

ELE130 Anvendt fysikk og matematikk i robotprogrammering

Øving 3

Introduksjon til numerisk integrasjon, numerisk derivasjon og filtrering¹

Leveret av Lasse Lorentzen

¹Tilhørende filer: `oving3_skallfiler.zip` og L^AT_EX-mal i `oving3.zip`.

Om øvingen og innleveringen

- For å bli kjent med prinsippet bak numerisk integrasjon, numerisk derivasjon og filtrering, så er alle oppgavene i denne øvingen svært nyttige, og de vil fungere som et oppslagsverk for deg når du gjør senere øvinger.
- For å tilrettelegge for at du skal gjøre flest mulig oppgaver, så kan du laste ned og pakke opp `oving3_skallfiler.zip`. I denne zip-filen vil finne
 - skallfilen `oving3_skallfil.m` som inneholder uferdig kode for alle oppgavene i denne øvingen
 - skallfiler for alle funksjonene som du skal ferdigstille
 - simulink-skallfilen `oving3_a_m_skallfil.slx` for oppgavene 3a) og 3m)

Mange av oppgavene virker kanskje omfattende, men svært mye av “infrastruktur-koden” er allerede gitt.

- På tilsvarende måte som i øving 1 bør du bruke trikset med å først kommentere hele skriptet `oving3_skallfil.m` for deretter å ta bort kommentarene celle for celle.
- **Hensikten med øvingen er du skal bruke oppgavene som et verktøy til å gi deg forståelse og innsikt for de forskjellige temaene/begrepene.**
- **For å få øvingen godkjent må du minst gjøre følgende oppgaver:**
`b), d), e), f), h), i), j)`
- Oppgavene `a), c), g), k), l), m), n) o)` er også lærerike og nyttige, men frivillige. Dette er markert i toppteksten.
- **For å ta eksamen i emnet ELE130 så må denne øving være godkjent. Husk at øvingene må leveres individuelt.**
- Basis for denne øvingen er [kapittel 7, 8 og 9](#) i kompendiet.
- På samme måte som i øving 1 og 2 finner du en L^AT_EX-mal i filen `oving3.zip`.

Forskjellig L^AT_EX-kode

Under er det gitt litt forskjellig kode som du kan kopiere inn i svarene under de oppgavene du gjør. Husk bare på å oppdatere bokmerkene til nye, unike navn, så slipper du feilreferering.

- Resultatet fra scopet er vist i figur 1.



Figur 1: Resultat av blokkskjemaet i modellen.

- Svaret er gitt i ligning (1).

$$y = 4 \tag{1}$$

- Eksempel på bruk av `align` er vist i ligning (2). For å bruke kommandoen `\uuline` må du bruke pakken `ulem` som `\usepackage{ulem}`.

$$\begin{aligned} A &= \int_0^1 2 \cdot \sin(5 \cdot t) dt \\ &= -\frac{2}{5} \cdot \cos(5 \cdot t) \Big|_0^1 \\ &= -\frac{2}{5} \cdot \cos(5 \cdot 1) - \left(-\frac{2}{5} \cos(0) \right) \\ &= \underline{\underline{0.287}} \end{aligned} \tag{2}$$

- Du finner også mye L^AT_EX-kode i hver av .tex-filene til deloppgavene.

Numerisk integrasjon og derivasjon

I deloppgaven a)-e) skal du jobbe med følgende signal

$$u(t) = 2 \cdot t^2 \quad (3)$$

Basert på det diskret signalet u_k gjeldende ved tidspunktene t_k skal du numerisk integrere u_k og beregne y_k som en tilnærming til operasjonen

$$y(t) = \int_0^t u(\tau) d\tau + y(0) \quad \text{gitt } y(0)=0 \quad (4)$$

Videre skal du numerisk derivere u_k for å beregne v_k som en tilnærming til

$$v(t) = \frac{d}{dt} u(t) \quad (5)$$

Fra matematikken vet du at de analytiske uttrykkene for $y(t)$ og $v(t)$ er

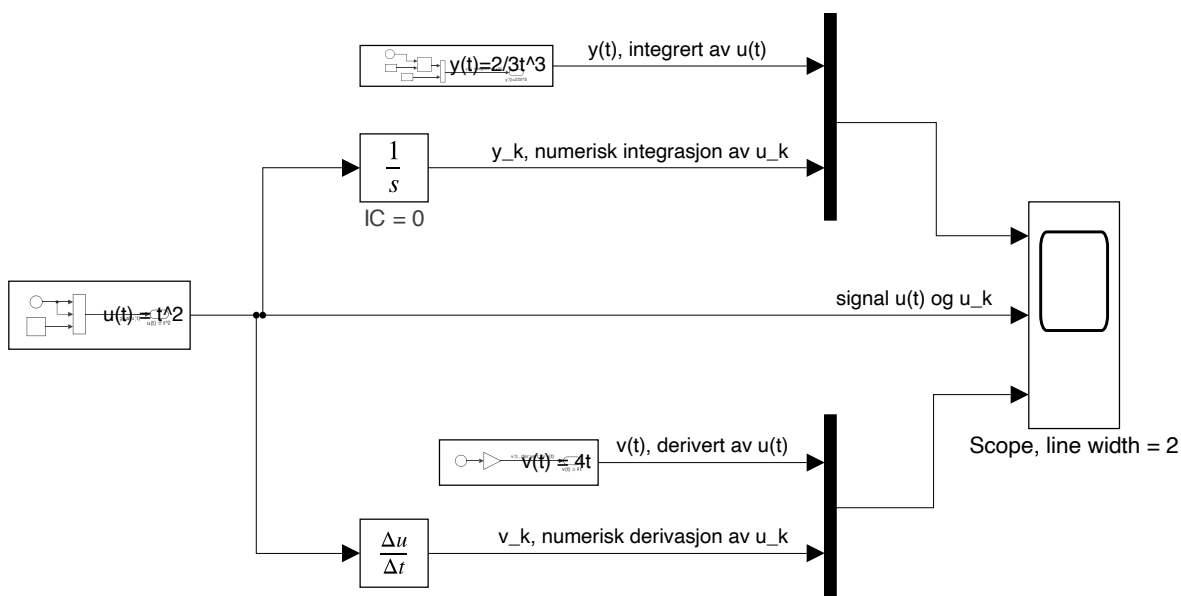
$$y(t) = \frac{2}{3} \cdot t^3 \quad (6)$$

$$v(t) = 4 \cdot t \quad (7)$$

Disse skal du benytte som “fasit” i noen av oppgavene slik at du visuelt kan sammenligne med de numeriske beregningene av v_k og y_k . Bruken av rød og blå farger brukes til å skille mellom henholdsvis derivering og integrering.

a) Numerisk integrasjon og derivasjon i Simulink

- Ta utgangspunkt i Simulinkmodellen i skallfilen `oving3_a_m_skallfil.slx` vist i figur 2, og implementer ligning (3) og de to analytiske løsningene i ligning (6) og (7) i hvert sitt tilhørende subsystem. Bruk blokkene `Clock`, `Math Function` hvor du velger `pow` fra rullegardinmenyen, samt `Product` for å lage disse signalene. Ta med skjermdump av de 3 delmodellene i innleveringen.



Figur 2: Simulinkmodell hvor ligningene (3), (6) og (7) er implementert i hvert sitt subsystem.

For å sammenligne analytiske og numeriske resultat velger vi å bruke tidspunkt $t=2$ sekund. De analytiske verdiene ved dette tidspunktet finner vi fra ligningene (6) og (7) som

$$y(2) = \frac{2}{3} \cdot 2^3 = 5.333 \quad (8)$$

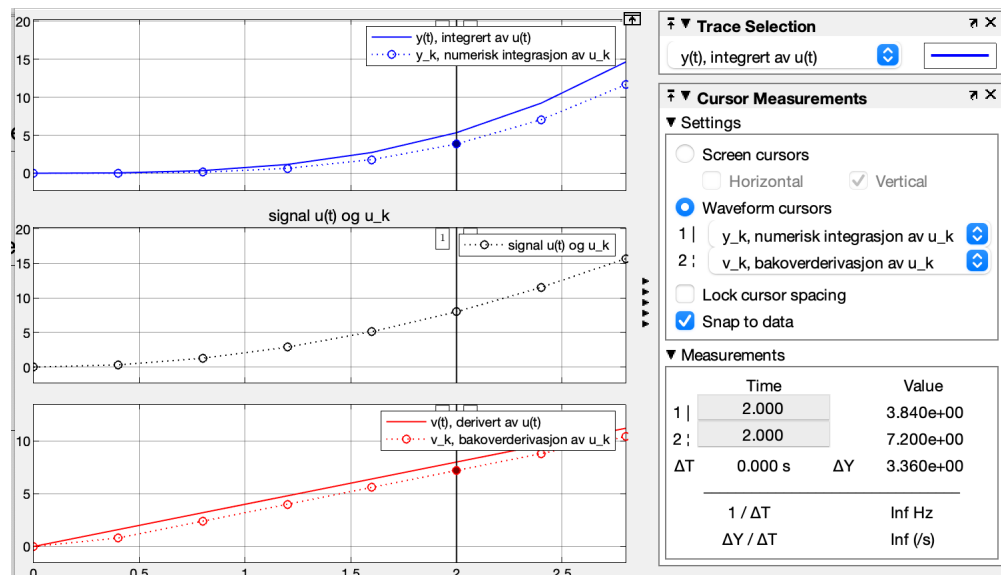
$$v(2) = 4 \cdot 2 = 8 \quad (9)$$

For å lese av de tilsvarende verdiene y_k og v_k ved $t_k=2$ sekund i scopet, er scopet i skallfilen klargjort med kurveavlesingsverktøyet hentet fra

`Tools -> Measurements -> Cursor Measurements`.

Det er huket av for `Snap to data`, men du må selv velge hvilket signal du skal lese av (i skallfilen er begge avlesningene på u_k). I skallfilen har vi lagt på markør på kurvene for å fremheve beregningstidspunktene.

- Spesifiser *Eulers forovermetode* (`ode1`) med steglengde lik $T_s=0.4$ sekund, simuler modellen i 3 sekund og vis at du får følgende respons



Figur 3: Simuleringsresultat

Ta med din egen respons inkludert avlesingene i innleveringen, og vis som over at de avleste verdiene ved $t_k=2$ sekund er

$$y_k = 3.840 \quad (10)$$

$$v_k = 7.2 \quad (11)$$

- Spesifiser deretter *Eulers bakovermetode* (`ode1be`) med samme steglengde og simuler på ny. Avles y_k og v_k ved samme tidspunkt. Ta med din egen respons inkludert avlesingene i innleveringen.

