindre spredning findes der mellem min og max værdierne. Dette tyder på, at mere og mere støj fjernes og det dermed bevæger sig i retning af konstante værdier, hvis man ville anvende filtre med endnu større ordener.

## Opgave 2.C (Peer)

I denne opgave skal den maksimale filterorden findes, for at indsvingningstiden ikke overstiger 100ms.

### Teori

Ud fra plottet af impulsrespons på Figur 6 ved vi, at det tager et M.-ordens midlingsfilter M samples for at den indsvinger. Deruodover kender vi også samplingfrekvensen for vejecelle-dataene. Dermed kan filterordenen beregnes ved at dividere indsvingningstiden med samplingfrekvensen. Beregningen ser således ud:

### Resultat

Resultatet bliver, at det tager 30 samples, for at filteret er indsvinget. Dette betyder, at filteret maksimalt må være af 30. orden for at overholde kravet af 100ms indsvingningstid.

## Opgave 2.d (Mojtaba)

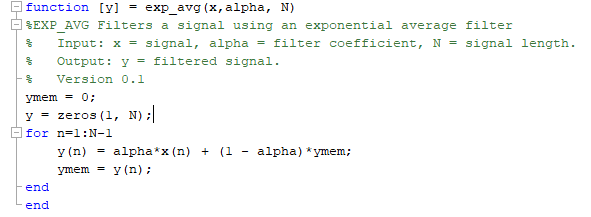
I denne opgave skal der implementeres et eksponentielt midlingsfilter hvorefter der skal eksperimenter med α-værdien - også kaldet for vægtning faktor - med forskellige værdier.

### Teori

α-værdien, som også kan kaldes for vægtningsfaktor, har en værdi fra 0 til 1. Hvis α-værdi har en værdi på 1, så vil output være lige med input det vil sige at ingen støj-reduktion-filter er påtaget. Har vi derimod α-værdi på meget lav værdi, så vil der reducere støj og til gengæld vil man få langsom respons og stabilisere signalet end hvis man har et større α-værdi. Ligningen [7] ved nedenstående tager signalet fra input og ganger med vægtningsfaktor og ligger sammen med forrige output signalet ganget med vægtningsfaktor, hvorefter vi får en output signal.

### Implementering

Vi opstiller en funktion med en eksponentiel for exp\_avg hvor den har 3 parametre, input, vægtningsfaktor og antal samples. Herefter initialiserer vi ymem til at være 0 da den skal bruges til at gemme værdier i en for-løkke. Vi danner en række med masse nuller (zeros) med N-antal søjle. Følgende bliver der i en for-løkke beregnet outputsamples for input signal ganget med vægtningsfaktor, og addere sammen med forrige outputsample hvorefter den ganges med difference mellem 1 og vægtningsfaktor. Til sidste gemmes outputsignalet til memory variablen ymem.



### Resultat



Figur 14: Pulssvar med α = 0.1



Figur 15: Pulssvar med α = 0.5



Figur 16: Pulssvar med α = 0.9

Vi kan ud fra disse 3 forskellige grafer se at der er givet forskellige α-værdi, hvor det tager mindre tid, hvis α-værdi er højere. Desuden vil højere α-værdi reagerer hurtigt på skift i signal og det betyder at filteret vil reducere støjen mindre, hvilket så betyder at der vil forekomme mere højfrekvent støj i outputtet. Hvis man ser på y-aksen, vil man kunne se, at jo højere α-værdien er, jo større er amplituden af impulsresponsen. Dette er også en egenskab af filtre med store α-værdier, da filtreringen ikke er så effektivt for sådanne pulser, der kan anses som højfrekvent støj.

### Konklusion

Vi kan tydeligt se at støjen på Figur 14 bliver filtrer mere end i forhold til andre værdier som er vist på de forskellige figurer ved ovenstående.

## Opgave 2.e (Mojtaba)

I denne delopgave skal der sættes α-værdien sådan at vi får samme støj reduktion for vores 100. ordens FIR midlingsfilter.

### Teori

Ligningen nedenstående kan bruges til at ændre på vores orden, og dette gøres ved at ændre på R efter ønsket antal orden. R er støj reduktion faktor [8].