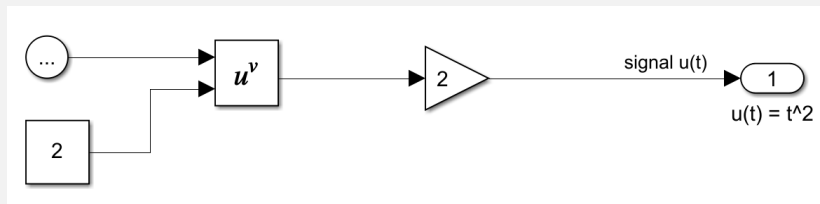
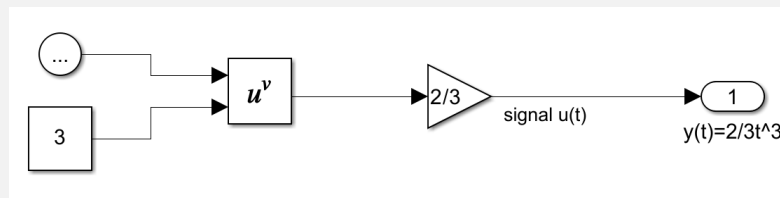
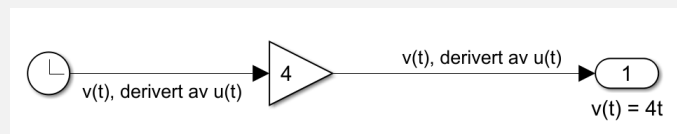
Gi en forklaring på endringen i den avleste verdien for y_k . Gi også en forklaring på hvorfor v_k er uendret.

- Spesifiser deretter *Heuns metode* (`ode2`) som tilsvarer *trapesmetoden*, og simuler på ny. Avles igjen y_k og v_k ved samme tidspunkt. Ta med din egen respons inkludert avlesingene i innleveringen.

Gi også her en forklaring på endringen i den avleste verdien for y_k .

Svar

Subsystemene er vist i figurene 4 til 6.

Figur 4: Blokkskjema for $u(t)2t^2$.Figur 5: Blokkskjema for integralet av $u(t)$, $y(t) = \frac{2}{3}t^3$.Figur 6: Blokkskjema for den deriverte av $u(t)$, $v(t) = 4t$.

b) Numerisk integrasjon og derivasjon i Matlab

I denne oppgaven skal du ta utgangspunkt i skallfilen `oving3_skallfil.m` for å lage kode for numerisk integrasjon av $u(t)$ fra ligning (3) ved å bruke både *Eulers forovermetode*, *Eulers bakovermetode* og *trapesmetoden*, henholdsvis gitt som:

$$y_k = y_{k-1} + T_s \cdot u_{k-1}, \quad \forall k=2, \dots, n \quad \text{gitt } y_1 \quad (12)$$

$$y_k = y_{k-1} + T_s \cdot u_k, \quad \forall k=2, \dots, n \quad \text{gitt } y_1 \quad (13)$$

$$y_k = y_{k-1} + T_s \cdot \frac{1}{2} \cdot (u_{k-1} + u_k), \quad \forall k=2, \dots, n \quad \text{gitt } y_1 \quad (14)$$

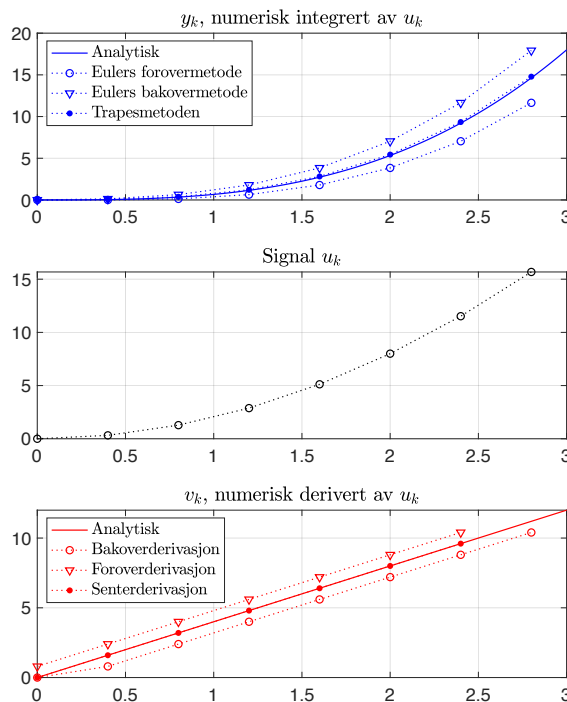
Du skal også utføre kodebasert numerisk derivasjon av $u(t)$ ved å bruke både *bakoverderivasjon*, *foroverderivasjon*, og *senterderivasjon*, henholdsvis gitt som

$$v_k = \frac{u_k - u_{k-1}}{T_s}, \quad \forall k=2, \dots, n \quad \text{gitt } v_1 = 0 \quad (15)$$

$$v_{k-1} = \frac{u_k - u_{k-1}}{T_s}, \quad \forall k=2, \dots, n \quad (16)$$

$$v_{k-1} = \frac{u_k - u_{k-2}}{2 \cdot T_s}, \quad \forall k=3, \dots, n \quad \text{gitt } v_1 = 0 \quad (17)$$

- Ferdigstill den uferdige koden for denne deloppgaven, og vis at du får følgende resultat (ta med din egen figur i innleveringen).

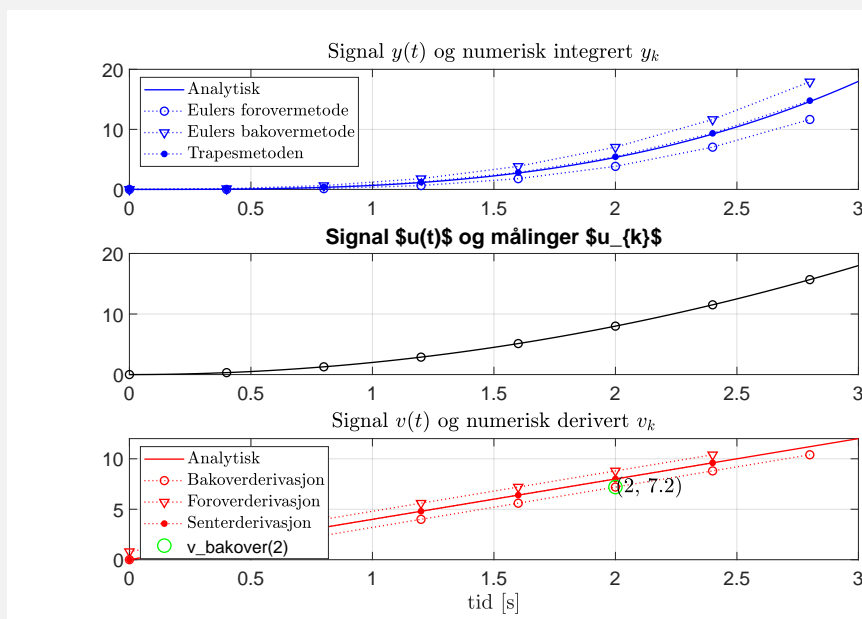


Figur 7: Simuleringsresultat

Når du skal plotte forover- og senterderivasjon, så må du passe på at dimensjonene til tidsvektoren og henholdsvis $v_{\text{forover}}(k)$ og $v_{\text{senter}}(k)$ er like lange.

- Avles verdiene av $y_{\text{EulerF}}(k)$, $y_{\text{EulerB}}(k)$ og $y_{\text{Trapes}}(k)$ ved tidspunkt $t_k=2$ sekund som i deloppgave a), og vis at de er identiske med y_k -verdiene du fant der.
- Avles verdien av $v_{\text{bakover}}(k)$ ved tidspunkt $t_k=2$ sekund som i deloppgave a), og vis at den er identiske med alle tre v_k -verdiene du fant der.

Svar



Figur 8: Resultat av numerisk integrasjon og derivasjon. Verdien av $v_{\text{bakover}}(k)$ er indikert i det nederste diagrammet.

c) Numerisk integrasjon og derivasjon for hånd

For å få litt eksamenstrening, skal du gjøre numerisk integrasjon og derivasjon for hånd basert på et begrenset datasett. Vi tar utgangspunkt i de fem første tidspunktene i t_k gitt som

$$t_k = [0, 0.4, 0.8, 1.2, 1.6, 2] \quad (18)$$

De tilhørende verdiene for u_k er

$$u_k = 2 \cdot t_k^2 \quad (19)$$

$$= [0, 0.32, 1.28, 2.88, 5.12, 8] \quad (20)$$

- Bruk *Eulers forovermetode* til manuelt å beregne det numeriske integralet av u_k , som betyr å finne

$$y_k = y_{k-1} + T_s \cdot u_{k-1}, \quad \forall k = 2, \dots, 6, \quad \text{gitt } y_1 = 0 \quad (21)$$

- Bruk *bakoverderivasjon* til manuelt å beregne den numeriske deriverte av u_k , som betyr å finne

$$v_k = \frac{u_k - u_{k-1}}{T_s}, \quad \forall k=2, \dots, 6 \quad \text{gitt } v_1 = 0 \quad (22)$$

Svar

d) Numerisk integrasjon som Matlab-funksjon

I denne oppgaven skal du lage en egen funksjon av Eulers forovermetode for numerisk integrasjon. Du kan også velge å gjøre en frivillig ekstraoppgave hvor du implementerer alle tre metodene for numerisk integrasjon, og spesifiserer i funksjonskallet hvilken metode du vil bruke. For introduksjon til funksjoner i Matlab, skriv `doc function` i Command Window.

- Koden i skallfilen kaller på funksjonen `EulerForover` som vist under

Kode 1: Kodeutdrag som kaller på funksjonen `EulerForover`.

```
y(1) = 0; % initialverdi
for ..
    y(k) = EulerForover(.. , .. , ..);
end
```

Med utgangspunkt i Eulers forovermetode gitt som

$$y_k = y_{k-1} + T_s \cdot u_{k-1} \quad (23)$$

fullfør koden i skallfilen til funksjonen `EulerForover.m`, gjengitt under.

Kode 2: Funksjonen/filen `EulerForover.m`.

```
function IntValueNew = EulerForover(IntValueOld, Timestep, FunctionValue)
% fyll inn
end
```

Som du ser benytter funksjonen variabelnavn som er matematisk beskrivende, samtidig som at den ikke skal benytte indeks `k`.