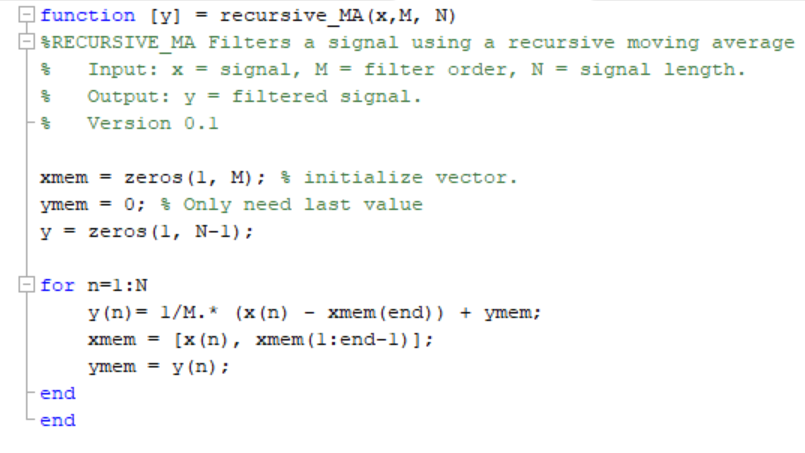
### Implementering

Vi opstiller formel for vores α-værdien og sætter R til den ønsket orden som er på 100. Som tidligere skrevet skal α-værdi ligges mellem 1 og 0.

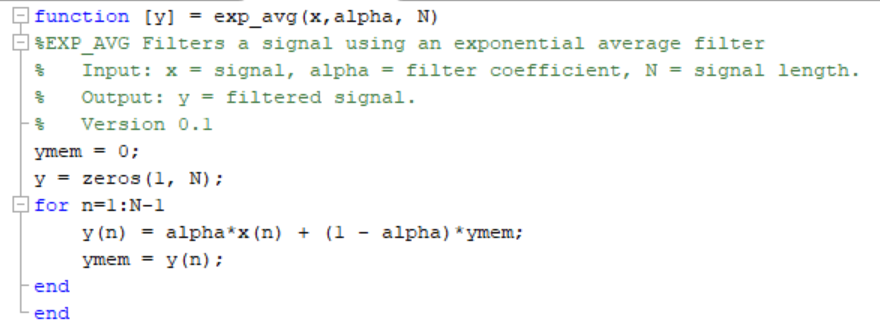


Vi har fået α-værdi til at være 0.0198, hvilket ligger mellem 1 og 0.

Herefter bruger vi samme implementering fra tidligere opgaver for det rekursive midlingsfilter. Det samme gælder for eksponentiel moving average.



Figur - Implementering af rekursivt midlingsfilter



Figur - Implementering af eksponentielt midlingsfilter

### Resultat



Figur : Plot af signal filtreret med eksponentiel filter



Figur : Plot af signal filtreret med rekursivtl filter

På Figur 19 kan vi se at signal er filtreret med eksponentiel filter hvor der dannes runde indsvingningstiden. På Figur 20 har vi signal er filtreret med rekursivt filter, som dannes mere en lineær linje. Desuden kan vi se at signalet bliver stabile ved 3,8 s hvorimod ved eksponentiel det lidt længere tid at holde stabile, som sker ved cirka 4,1 s.

### Konklusion

Vi kan se at signalet filtreret med rekursivt filter er hurtigere til at svare og holde stabile end der gøres ved eksponentiel filter, derfor er rekursiv filter mere fordele at bruge for vores use-casen.

## Opgave 2.f (Mojtaba)

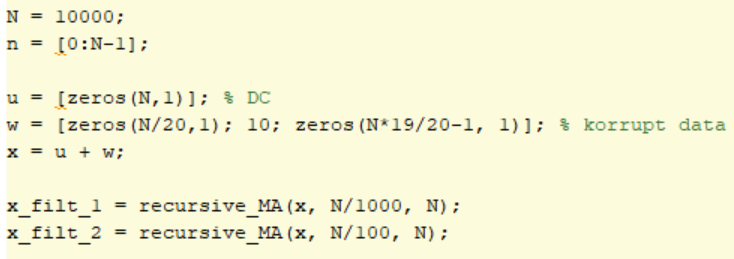
I denne delopgave skal vi se på hvad for betydning korrupt data har for vores signal.

### Teori

Korrupt data vil medføre ændringer på filtrets outputtet, hvis signal med korrupt data er stor. Det vil sige at hvis korrupt data er stor, så vil filtrets output afvige og svinge meget, så for eksempel et interrupt ville kunne trigges af et korrupt signal, hvis den korrupte del afviger meget fra det normale signal.