- Kjør koden og vis at `y(k)` gir samme resultat som `y_EulerF(k)` i oppgave b).

Generell funksjon for numerisk integrasjon (frivillig)

For å ha en mer fleksibel integrasjonsrutine skal vi lage en ny funksjon hvor vi inkluderer et argument som sier noe om hvilken metode som skal brukes. Et utgangspunkt for dette er funksjonen `Integrasjon` vist i kode 3, og denne finner du igjen i skallfilen `Integrasjon.m`.

Kode 3: Den utvidede funksjonen/filen `Integrasjon.m`.

```
function IntValueNew = Integrasjon(IntValueOld, Timestep, FunctionValues, options)

arguments
    IntValueOld (1,1) double           % spesifiserer som skalar, 1x1
    Timestep (1,1) double              % spesifiserer som skalar, 1x1
    FunctionValues (1,2) double        % spesifiserer som vektor, 1x2
    options.metode (1,:) char = 'Trapez' % metodevalg, default er Trapez
end

if strcmp(options.metode, 'EulerForover')
    % fyll inn
elseif strcmp(options.metode, 'EulerBakover')
    % fyll inn
elseif strcmp(options.metode, 'Trapez')
    % fyll inn
else
    error('Feil metode spesifisert')
    return
end
end
```

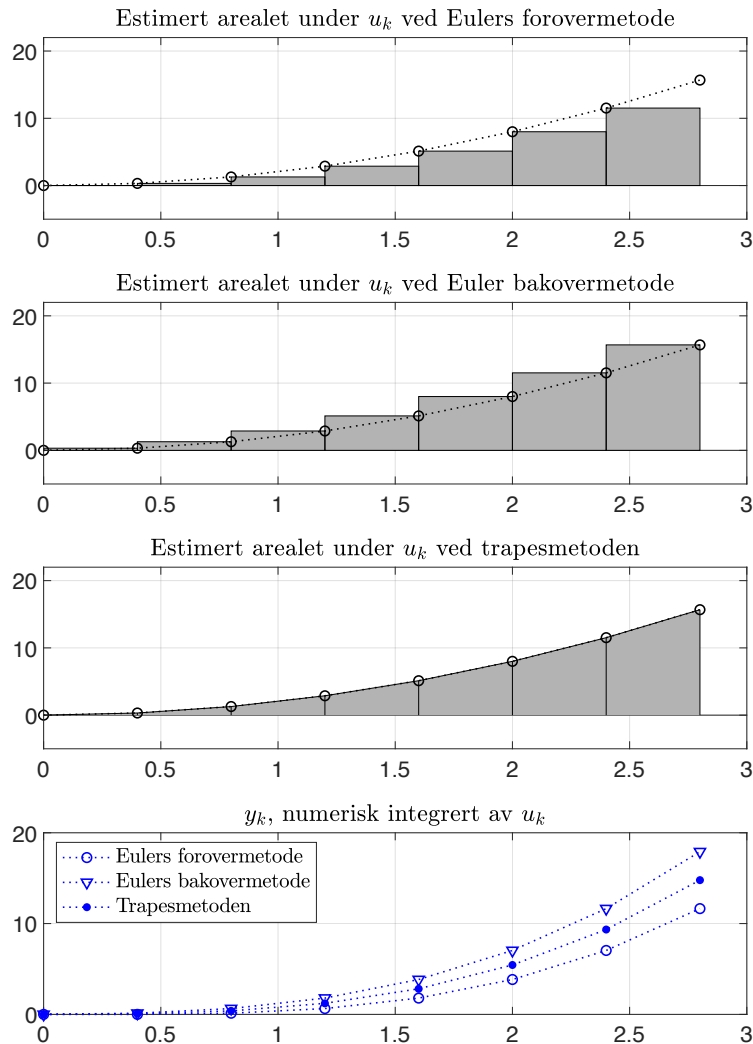
For å kalle på denne funksjonen benyttes syntaksen vist i kode 4 hvor du ser at de to siste elementene `u(k-1:k)` sendes inn, uansett metodevalg. Du må derfor passe å plukke ut riktig element for hver metode i funksjonen.

Kode 4: Alternativ bruk av funksjonen `Integrasjon`.

```
y(k) = Integrasjon(y(k-1), T_s, u(k-1:k), metode='EulerForover');
y(k) = Integrasjon(y(k-1), T_s, u(k-1:k), metode='EulerBakover');
y(k) = Integrasjon(y(k-1), T_s, u(k-1:k), metode='Trapez');
y(k) = Integrasjon(y(k-1), T_s, u(k-1:k)); % default: Trapez
```

Vis at ved å kjøre koden så får du resultatet vist i figur 9.

- Lek litt med å øke og/eller redusere på steglengden `T_s` i linje 5, og studer effekten på integralene.

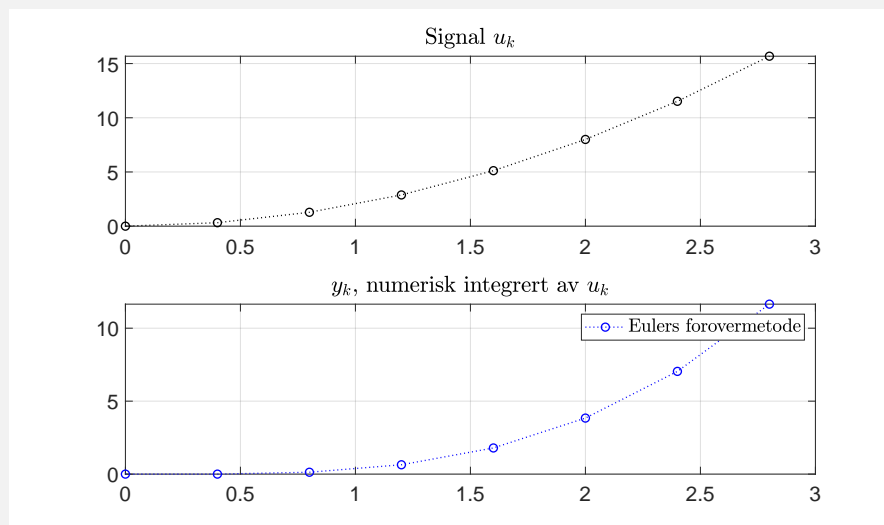


Figur 9: Presentasjon av arealet under kurven u_k for de tre integrasjonsmetodene ved å bruke henholdsvis `bar` og `area`-kommandoene.

Svar

Kode 5: Kode for Eulers Forovermetode.

```
1 function IntValueNew = EulersForover(IntValueOld, Timestep, FunctionValue)
2   IntValueNew = IntValueOld + Timestep * FunctionValue;
3 end
```



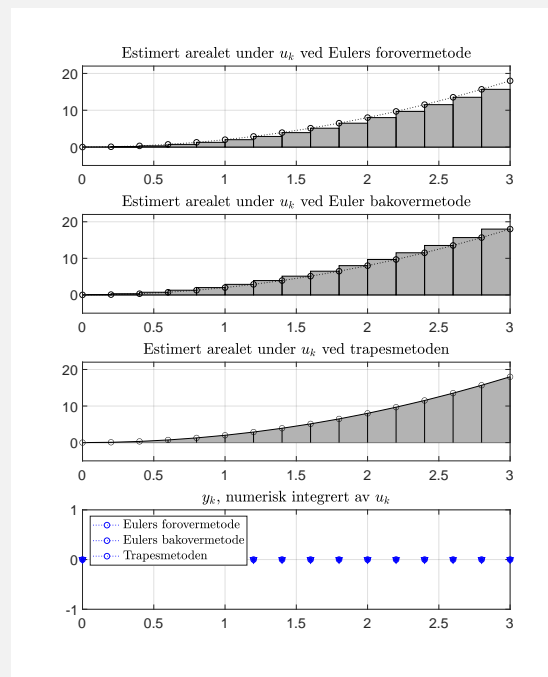
Figur 10: Egen funksjon for Eulers forovermetode gir samme resultat som i oppgave b).

Kode 6: Kode for funksjon for integrasjonsmetoder.

```

1  if strcmp(options.metode, 'EulersForover')
2      IntValueNew = IntValueOld + Timestep * FunctionValues(2);
3
4  elseif strcmp(options.metode, 'EulersBakover')
5      IntValueNew = IntValueOld + Timestep * FunctionValues(1);
6
7  elseif strcmp(options.metode, 'Trapez')
8      IntValueNew = IntValueOld + Timestep * 0.5 * (FunctionValues(1) + ...
          FunctionValues(2));

```



Figur 11: Result med bruk av egendefinert funksjon for integrasjonsmetoder gitt av kodeutdrag 6. $T_s = 0.2s$.

e) **Numerisk derivasjon som Matlab-funksjon**

I denne oppgaven skal du lage en egen funksjon av bakoverderivasjon. På samme måte som i deloppgave d) kan du velge å gjøre den frivillige ekstraoppgaven hvor du implementerer alle tre metodene og spesifiserer i funksjonskallet hvilken metode du vil bruke.

- Koden i skallfilen kaller på funksjonen `BakoverDerivasjon` som vist under

Kode 7: Kodeutdrag som kaller på funksjonen `BakoverDerivasjon`.

```
v(1) = 0; %initialverdi
for ..
    v(k) = BakoverDerivasjon( .., .. );
end
```

Med utgangspunkt i bakoverderivasjon gitt som

$$v_k = \frac{u_k - u_{k-1}}{T_s} \quad (24)$$

fullfør koden i skallfilen til funksjonen `BakoverDerivasjon.m`, gjengitt under.

Kode 8: Funksjonen/filen `BakoverDerivasjon.m`.

```
function Sekant = BakoverDerivasjon(FunctionValues, Timestep)
% fyll inn
end
```

Som du ser benytter funksjonen variabelnavn som er matematisk beskrivende, og den skal formuleres uten bruk av indeks `k`.

- Kjør koden og vis at `v(k)` gir samme resultat som `v_bakover(k)` i oppgave b).

Generell funksjon for numerisk derivasjon (frivillig)

For å ha en mer fleksibel derivasjonsrutine skal vi lage en ny funksjon hvor vi inkluderer et argument som sier noe hvilken metode som skal brukes. Et utgangspunkt for dette er funksjonen `Derivasjon` vist i kode 9, og denne finner du igjen i skallfilen `Derivasjon.m`.