### Implementering

Vi starter med at skrive antal output samples og input samples. Derefter opstiller vi DC hvor der dannes en masse nuller med 10000 samples og angives variablen u. Følgende bliver der lavet korrupt data, hvor



### Resultat



Figur : Plot af signal med korrupt data



Figur : Plot af filtreret signal med korrupt data med M = 10



Figur : Plot af filtreret signal med korrupt data med M = 100

På Figur 21 har vi uden filter hvor vi kan se at der er et unaturligt højt spike, som ikke passer ind på vores signal. Ser vi på Figur 22 har vi en 10. ordens filter hvor spikes er blevet filtreret og gjort mindre. På Figur 23 har vi en 100 ordens filter hvor spike er blevet gjort betydeligt meget mindre, hvor den næsten ikke har nogen effekt på vores måledataene. Det vil sige at den er meget mindre følsom end hvis man har højere spikes hvor måledataene vil få en stor effekt ændring. Desuden kan der siges, at jo højere ordenen bliver, jo mindre bliver frekvensspektret der bliver ladt igennem filteret, da det arbejder som et lavpasfilter.

### Konklusion

Vi kan konkludere at det er mere fordele at arbejde med højere orden end med mindre orden, da man ønsker at have mindre effekt ændring på måledataene, så længe der ikke arbejdes med applikationer, der kræver en hurtigst mulig filtrering.

# Opgave 3 (Morten)

I denne opgave vil det blive undersøgt, hvor mange betydende cifre et display med vejecellen vil kunne vise, såfremt der stilles krav til støjens spredning om, at den skal ligge på under 1/10-del af værdien af det mindst betydende ciffer. Vejecellen er desuden forbundet til et 100. ordens FIR midlingsfilter.

### Teori

For at finde den mindste værdi displayet retvisende vil kunne vise under de givne krav, skal man anvende standardafvigelsen og opløsningen fra de tidligere opgaver. For at efterleve kravene skal standardafvigelsen ganges op med 10. Derefter kan man tage den fundne opløsning og gange med resultatet fra før. Antallet af betydende cifre svarer så til at runde op til nærmeste værdi.

### Implementering

I kodeudsnippet nedenfor ses implementeringen. Der tages udgangspunkt i de to standardafvigelser, hvortil den højeste vælges for at få worst case.

% Sætter M til 100 i opgave 2 og får følgende:  
std\_no\_load = 2.98408133026240  
std\_load = 4.22730656201442  
res = 3.36 % 3.36 gram pr. bit (fra tidl. opg.)  
std\_load\_10 = std\_load\*10; % Tager med load fordi det er worst-case  
min\_acc = std\_load\_10 \* res;

### Resultat

Resultatet bliver at min\_acc giver 142. Det svarer til at den mindste værdi vejecellen retvisende vil kunne vise er 142 g.

### Konklusion

Da vægten kun kunne vise 142 g retvisende, vil displayet kunne vise med en præcision på 0 betydende cifre. Det vil sige at vejecellen udelukkende kan give et retvisende resultat i hele kilo. Man kan overveje hvorvidt der skal anvendes et højere ordens filter, for at forbedre standardafvigelsen yderligere.

# Appendix

## References

[1] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 868

[2] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 869

[3] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 870

[4] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 883

[5] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 61

[7] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 608

[8] Richard G. Lyons, 3. ed. (2011) Understading Digital Signal Processing, p. 610

## Matlab Kode Implementering

### Main Kode

%% Opgave 1  
clf; clear; clc; close all;  
load('vejecelle\_data.mat');  
fs = 300; %Hz  
T\_length = length(vejecelle\_data)/fs; % Antal sekunder vi sampler over  
N = length(vejecelle\_data);  
t = [0:1/fs:N/fs-1/fs]; % tidsakse   
   
figure(10);  
plot(t,vejecelle\_data, '\*', 'MarkerSize', 3)  
title('Signal ufiltreret')  
ylabel('ADC-værdi')  
xlabel('tid [s]')  
legend('Vejecelle data');  
xlim([0 t(**end**)]);  
grid on;  
   
signal\_ubelastet = vejecelle\_data(1:1000);  
signal\_belastet = vejecelle\_data(length(vejecelle\_data)-999:length(vejecelle\_data));  
   
N\_ubelastet = length(signal\_ubelastet);  
N\_belastet = length(signal\_belastet);  
   
% Estimeret middelværdi:  
est\_middel\_ubelastet = 1/N\_ubelastet \* sum(signal\_ubelastet);  
est\_middel\_belastet = 1/N\_belastet \* sum(signal\_belastet);  
% Estimeret varians:  
est\_var\_ubelastet = 1/(N\_ubelastet-1)\*sum((signal\_ubelastet-est\_middel\_ubelastet).^2);  
est\_var\_belastet = 1/(N\_belastet-1)\*sum((signal\_belastet-est\_middel\_belastet).^2);  
% Estimeret standardafvigelse:  
est\_std\_ubelastet = sqrt(est\_var\_ubelastet);  
est\_std\_belastet = sqrt(est\_var\_belastet);  
   
   
figure(20);  
subplot(2,1,1);  
histogram(signal\_ubelastet);  
title('Signal ubelastet')  
ylabel('Hyppighed')  
xlabel('ADC-værdi')  
legend('Vejecelle ubelastet');  
grid on;  
   
subplot(2,1,2);  
histogram(signal\_belastet);  
title('Signal belastet')  
ylabel('Hyppighed')  
xlabel('ADC-værdi')  
legend('Vejecelle belastet');  
grid on;  
   
   
bin\_ubelastet\_res = fs/N\_ubelastet;  
freq\_akse\_ubelastet = 0:bin\_ubelastet\_res:fs-bin\_ubelastet\_res;  
% Bemærk vi har trukket middelværdien fra da dennes energi er meget større  
% end støjens enkelte bins:  
fft\_ubelastet = fft(signal\_ubelastet - est\_middel\_ubelastet);   
   
figure(30);  
stem(freq\_akse\_ubelastet, abs(fft\_ubelastet));  
title('FFT af ubelastet vejecelle data');  
ylabel('Magnitude');  
xlabel('Frekvens (Hz)');  
   
bin\_belastet\_res = fs/N\_belastet;  
freq\_akse\_belastet = 0:bin\_belastet\_res:fs-bin\_belastet\_res;  
% Bemærk vi har trukket middelværdien fra da dennes energi er meget større  
% end støjens enkelte bins:  
fft\_belastet = fft(signal\_belastet - est\_middel\_belastet);  
   
figure(40);  
stem(freq\_akse\_ubelastet, abs(fft\_ubelastet));  
title('FFT af belastet vejecelle data');  
ylabel('ADC-værdi');  
xlabel('Frekvens (Hz)');  
   
% Spørgsmål: Ligner det hvid støj? Det ligner ret godt hvid støj, selvom  
% det ikke er helt normalfordelt.  
   
% Spørgsmål: Hvad er afstand imellem bit-niveauer i gram ? (dvs. værdi af  
% LSB)  
   
% Forskel mellem de to middelværdier:  
diff = est\_middel\_belastet - est\_middel\_ubelastet;  
% Opløsning (vi ved at loddet vejer 1000 g):  
res = 1000/diff;  
% Dvs. 1 bit = res gram. res = ~3.36g.  
   
%% Pre-Opgave 2  
% Viser filterkarakteristik  
clf; clear; clc; close all;  
   
pulse = [1, zeros(1,25)];  
   
% Implementering af midlingsfiltre - der er valgt rekursivt MA.  
ma\_10 = recursive\_MA(pulse, 10, 25);  
   
   
figure(50);  
stem([0:24],ma\_10);  
title('Plot af pulssvar')  
ylabel('Filter Output')  
xlabel('n')  
grid on;  
   
%% Opgave 2:  
clf; clear; clc; close all;  
load('vejecelle\_data.mat');  
fs = 300; %Hz  
T\_length = length(vejecelle\_data)/fs; % Antal sekunder vi sampler over  
N = length(vejecelle\_data);  
n = [0:N-1]; % Antal samples  
   
M = 100; % Ændre M for nye billeder  
   
signal\_filt = recursive\_MA(vejecelle\_data, M, N);  
   
% Vi har valgt at tage 900 samples her, for at undgå at se artefaktor med  
% at signalet skal finde steady-state i de første samples efter filteret.  
s\_noload = signal\_filt(100:999);  
s\_load = signal\_filt(length(signal\_filt)-899:length(signal\_filt));  
   
N\_noload = length(s\_noload);  
N\_load = length(s\_load);  
   