Kode 9: Utvidet funksjon for numerisk derivasjon.

```
function Sekant = Derivasjon(FunctionValues, Timestep, options)
arguments
    FunctionValues (1,3) double           % spesifiserer som vektor, 1x3
    Timestep (1,1) double                 % spesifiserer som skalar, 1x1
    options.metode (1,:) char = 'Bakover' % metodevalg, default er Bakover
end

if strcmp(options.metode, 'Forover')
    % fyll inn

elseif strcmp(options.metode, 'Bakover')
    % fyll inn

elseif strcmp(options.metode, 'Senter')
    % fyll inn

else
    error('Feil metode spesifisert')
    return
end

end
```

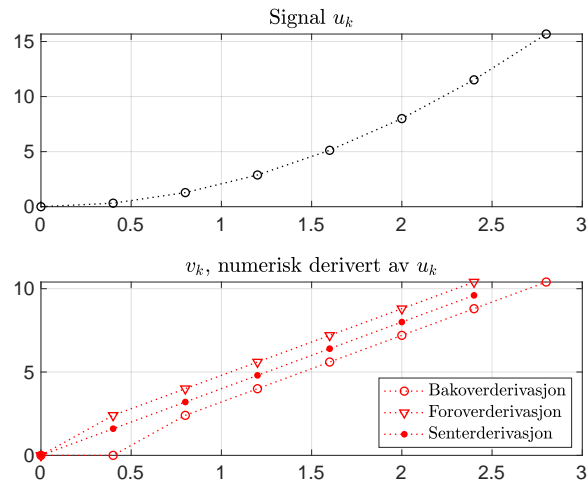
For å kalle denne funksjonen benyttes syntaksen vist i kode 10 hvor du ser at de tre siste elementene `u(k-2:k)` sendes inn, uansett metodevalg. Du må derfor passe å plukke ut riktig element for hver metode i funksjonen.

Kode 10: Alternativ bruk av funksjonen `Derivasjon`.

```
v(k) = Derivasjon(u(k-2:k), T_s, metode='Bakover');
v(k-1) = Derivasjon(u(k-2:k), T_s, metode='Forover');
v(k-1) = Derivasjon(u(k-2:k), T_s, metode='Senter');
```

For å sende inn tre verdier av `u` så kan ikke funksjonen kalles på før $k=3$. Det betyr også at noe må gjøres med initialverdiene, og dette er vist i skallfilen.

Vis at ved å kjøre koden så får du resultatet vist i figur 12.



Figur 12: Resultat av numerisk derivasjon ved bruk av funksjonen Derivasjon.

Svar

Relevant kode er vist i kodeutdrag 11 og 12.

Kode 11: Kode fra oving3.m for å kalle egendefinert funksjon for derivasjon.

```

1 % initialverdi
2 v1(1) = 0;      % Bakoverderivasjon
3 v2(1) = 0;      % Foroverderivasjon
4 v3(1) = 0;      % Senterderivasjon
5
6 for k = 3:length(t)
7     v1(k) = Derivasjon(u(k-2:k), T_s, metode='Bakover');
8     v2(k-1) = Derivasjon(u(k-2:k), T_s, metode='Forover');
9     v3(k-1) = Derivasjon(u(k-2:k), T_s, metode='Senter');
10 end

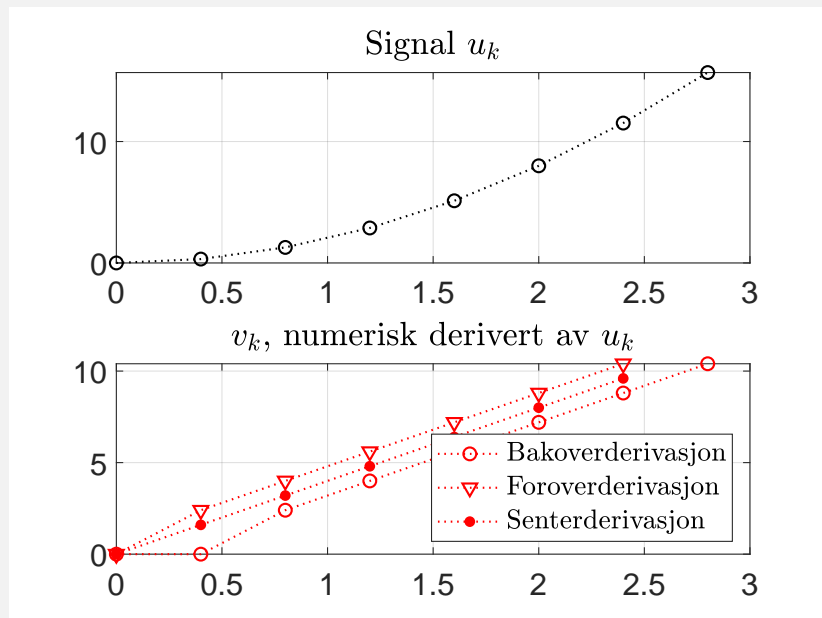
```

Kode 12: Kode fra Derivasjon.m som viser egendefinert funksjon for derivasjon.

```

1 if strcmp(options.metode, 'Bakover')
2     Sekant = (FunctionValues(3) - FunctionValues(2))/Timestep;
3
4 elseif strcmp(options.metode, 'Forover')
5     Sekant = (FunctionValues(3) - FunctionValues(2))/Timestep;
6
7 elseif strcmp(options.metode, 'Senter')
8     Sekant = (FunctionValues(3) - FunctionValues(1))/(2*Timestep);

```



Figur 13: Resultat av derivasjon.

Viktige egenskaper ved numerisk integrasjon og derivasjon

I de neste to deloppgavene f)-g) skal du jobbe med følgende enkle signal

$$u(t) = 1 \quad (25)$$

Hensikten med å bruke et såpass enkelt signal er å demonstrere noen viktige egenskaper ved numerisk integrasjon og derivasjon.

f) Numerisk integrasjon av et signal som er forsterket

I denne oppgaven skal du beregne følgende integral

$$y(t) = K_i \int_0^t u(t) dt \quad (26)$$

ved å bruke funksjonen `EulerForover` fra deloppgave d). Parameteren K_i er en forsterkningsfaktor som kan være et vilkårlig tall, positivt eller negativt. I koden nedenfor er $K_i=0.4$, og det er implementert to forskjellige varianter av numerisk integrasjon. Den ene varianten er riktig, den andre er feil, og målet med oppgaven er at du skal finne ut hvilken som er riktig, og forklare hvorfor det er slik.

Kode 13: Kode for oppgaven.

```
% initialverdi
y1(1) = 0;
y2(1) = 0;

Ki = 0.4;
for k = 1:100
    y1(k) = EulerForover(y1(k-1), T_s, Ki*u(k-1));
    y2(k) = Ki*EulerForover(y2(k-1), T_s, u(k-1));
end
```

- Pass på at `Ki = 0.4;` og kjør koden. Kan du allerede nå ut fra kurvene finne ut om det er `y_1` eller `y_2` som er riktig?
- Endre parameteren K_i til `Ki = 1.4;` og kjør koden på ny. Blir du noe klokere av det?
- For å bli sikker på svaret er det smart å legge inn følgende kode i linjen etter at `u` blir definert.

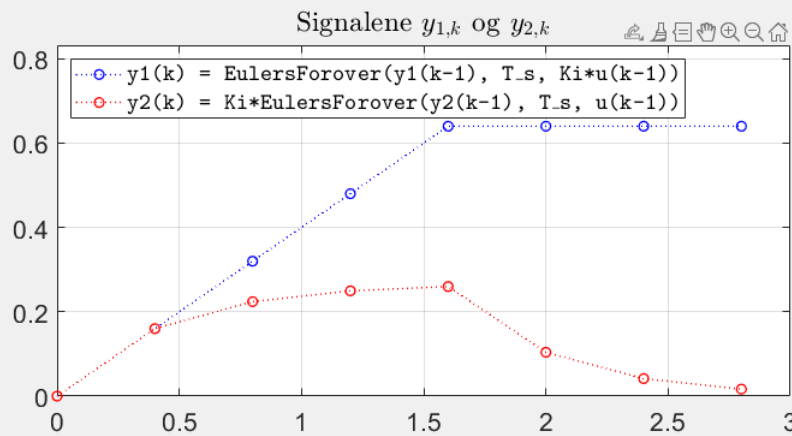
```
u(5:end) = 0; % setter siste del av u som 0
```

Du kan nå endre K_i mellom `Ki = 0.4;` og `Ki = 1.4;`, og du vil forhåpentligvis oppdage hvilken som er den riktige.

- Hva er den egentlige grunnen til at den som er feil, er feil.

Svar

- Rent matematisk skulle man jo tro at y_2 er riktig, fordi den opprinnelige ligningen i (26) har K_i utenfor integralet. I y_2 multipliseres forsterkningsfaktoren med resultatet av hele integralet. y_1 gir integralet av et forsterket signal. y_2 gir et forsterket integral av det originale signalet.
- Pga. Eulers forovermetode bruker startverdien av et intervall og multipliserer den med intervallbredden får vi to ulike resultat. y_1 skalerer funksjonsverdien, og det er denne som multipliseres med intervallbredden. y_2 finner "arealet under kurven", og skalerer deretter arealet.



Figur 14: Resultat av integrasjon.

g) Effekten av støy ved numerisk derivasjon

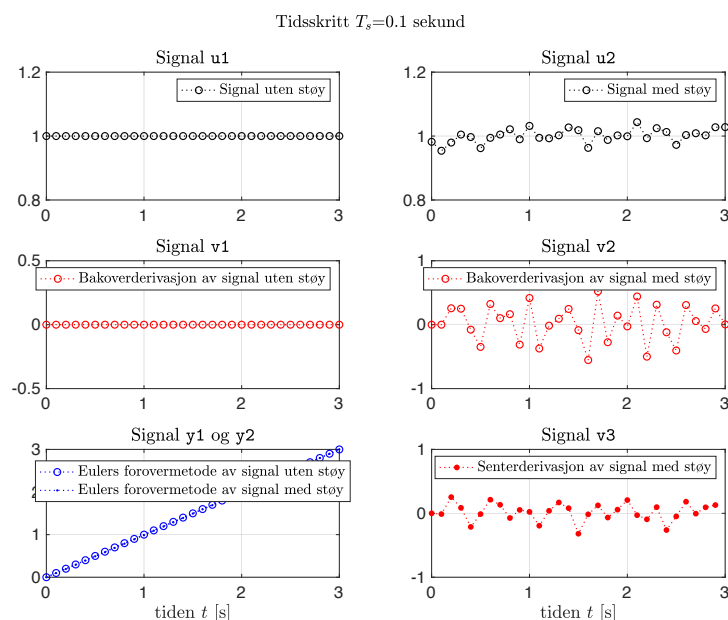
For å gjøre denne oppgaven må du gjort de frivillige oppgavene i d) og e).