% Estimeret middelværdi:  
mean\_noload = 1/N\_noload \* sum(s\_noload);  
mean\_load = 1/N\_load \* sum(s\_load);  
% Estimeret varians:  
var\_noload = 1/(N\_noload-1)\*sum((s\_noload-mean\_noload).^2);  
var\_load = 1/(N\_load-1)\*sum((s\_load-mean\_load).^2);  
% Estimeret standardafvigelse:  
std\_noload = sqrt(var\_noload);  
std\_load = sqrt(var\_load);  
   
figure(60);  
subplot(2,1,1);  
histogram(s\_noload);  
title(['Signal ubelastet M = ' ,num2str(M)])  
ylabel('Hyppighed')  
xlabel('ADC-værdi')  
legend('Vejecelle ubelastet');  
grid on;  
xlim([1080 1140]);  
   
subplot(2,1,2);  
histogram(s\_load);  
title(['Signal belastet M = ' ,num2str(M)])  
ylabel('Hyppighed')  
xlabel('ADC-værdi')  
legend('Vejecelle belastet');  
grid on;  
xlim([1360 1440]);  
   
fs = 300; %Hz  
T\_length = length(vejecelle\_data)/fs; % Antal sekunder vi sampler over  
N = length(vejecelle\_data);  
t = [0:1/fs:N/fs-1/fs]; % tidsakse   
   
figure(500);  
plot(t,signal\_filt, '\*', 'MarkerSize', 3)  
title(['Signal filtreret ', num2str(M), ' ordens filter']);  
ylabel('ADC-værdi')  
xlabel('tid [s]')  
legend('Vejecelle data');  
xlim([0 t(**end**)]);  
grid on;  
   
% Spørgsmål: Stemmer ændringen af variansen/effekten overens med teorien ?  
% Teorien passer ret godt. Ved M = 10 er variansen cirka faldet med en  
% faktor 10 (vis flere eksempler i rapport)  
   
% Et krav til max. indsvingnings-tid kunne være 100 millisekunder til et  
% praktisk veje-system. Beregn den maksimale længde af jeres FIR  
% midlingsfilter.  
   
max\_time = 0.1; % [s]  
max\_order = fs\*max\_time;  
   
%% Exp-Opgave 2  
%Implementér et eksponentielt midlingsfilter. Eksperimenter med ?-værdien –  
%prøv f.eks. at sætte den meget lavt / højt. Hvad er betydningen af  
%?-værdien ?  
   
clf; clear; clc; close all;  
   
pulse = [1, zeros(1,50)];  
   
% Implementering af midlingsfiltre - der er valgt rekursivt MA.  
exp1 = exp\_avg(pulse, 0.5, length(pulse));  
figure(70);  
subplot(2,2,1);  
stem([0:length(pulse)-1],exp1);  
title('Plot af pulssvar \alpha = 0.5')  
ylabel('Spænding [V]')  
xlabel('n')  
grid on;  
   
   
% Implementering af midlingsfiltre - der er valgt rekursivt MA.  
exp2 = exp\_avg(pulse, 0.1, length(pulse));  
subplot(2,2,2);  
stem([0:length(pulse)-1],exp2);  
title('Plot af pulssvar \alpha = 0.1')  
ylabel('Spænding [V]')  
xlabel('n')  
grid on;  
   
% Implementering af midlingsfiltre - der er valgt rekursivt MA.  
exp3 = exp\_avg(pulse, 0.9, length(pulse));  
subplot(2,2,3);  
stem([0:length(pulse)-1],exp3);  
title('Plot af pulssvar \alpha = 0.9')  
ylabel('Spænding [V]')  
xlabel('n')  
grid on;  
   
%% Opgave 2 cont.  
   
% Prøv at sætte ?-værdien, således at I får samme støj-reduktion, som for  
% jeres 100. ordens FIR midlingsfilter – Kommentér på indsvingnings-tiderne  
% (dvs. tiden fra “belastning” til “ingen belastning”).  
clf; clear; clc; close all;  
   
load('vejecelle\_data');  
   
% Fra bogen side 541:  
% alpha = 2/(N + 1), hvor N = FIR filterets orden og hvor N > 3  
alpha = 2/(100+1);  
   
N = length(vejecelle\_data);  
t = [0:1/fs:N/fs-1/fs]; % tidsakse   
   
signal\_filt\_exp = exp\_avg(vejecelle\_data, alpha, N);  
signal\_filt\_fir = recursive\_MA(vejecelle\_data, 100, N);  
   
   
% Plottes op imod hinanden.  
figure(80);  
subplot(2,1,1);  
plot(t,signal\_filt\_exp,'\*', 'MarkerSize', 3)  
title('Signal filtreret med eksponentielt filter')  
ylabel('ADC-værdi')  
xlabel('tid [s]')  
legend('Vejecelle data');  
grid on;  
xlim([3.333 4.33]);  
   
subplot(2,1,2);  
plot(t,signal\_filt\_fir,'\*', 'MarkerSize', 3)  
title('Signal filtreret med rekursivt filter')  
ylabel('ADC-værdi')  
xlabel('tid [s]')  
legend('Vejecelle data');  
xlim([3.333 4.33]);  
grid on;  
   
%% Opgave 2 cont.  
clf; clear; clc; close all;  
% Spørgsmål: Hvad vil betydningen af “korrupt data” være ? dvs. fx. et par samples med  
% en værdi, som er meget større end alle de andre. Problemet kan til dels  
% løses med et såkaldt ”median-filter”  
N = 10000;  
n = [0:N-1];  
   
u = [zeros(N,1)]; % DC  
w = [zeros(N/20,1); 10; zeros(N\*19/20-1, 1)]; % korrupt data  
x = u + w;  
   
x\_filt\_1 = recursive\_MA(x, N/1000, N);  
x\_filt\_2 = recursive\_MA(x, N/100, N);  
   
figure(90);  
subplot(2, 2, 1);  
stem(n,x)  
title('Plot af signal med korrupt data')  
ylabel('ADC-værdi')  
xlabel('n')  
ylim([0 11]);  
xlim([N/25 N/15]);  
hold on  
   
subplot(2, 2, 2);  
stem(n,x\_filt\_1)  
title(['Plot af filtreret signal med korrupt data. M = ', num2str(N/1000)])  
ylabel('ADC-værdi')  
xlabel('n')  
ylim([0 11]);  
xlim([N/25 N/15]);  
hold on  
   
subplot(2, 2, 3);  
stem(n,x\_filt\_2)  
title(['Plot af filtreret signal med korrupt data. M = ', num2str(N/100)])  
ylabel('ADC-værdi')  
xlabel('n')  
ylim([0 11]);  
xlim([N/25 N/15]);  
hold on  
   
% Svar: Jo mindre filteret er, jo større indflydelse vil hvert enkelt  
% korrupt data punkt have.  
   
%% Opgave 3:  
clf; clear; clc; close all;  
   
%Tag udgangspunkt i det 100. ordens FIR midlingsfilter. Hvor mange  
%betydende cifre kan I medtage i et display, hvis det skal vise vægt i kg  
%(op til fx. 5 kg) og hvis støjens spredning (=kvadrat af varians) skal  
%ligge på under 1/10 af værdien af det mindst betydende ciffer i displayet  
%? Kravet til støjens spredning stilles for at alle viste cifre er  
%pålidelige  
   
% Sætter M til 100 i opgave 2 og får følgende:  
std\_no\_load = 2.98408133026240  
std\_load = 4.22730656201442  
   
res = 3.36 % 3.36 gram pr. bit (fra tidl. opg.)  
   
std\_load\_10 = std\_load\*10; % Tager med load fordi det er worst-case  
min\_acc = std\_load\_10 \* res;  
   
%min\_acc = 142. I så fald kan vi have nul betydende cifre, dvs. vi kan kun  
%vise i hele kilo.

### Rekursivt Midlingsfilter (Fra Øvelse 1)

**function** [y] = **recursive\_MA**(x,M, N)  
%RECURSIVE\_MA Filters a signal using a recursive moving average  
%   Input: x = signal, M = filter order, N = signal length.  
%   Output: y = filtered signal.  
%   Version 0.1  
   
xmem = zeros(1, M); % initialize vector of last M samples memory  
ymem = 0; % Memory for last output  
y = zeros(1, N-1); % Vector containing all output values  
   
**for** n=1:N  
    y(n)= 1/M.\* (x(n) - xmem(**end**)) + ymem; % Calculation of output value  
    xmem = [x(n), xmem(1:**end**-1)]; %Pushing current sample value in sample memory  
    ymem = y(n); % Assigning last output memory variable to current output  
**end**  
**end**

### Eksponentielt Midlingsfilter (Fra Øvelse 2)

**function** [y] = **exp\_avg**(x,alpha, N)  
%EXP\_AVG Filters a signal using an exponential average filter  
%   Input: x = signal, alpha = filter coefficient, N = signal length.  
%   Output: y = filtered signal.  
%   Version 0.1  
ymem = 0;  
y = zeros(1, N);  
**for** n=1:N-1  
    y(n) = alpha\*x(n) + (1 - alpha)\*ymem;  
    ymem = y(n);  
**end**  
**end**