Hensikten med oppgaven er å studere effekten av støy ved numerisk integrasjon og derivasjon. Vi tar utgangspunkt i signalet fra ligning (25) hvor vi lager et signal uten støy som vi kaller $u1$, og et signal med støy som vi kaller $u2$.

Vi benytter deretter Eulers forovermetode på begge signalene for å beregne henholdsvis $y1$ og $y2$.

I tillegg benytter vi bakoverderivasjon på begge signalene for å beregne henholdsvis $v1$ og $v2$. For å studere effekten av metodevalg for derivasjon, benytter vi senterderivasjon på signalet med støy og beregner $v3$.

- Pass på at $T_s = 0.4$ og kjør koden. Endre tidsskrittet i steg fra $T_s = 0.4$ til $T_s = 0.05$. Vi at du får et resultatet som ligner på figur 15. Ta med figuren din i innleveringen.



Figur 15: Resultat som viser hvordan støy påvirker numerisk integrasjon og derivasjon.

- Hva skjer med signalene $v2$ og $v3$ etterhvert som tidsskrittet synker? Hva er forklaringen på at utslaget i disse to blir større og større?
- Forklar hvorfor støy ikke påvirker numerisk integrasjon i særlig grad.
- Hva er grunnen til at senterderivasjon gir bedre resultat sammenlignet med bakoverderivasjon?

Svar

Effekten av samplingsfrekvensen f_s ved måling

h) Fenomenet nedfolding

I denne oppgaven tar vi utgangspunkt i hvordan sampling blir utført som vist i kapittel 7.2 i kompendiet. Hensikten med oppgaven er å demonstrere hvilken betydningen samplingsfrekvensen f_s [Hz] har ved sampling av signaler som består av forskjellige frekvenser f [Hz]. Vi skal derfor bruke følgende 5 cosinussignal

$$x_1(t) = \cos(2\pi f_1 \cdot t) \quad (27)$$

$$x_2(t) = \cos(2\pi f_2 \cdot t) \quad (28)$$

$$x_3(t) = \cos(2\pi f_3 \cdot t) \quad (29)$$

$$x_4(t) = \cos(2\pi f_4 \cdot t) \quad (30)$$

$$x_5(t) = \cos(2\pi f_5 \cdot t) \quad (31)$$

hvor vi benytter følgende frekvenser

$$f_1 = 0.1, f_2 = 0.3, f_3 = 0.5, f_4 = 0.9 \text{ og } f_5 = 1.3 \text{ Hz}$$

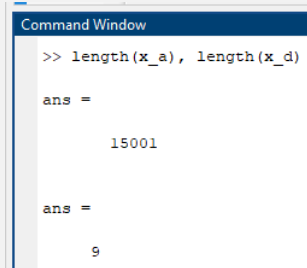
Som du ser i skallfilen lager vi først den “kontinuerlige” tidsvektoren `tid` ved å benytte et veldig kort tidsskritt `delta_t = 0.001`.

Deretter lages vektoren `t` med samplingstidspunkt som representerer de tidspunktene hvor målingene blir tatt. Basert på tidsvektorene `tid` og `t` lages det deretter en *analog* og en *diskret* versjon av alle fem signalene `x_1` til `x_5`. Til slutt summeres alle de analoge signalene til signalet `x_a` og alle de diskrete signalene til signalet `x_d`. På denne måten kan du se effekten av samplingsfrekvensen `f_s` på et sammensatt og mer reelt signal (ikke bare et rent cosinussignal).

- Hvor stor er samplingsperioden T_s for den valgte samplingsfrekvensen $f_s=0.6$ Hz?
- Hvor mange elementer er det i signalet `x_a` og i signalet `x_d`?
- Forklar hva figuren viser?
- Hvilke signal ser det ut som vi sampler i `subplot(7,1,2)` til `subplot(7,1,5)`? Avles periodetidene T_p fra de blå stiplede kurvene og beregn de tilsynelatende frekvensene for de diskrete signalene. Sammenlign med frekvensene for de analoge signalene $x_2(t)$ til $x_5(t)$.
- Hva skjer dersom du spesifiserer samplingsfrekvensen som $f_s=1$ Hz?
- Hva skjer dersom du spesifiserer samplingsfrekvensen som $f_s=1.8$ Hz?
- Hva skjer dersom du spesifiserer samplingsfrekvensen som $f_s=2.6$ Hz?
- Forklar hva som skjer når samplingsfrekvensen øker.

Svar

- Samplingsperioden $T_s \approx 1.67\text{s}$
- Antall elementer i x_a : 15001
- Antall elementer i x_d : 9



```

Command Window
>> length(x_a), length(x_d)

ans =
    15001

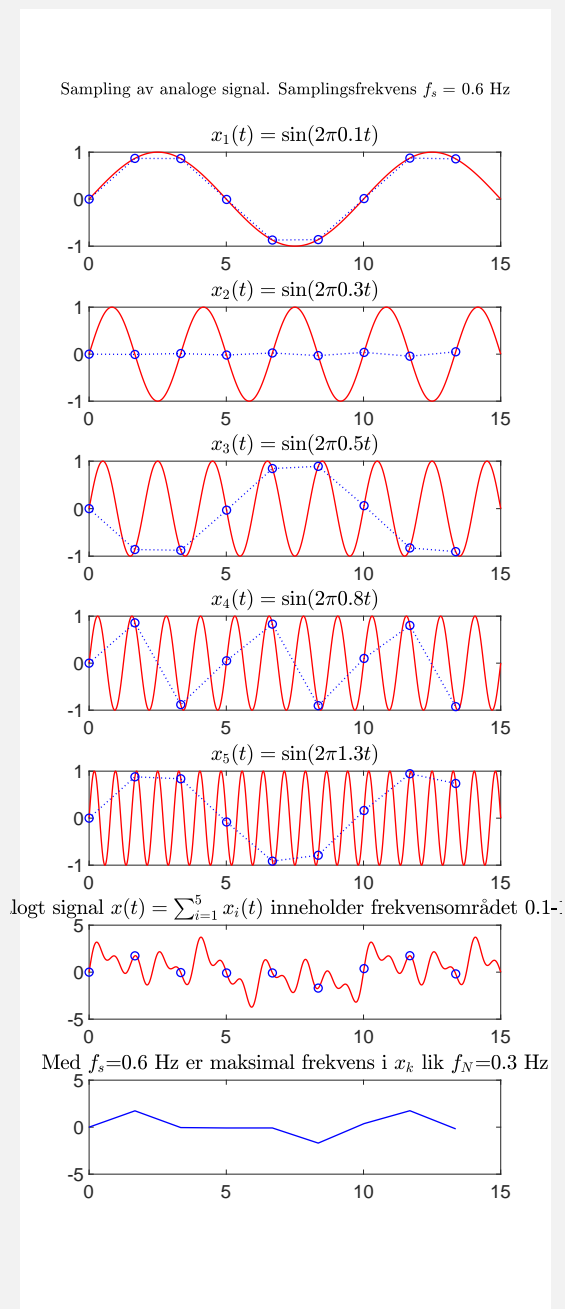
ans =
     9
  
```

Figur 16: Lengde av x_a og x_d .

- Tabell h viser periodetid og vinkelfrekvens for x_1 til x_5 .

	x_1	x_2	x_3	x_4	x_5
T_p	10.02	3.34	10.02	5.01	10.02
ω	0.1	0.3	0.1	0.2	0.1

- Figur 17 viser hvordan samplingfrekvens påvirker hvordan vi oppfatter et signal. Dårlig valg av samplingfrekvens kan gi et uriktig bilde av det faktiske, underliggende signalet.



Figur 17: Samplet, diskret signal vs underliggende, analogt signal.

- Den høyeste frekvensen som kan gjengis nøyaktig i det diskrete signalet

kalles Nyquist-frekvensen, f_N , og den er halvparten av samplingfrekvensen, $f_s/2$.

- Høyere frekvens gir flere verdipunkter. Ved $f_s=1$ Hz sees spesielt forbedret sporing av det analoge signalet i x_2 . Samplingen av x_3 er nå synkronisert med det analoge signalet.
- $f_s=1.8$ Hz gir stor forbedring i x_3 , men også synlig bedring i x_4 . x_1 er nå helt i tråd med underliggende signal, x_2 er ikke langt unna. x_5 har fortsatt en del å gå på. x_t ligger på.
- $f_s=2.6$ Hz. x_4 begynner å ligne på seg selv. x_5 er nå synkronisert, da den ligger på 1.3 Hz, nøyaktig halvparten av samplingfrekvensen.
- Når samplingsfrekvensen øker, blir tidsintervallet mellom hver prøvepunkt mindre. Dette betyr at vi tar flere prøver av det analoge signalet per sekund. Økningen i samplingsfrekvensen fører blant annet til økt oppløsning, redusert risiko for nedfolding, men høyere ressurskrav (databehandling, lagring, osv.).

Filtrering

i) FIR-filter

I Lego-prosjektet skal du jobbe med FIR-filter (midlingsfilter) gitt som

$$y_k = \frac{1}{M} \sum_{n=0}^{M-1} x_{k-n} \quad (32)$$

hvor M er antall målinger av x som skal være med i midlingen. I denne oppgaven skal du først implementere kode for filteret i ligning (32) og deretter skal du lage en egen funksjon for FIR-filteret i skallfilen `FIR_filter.m`.

Ta utgangspunkt i skallfilen hvor du ser at den diskret tidsvektoren `t` er basert på en antatt samplingsfrekvens på

$$f_s = 10 \text{ Hz} \quad (33)$$

som gir et tidsskritt på

$$T_s = \frac{1}{f_s} = 0.1 \text{ sekund} \quad (34)$$

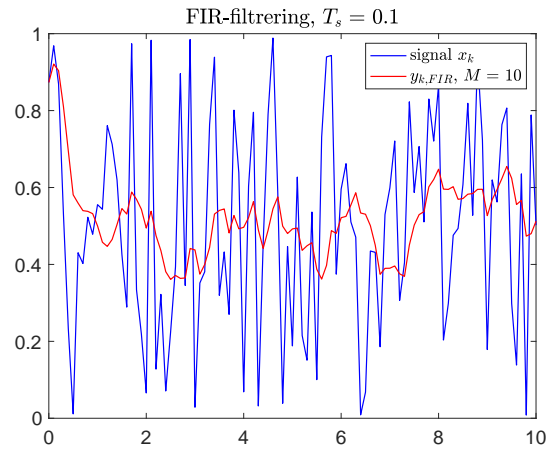
Med en slutt-tid på 10 sekund betyr det at vi har 100 målinger tilgjengelig, og målesignalet `x` er som du ser bare støy. Gjør følgende oppgaver:

- Implementer kode for FIR-filteret i ligning (32) i *for*-løkken. Vær nøye med indeksene.

Der hvor det står `% juster på M her` så gjelder dette for situasjonen hvor $k < M$ hvor det ikke er nok målinger til å bruke $M = 10$. En måte å løse det på er å utsette filtreringen til $k > M$, men en bedre løsning er justere på M slik at filtreringen kan starte med en gang. Hint: Hva skulle du ønske at M var i starten av filtreringen, og så lenge $k < M$?

Vær klar over at siden du gjør justeringer på M inne i *for*-løkka når $k < M$, så må koden `M = 10;` stå inne i den samme løkka. Setter du koden `M = 10;` foran *for*-løkka så vil ikke verdien av M bli resatt for hver runde i løkka. Test det gjerne ut.

Ferdigstill og kjør koden og vis at du får resultatet vist i figur 18.



Figur 18: Respons i FIR-filter med $M=10$.

- Innholdet i skallfilen `FIR_filter.m` er vist under, og som du ser benyttes beskrivende matematiske variabelnavn i funksjonen.

Kode 14: Skall for funksjonen for FIR-filtrering.

```
function FilteredValue = FIR_filter(Measurement, NoOfMeas)
% juster på NoOfMeas her
% fyll inn filterkode
end
```

Ferdigstill koden i denne funksjonen. Legg merke til at vi flytter koden som justerer på M inn i funksjonen. På den måten blir mer funksjonen mer funksjonell i bruk, og M kan spesifiseres før *for*-løkken i hovedfilen.

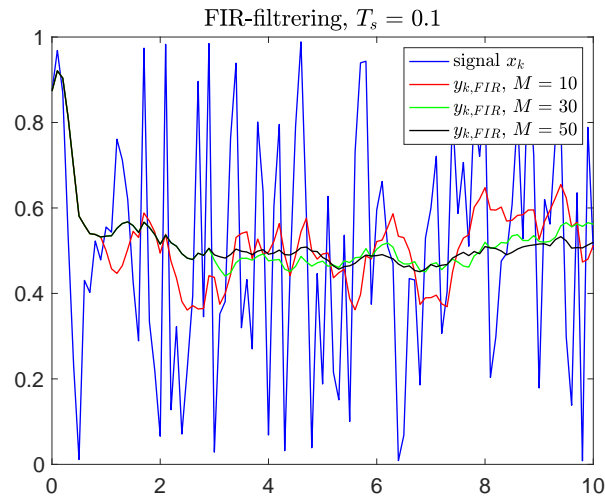
- Vis at du får samme resultat som i figur 18 ved å endre *for*-løkken i skallfilen til

Kode 15: Syntaks for å kalle funksjonen `FIR_filter`.

```
M = 10;
for k = 1:NoOfMeas
    y_FIR(k) = FIR_filter(Measurement(k,:), M);
end
```

Effekten av M

- For å teste betydningen av verdien på M så er det i skallfilen `oving3_skallfil.m` laget til en egen seksjon som du kan kjøre direkte når du har laget ferdig funksjonen `FIR_filter`. Vis at du får resultatet vist i figur 19.



Figur 19: Respons i FIR-filter med $M=[10, 30, 50]$.

- Hva er grunnen til at alle de tre filtrerte kurvene ligger over hverandre frem til ca 1 sekund? Samme fenomen ser du gjelder for to av kurvene mellom 1 og 3 sekund.
- Er filtreringen som oppnås fornuftig i forhold til stigende tall på M ? Begrunn svaret.
- Lek gjerne med andre verdier på M . Hva skjer hvis du setter $M = k$; inne i *for*-løkken?

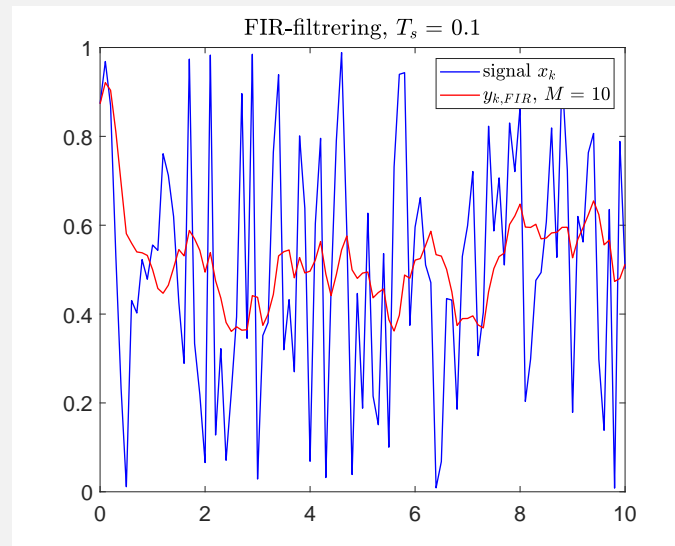
Svar

Kode 16: Kode for IR-filter initialisering og *for*-løkke.

```

1 % initialisering
2 y_FIR(1) = x(1);
3
4 for k = 1:length(t)
5     M = 10;
6     if k < M
7         M = k;
8     end
9     y_FIR(k) = 1/M * sum(x(k-M+1:k));
10 end

```



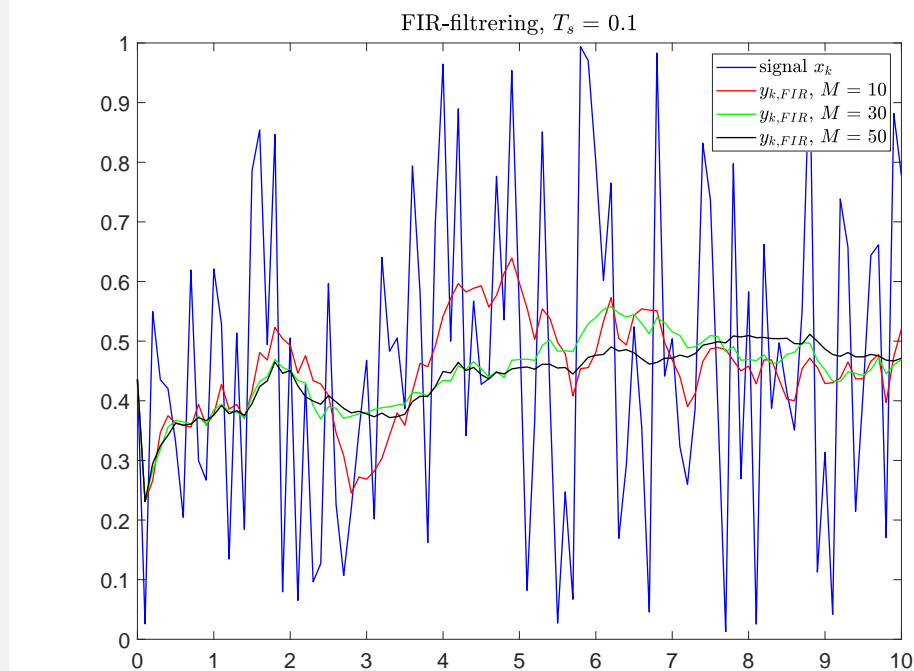
Figur 20: Resultat av FIR-filter.

Kode 17: Kode for FIR-filter egendefinert funksjon

```

1 function [FilteredValue] = FIR_filter(Measurement,NoMeas)
2
3 % Persist recent measurements in a list
4 persistent Measurements
5
6 % Initialize Measurements if it's empty
7 if isempty(Measurements)
8     Measurements = Measurement;
9 else
10    Measurements = [Measurements, Measurement]; % Append the new measurement
11 end
12
13 % Adjust NoOfMeasurements if there are not enough measurements
14 if length(Measurements) < NoMeas
15     NoMeas = length(Measurements);
16 end
17
18 % Compute the filtered value as the average of the last NoOfMeasurements ...
19     measurements
20 start_index = max(1, length(Measurements)-NoMeas+1);
21 FilteredValue = sum(Measurements(start_index:end)) / NoMeas;
22 end

```



Figur 21: Resultat av egendefinert funksjon for FIR-filter.

- Så lenge antall målinger ikke overgår M vil Moving Averages være like, i.e., $M=10$, 30, eller 50 vil være like til og med 10 målinger. $M=30$ og $M=50$ vil være like til og med 30 målinger, osv.
- Det ser fornuftig ut. Høyere M gir jevnere kurve, som forventet. Økning i antall verdier gjør at nye verdier har redusert påvirkning og gir dermed mindre utslag.
- Dersom M settes lik k inne i for-løkken vil hver filtrerte verdi være gjennomsnittet av alle verdiene frem til det punktet.

j) **IIR-filter**

I Lego-prosjektet skal du også jobbe med første ordens IIR-filter gitt som

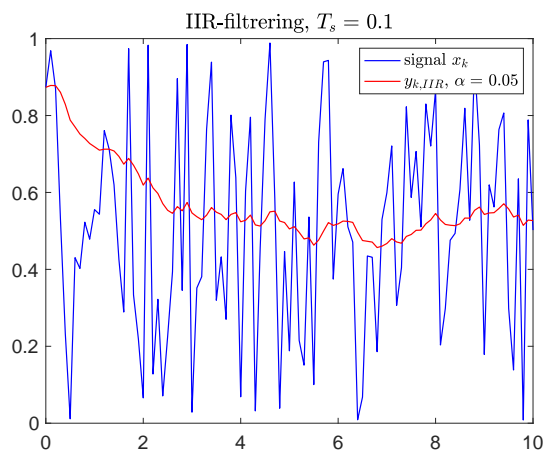
$$y_k = (1 - \alpha) \cdot y_{k-1} + \alpha \cdot x_k \quad (35)$$

hvor $0 \leq \alpha \leq 1$. Dette filteret bruker kun siste måling x_k som vektes med verdien α , og legger sammen med forrige filtrerte verdi y_{k-1} som vektes med $(1 - \alpha)$. Dersom $\alpha = 1$ så har du ingen filtrering og dersom $\alpha = 0$ så får du en rett strek (alt filtreres).

På samme måte som i deloppgave i) skal du også i denne oppgaven først implementere kode for filteret gitt i ligning (35) og deretter lage en egen funksjon av IIR-filteret i skallfilen `IIR_filter.m`.

Ta utgangspunkt i koden i skallfilen hvor du ser at målesignalet `x` er det samme som i forrige deloppgave, det samme gjelder samplingsfrekvensen og slutt-tiden. Gjør følgende oppgaver:

- Implementer kode for IIR-filteret i ligning (35) i *for*-løkken. Kjør koden og vis at du får resultatet vist i figur 22.



Figur 22: Respons i IIR-filter med $\alpha = 0.05$.

- Innholdet i skallfilen `IIR_filter.m` er vist under, og igjen brukes det beskrivende matematiske begrep for variabelnavn.

Kode 18: Skall for funksjonen for IIR-filtrering.

```
function FilteredValue = IIR_filter(OldFilteredValue, Measurement, para)
% fyll inn
end
```

Ferdigstill koden i denne funksjonen.

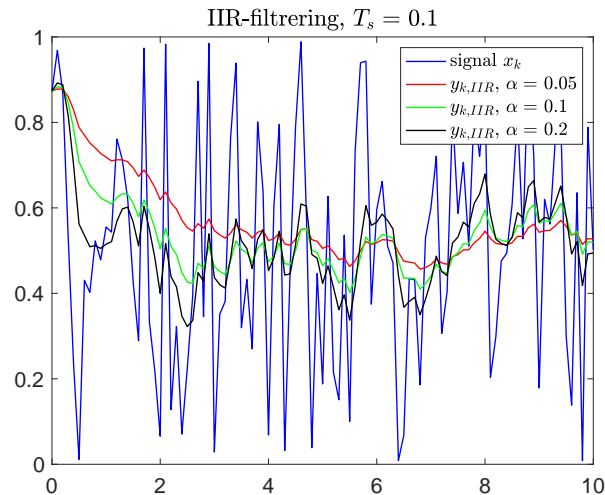
- Vis at du får samme resultat som i figur 22 ved å endre *for*-løkken i skallfilen til

Kode 19: Skall for funksjonen for IIR-filtrering.

```
for k = 2:length(t)
    y_IIR(k) = IIR_filter(..., ..., ...)
end
```

Effekten av α

- For å teste betydningen av α så er det i skallfilen `oving3_skallfil.m` laget til en egen seksjon som du kan kjøre direkte når du har laget ferdig funksjonen `IIR_filter`. Vis at du får resultatet vist i figur 23.



Figur 23: Respons i IIR-filter med $\alpha=[0.05, 0.1, 0.2]$.

- Er filtreringen som oppnås fornuftig i forhold til synkende tall på α ?
- Test ut $\alpha = 0$ eller $\alpha = 1$. Er resultatet fornuftig?
- Hva skjer dersom du setter $\alpha = -0.1$, $\alpha = 1.7$, $\alpha = 2$ eller $\alpha = 2.3$? Kan du forklare hva som skjer?

Svar

Kode 20: Kode for IIR-filter for-løkke

```
1 % initialisering
```

```

2 y_IIR(1) = x(1);
3 alfa = 0.05;
4 for k = 2:length(t)
5     %y_IIR(k) = (1-alfa) * y_IIR(k-1) + alfa * x(k);
6     y_IIR(k) = IIR_filter(y_IIR(k-1),x(k),alfa);
7 end

```

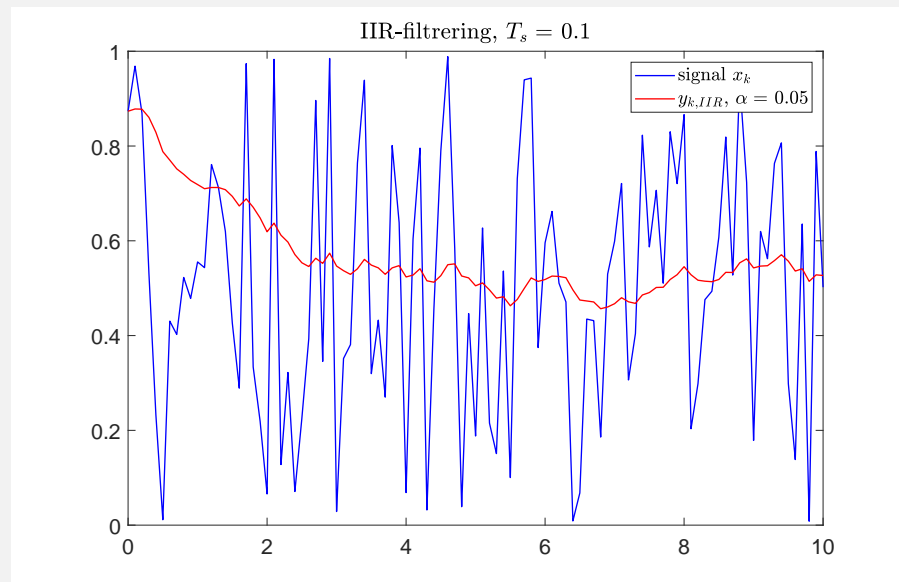
Kode 21: Kode for egendefinert funksjon for IIR-filter.

```

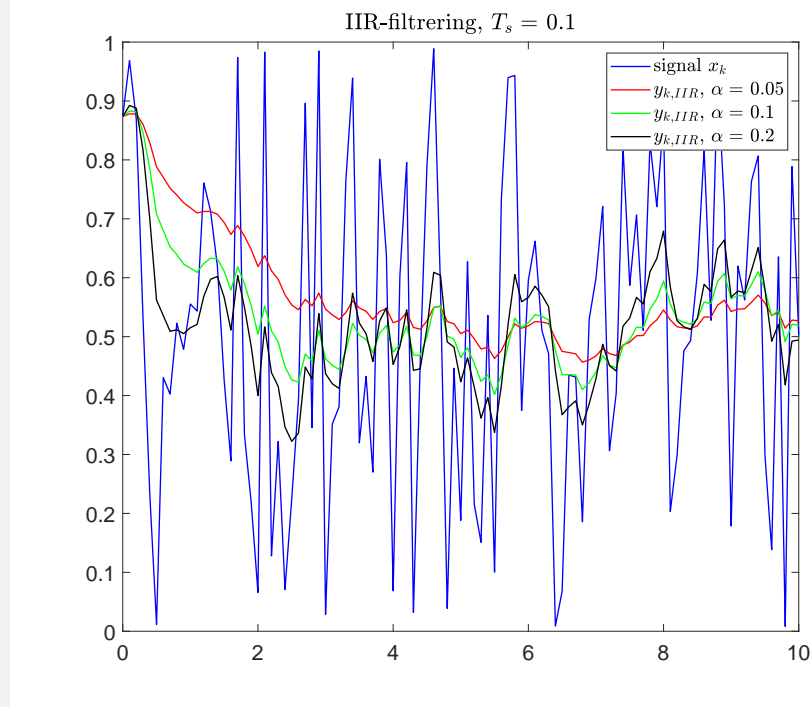
1 function [FilteredValue] = IIR_filter(OldFilteredValue, Measurement, Parameter)
2 FilteredValue = (1-Parameter) * OldFilteredValue + Parameter * Measurement;
3 end

```

Begge kodesnuttene over gir samme figur, som vist i 24.



Figur 24: Resultat av IIR-filter med $\alpha=0.05$.



Figur 25: Resultat av IIR-filter med $\alpha=[0.05, 0.1, 0.2]$.

- Ja.
- Resultatet er fornuftig for begge. Ved $\alpha=1$ er det filtrerte signalet lik inn-gangssignalet. Ved $\alpha=0$ er det filtrerte signalet lik det initiale signalet/i-nitiert verdi.
- Dersom α settes utenfor grensen 0, 1 blir filterligningen ustabil. Er den mellom 0 og 1 sørger den for balanse mellom forrige filtrerte verdi, $y(k-1)$ og signalets nåverdi, $x(k)$.
 - For $\alpha=-0.1$ vil nåverdien bli trukket fra i stedet for å bli lagt til. I tillegg vil den forrige filtrerte verdien skaleres med 110%.
 - For $\alpha=1.7$ og opp blir signalets nåverdi forsterket i stedet for dempet, samt at forrige filtrerte verdi trekkes fra i stedet for å legges til.

k) Filtrering av ulike testsignal

I denne oppgaven skal du bruke filterfunksjonene fra de forrige deloppgavene til å filtrere følgende fire testsignal (formulert tidskontinuerlig):

$$\text{impuls: } x_1(t) = \begin{cases} 0 & \text{for } t < 10 \\ 1 & \text{for } t = 10 \\ 0 & \text{for } t > 10 \end{cases} \quad (36)$$

$$\text{firkantpuls: } x_2(t) = \begin{cases} 0 & \text{for } t < 10 \\ 1 & \text{for } 10 \leq t \leq 20 \\ 0 & \text{for } t > 20 \end{cases} \quad (37)$$

$$\text{sinussignal: } x_3(t) = 0.1 \sin(2\pi 0.1t) + 0.2 \quad (38)$$

$$\text{sinussignal med støy: } x_4(t) = 0.1 \sin(2\pi 0.1t) + 0.2 + w(t) \quad (39)$$

I skallfilen er den diskret tidsvektoren `t` basert på en samplingsfrekvens på

$$f_s = 2 \text{ Hz} \quad (40)$$

som gir en samplingstid på $T_s=0.5$ sekund. Siden Nyquistfrekvensen ved denne samplingsfrekvensen er

$$f_N = 0.5 \cdot f_s = 1 \text{ Hz} \quad (41)$$

så betyr det at vi sampler hurtig nok til å få med oss informasjonen fra sinussignalene gitt i ligningene (38) og (39), siden begge har en svingefrekvensen på $f=0.1$ Hz. Videre er det i skallfilen gitt kode for de fire signalene `x1` til `x4` hvor bruker vi sammenhengen

$$k \cdot T_s = t_k \quad (42)$$

for å bestemme hvilken indeks k som hører til et gitt tidspunkt. Siden vi jobber med Matlabindekser legger vi til 1 i beregningen. Dersom du velger et tids-skritt som ikke gir heltallsverdier av k , så må du bruke `round`-funksjonen som `k = round(t_impuls/T_s+1);`.

Siden koden for FIR- og IIR-filteret er basert på variabelen `x`, og ikke `x1`, `x2`, `x3` eller `x4` som vi nettopp har definert, så velger du hvilket signal du vil bruke der hvor det i skallfilen står:

Kode 22: Syntaks for å velge hvilket signal som skal filtreres.

```
%-----
% Velg hvilket signal som skal filtreres.
x = x1;
% For x1 og x2 er det smart å bruke kurveform = ':x';
% For x3 og x4 er det smart å bruke kurveform = '-';
kurveform = '-';
%-----
```

Ferdigstill det som mangler i skallfilen, og gjør deretter følgende oppgaver, hvor du for hver oppgave tar med resultatfigurene i innleveringen din.

Filtrering av impulsen $x_1(t)$

Hensikten med dette signalet er å demonstrere hvorfor det heter FIR og IIR. Det betyr at du vil se at impulsen gir en respons som blir 0 etter en viss tid for FIR-filteret (*finite*), og en respons som aldri blir 0 for IIR-filteret (*infinite*).

- Bruk `M = 10` og `alfa = 0.3` og kjør koden. Trykk inn på de røde og grønne linjene til hhv. FIR- og IIR-filteret rundt $t \approx 30$ sekund og les av verdien. Ta med figuren i innleveringen din.
- Er det logisk at maksimalverdien av $y_{k,FIR}$ er 0.1 når `M = 10` mens maksimalverdien av $y_{k,IIR}$ er 0.3 når `alfa = 0.3`? Begrunn svaret.

Filtrering av firkantpulsen $x_2(t)$

Hensikten med dette signalet er å demonstrere *dynamikken* til filteret, altså hvor raskt det svinger seg inn til ny stasjonær verdi gitt at inngangen endres som et sprang. Husk at det positive spranget går ved $t_{\text{sprang}}=10$ sekund, og det negative spranget går ved $t_{\text{sprang}}=20$ sekund.

- Bruk `M = 10` og `alfa = 0.3` og kjør koden. Forklar med utgangspunkt i ligningen for FIR-filteret hvorfor responsen stiger som en lineær kurve før den til slutt flater ut.
- Bruk sammenhengen

$$M = \frac{t_{\text{FIR}}}{T_s} + 1 \quad (43)$$

til å anslå filtervinduet lengde t_{FIR} [s]. Hvordan relateres verdien av t_{FIR} til responsen i FIR-filteret?

- Bruk sammenhengen

$$\alpha = \frac{T_s}{\tau + T_s} \quad (44)$$

til å anslå tidskonstanten τ for IIR-filteret. Hvordan relateres verdien av τ til responsen i IIR-filteret?

- Hvordan endres dynamikken (hurtigheten) til IIR-filteret og selve responsen til FIR-filteret dersom du endrer til `M = 15` og `alfa = 0.15`?

Filtrering av sinussignalet $x_3(t)$

Hensikten med dette signalet er å vise hvordan sinussignalet endres gjennom filteret, altså hvor mye amplituden blir redusert og hvor stor faseforskyvning er som funksjon av filterparametrene. I denne oppgaven kan du med fordel endre kurveformen til `'-'` slik at kurvene blir “renere”. Gitt at resposnene i FIR- og IIR-filteret kan formuleres tidskontinuerlig som

$$y_{FIR}(t) = Y_{FIR} \cdot \sin(2\pi ft + \varphi_{FIR}) \quad (45)$$

$$y_{IIR}(t) = Y_{IIR} \cdot \sin(2\pi ft + \varphi_{IIR}) \quad (46)$$

- Bruk `M = 15` og `alfa = 0.15` og kjør koden. Hvor stor er amplitudene Y_{FIR} og Y_{IIR} ?
- Hvor stor er faseforskyvningen mellom innsignalet $x_3(t)$ og de to filtrerte signalene. Det vil si, bestem φ_{FIR}° og φ_{IIR}° . (oppgitt i grader)
- Øk samplingsfrekvensen f_s til `f_s = 4`. Hvordan påvirker dette filtreringen? Hva er forklaringen på det som skjer?
- Behold `f_s = 4`. Hva må du gjøre med M og α for å få et resultat som ligner på det du fikk når `f_s = 2`?
- Sett `M = k` inne i for-løkken, og forklar hva som skjer i FIR-filteret.

Filtrering av sinussignalet med støy $x_4(t)$

Hensikten med dette signalet er å vise at hvordan du kan filtrere bort støyen og helst sitte igjen med sinussignalet som ligger i bunn.

- Behold `f_s = 4` og finn de M og α -verdiene som du synes fjerner støy på best mulig måte samtidig som selve sinussignalet beholdes i størst mulig grad. Her er det ingen feil svar.

Svar

1) IIR-basert lavpass- og høypassfiltrering av *Chirp*-signal

I denne oppgaven skal vi ta utgangspunkt i sinussignalet fra ligning (38) skrevet med tidsvarierende frekvens $f(t)$ som

$$x(t) = 0.1 \cdot \sin(2\pi f(t) \cdot t) + 0.2 \quad (47)$$

Vi skal lage et *chirp*-signal $x(t)$ som går fra f_{initial} [Hz] til f_{target} [Hz] i løpet av 100 sekund. Den tidsavhengige frekvensen $f(t)$ kan formuleres som

$$f(t) = f_{\text{initial}} + \left(\frac{f_{\text{target}} - f_{\text{initial}}}{t_{\text{target}}} \right) \cdot t \quad (48)$$

hvor

- f_{initial} [Hz] tilsvarer startfrekvensen
- f_{target} [Hz] tilsvarer slutfrekvensen
- t_{target} [s] er tidspunktet når frekvensen er f_{target}

Du skal bruke to forskjellige varianter av første ordens IIR-filter, nemlig

$$y_k = (1 - \alpha) \cdot y_{k-1} + \alpha \cdot x_k \quad (49)$$

$$y_k = a_1 \cdot y_{k-1} + b_0 \cdot x_k + b_1 \cdot x_{k-1} \quad (50)$$

Du kjenner godt til den første varianten i ligning (49) som egentlig er et *lavpassfilter*. Det betyr at lave frekvenser passerer gjennom, mens høye frekvenser stoppes i filteret.

Filteret i ligning (50) er et filter som kan fjerne DC-ledd (konstantledd), og er dermed et såkalt *høypassfilter*. Lave frekvenser, som egentlig kan betraktes som et langsomt-varierende konstantledd fjernes og høye frekvenser slippes gjennom. Ved å velge like verdier for b_0 og b_1 , men med motsatt fortegn, så filtreres faktisk DC-leddet helt bort! Ferdigstill og kjør koden i skallfilen. Ta med figuren i innleveringen din.

Svar

m) Kodebasert filtrering i MATLAB Function-blokk i Simulink

Sinusfunksjonen $x(t)$ i ligning (38) oppgave k) kan skrives som

$$x(t) = 0.1 \sin(0.628t) + 0.2, \quad \forall \quad 0 < t < 30 \quad (51)$$

Åpne Simulink-skallfilen `oving3_a_m_skallfil.slx`, og gå videre til subsystemet `Oppgave 3m`. Der ser du at sinusfunksjonen i ligning (51) allerede er implementert. Siden vi ikke har noen integrator i modellen er valget av numerisk løser ikke så viktig. Velg derfor bare *Eulers forovermetode* med tidsskritt $T_s=0.5$, som tilsvarer en samplingsfrekvens på $f_s=2$ Hz på samme måte som utgangspunktet i oppgave k).

- I `MATLAB Function`-blokken skal du implementere IIR-filteret gitt som

$$y(k) = (1 - \alpha) \cdot y(k - 1) + \alpha \cdot x(k) \quad (35)$$

med `alfa = 0.15`. For å gjøre et indeksskift $y(k)$ til $y(k - 1)$ så benyttes en `Memory`-blokk fra `Discrete`-mappen. Denne forsinker signalet med ett tids-skritt før det mates inn som et inngangssignal. Husk å spesifisere riktig **Initial condition** for `Memory`-blokken.

Simuler i 30 sekund, og vis at du får et resultat som er identisk med det du fikk i oppgave k).

Svar

n) Demonstrasjon av sampling og nedfolding

I denne oppgaven skal du ikke programmere noe, men du kan kjøre koden og leke litt med den høyeste frekvensen f på det analoge signalet som skal samples, og samtidig bestemme hvilken samplingsfrekvens du vil bruke. Koden kan brukes til å demonstrere nedfolding, og du får følelsen av å observere dette i sann tid. Det diskret målesignalet y_k er basert på måling av det kontinuerlige “analoge” signal $y(t)$.

Som innlevering kan du ta med de signalene du har lekt med og samtidig forklare hva som skjer.

Svar

o) **Demonstrasjon av oppsampling**

Oppsampling (eng: *upsampling*) er en teknikk for å rekonstruere samplede signaler. I denne oppgaven skal vi demonstrere dette for en tenkt problemstilling hvor du har en motor med en såkalt encoder som kan måle vinkelposisjonen til motoren. Anta at motoren roterer ca. 10 runder, og at du sampler vinkelposisjonen for hver $\text{delta_phi}=260^\circ$, og at dette skjer med et tidsskrittet på $T_s=0.2$ sekund. Dette gir en vinkelhastighet på

$$\omega_d = \frac{260^\circ}{0.2} = 1300^\circ/\text{s} \quad (52)$$

hvor subskript d impliserer “degrees”. Ved å beregne og plotte

$$y(t) = \sin(\omega_d \cdot t) \quad (53)$$

så kan du få et visuelt bilde på rotasjonen som har foregått. Vær klar over at sinus-funksjonen forventer en vinkel oppgitt i radianer og ikke grader som over, og at vi strengt tatt burde skrevet

$$y(t) = \sin\left(\omega_d \cdot t \cdot \frac{2\pi}{360}\right) \quad (54)$$

Når det er sagt, så har Matlab en `sind()`-funksjon som tar grader som argument.

Som du vil se fra kurven øverst til høyre når du kjører koden, så er problemet at vi sampler såpass sjelden at det ser ut som motoren kun har rotert et par runder. Etter at du trykker OK, vil de se at vi legger inn fiktive, men egentlig høyst reelle, vinkelmålinger og tidspunkt mellom de samplede målingene. På den måten utvider vi datasettet vårt, og grunnen til at vi vet at dette går fint er at motoren har rotert jevnt i én retning.

Matlabfunksjonen for å oppsample er `interp` som interpolerer verdier mellom de avleste vinkelposisjonsmålingene.

Lek gjerne med verdien på `delta_phi`, fra 30° og oppover. Ved hvilken verdi går det galt?

Som innlevering kan du ta med de signalene du har lekt med og samtidig forklare hva som skjer.

Svar