Kapittel 19

Signalfiltrering

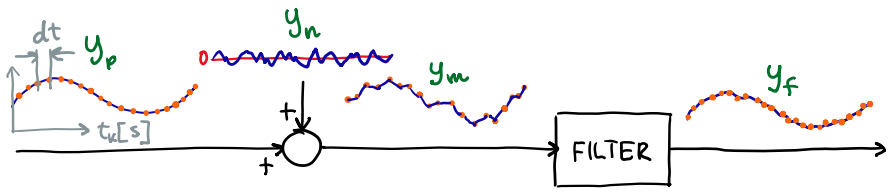
19.1 Innledning

Alle serier av målinger eller observasjoner av variabler eller signaler i et virkelig system inneholder forstyrrelser eller støy (engelsk: noise), slik:

$$y_m(t_k) = y_p(t_k) + y_n(t_k) \tag{19.1}$$

y_m er måleverdien eller målesignalet. y_p er den egentlige eller støyfrie verdien som måles. Vi kan omtale y_p som prosessverdien, f.eks. den egentlige temperaturen i en vannkoker. y_n er støyen (subindeks n for noise), som kan skyldes elektronisk eller digital støy, avlesningsunøyaktigheter eller annet. Vanligvis er y_s et mer eller mindre tilfeldig signal (engelsk: random), dvs. at det varierer tilfeldig fra et tidspunkt til et annet, og det er vanlig å anta at støyen varierer omkring null.

Ofte ønsker vi å dempe – ideelt: fjerne – støyen fra målesignalet. Vi bruker et egnet filter til å filtrere målesignalet. Det filtrerte målesignalet, y_f, er da mer eller mindre støyfritt, for vi kan vanligvis ikke regne med at støyen er helt fjernet. Figur 19.1 illustrerer filtreringen. I figuren har vi antydet at signalene er tidsdiskrete (samplede, på godt norsk).



Figur 19.1: Filtrering av støyfylt måling. Støyen antas å variere omkring null.

Filteret i figur 19.1 vil være et lavpassfilter, siden det skal sørge for at den *lavfrekvente* (relativt langsomt varierende) prosessverdien *passerer* uendret gjennom filteret, mens den høyfrekvente (relativt hurtig varierende) støyen skal dempes.[1]

[1] Hensikten med lavpassfiltere er faktisk ikke å la lavfrekvente signaler passere, men i stedet å stoppe

I tillegg til lavpassfiltere skal vi se på høypassfiltere, som brukes til å fjerne den lavfrekvente komponenten av et målesignal. Denne komponenten kan være konstantleddet eller biasen i målesignalet.

Innen tradisjonell filterteori brukes frekvensresponsteori for analyse og design av filtere, selv når sluttproduktet er en programmerbar algoritme. Vi skal i dette kapitlet hoppe bukk over denne filterteorien og gå rett på algoritmene som vil være klare for programmering. Vi skal selvsagt forklare hvordan filtrene virker, men da med utgangspunkt i selve filteralgoritmene og uten å involvere frekvensresponsteori.

Angående symbolbruk: I (19.1) er t_k diskret tid eller tidspunkt, der k er tidsindeksen. Dersom målingen skjer med tidsskritt eller samplingstid dt [s], er

$$t_k = dt \cdot k \tag{19.2}$$

Vi kommer til å forenkle notasjonen ved å skrive bare k i stedet for t_k (19.1) blir da:

$$y_{m,k} = y_{p,k} + y_{n,k} \tag{19.3}$$

Vi skal ta for oss de mest vanlige filteralgoritmene, som er:

- Middelverdifilter
- Tidskonstantfilter

Målesignaler kan foreligge på to forskjellige måter:

- Som en eksisterende måleserie (en array av måleverdier)
- Som et kontinuerlig målesignal der det kommer ny måleverdi ved hvert løpende tidspunkt

Vi nøyer oss med å se på filteralgoritmer for kontinuerlige målesignaler. Filteret beregner da filtrert målesignal ved hvert tidspunkt, etter hvert som ny måleverdi foreligger. Disse algoritmene kan imidlertid godt brukes også på eksisterende måleserier, selv om det fins bedre filteralgoritmer for slike måleserier.

19.2 Middelverdifilter

19.2.1 Filteralgoritmen

Se figur 19.1. Hvordan kan vi dempe støyen i målesignalet? Bare plukke ut støykomponenten og trekke det fra målesignalet? Nei, det går jo ikke, for vi vet jo ikke hva som er støy, og hva som er egentlig prosessverdi. Vi må implementere filtreringen på en annen måte.

høyfrekvente komponenter (støy). Derfor burde lavpassfiltere ikke hatt et slikt navn. I stedet burde de vært kalt høystoppfiltere, men det toget er gått.

Middelverdifiltere (engelsk: moving average filte[2]) gjør jobben ved å midle et antall av de nyligste måleverdiene og bruke middelverdien som filtrert verdi. Dette vil fungere bra når støyen varierer tilfeldig omkring null, for middelverdien av slik støy er tilnærmet null. Vi skal anta at støyen varierer slik.

Eksempel: Anta at filteret bruker $N_f = 5$ stk. måleverdier. Vi sier da at filteret har lengde 5. Da blir filtrert verdi ved «nå-tidspunktet» k (vi starter her summeringen med nyligste måleverdi):

$$\begin{aligned} y_{f,k} &= \frac{1}{5}\left(y_{m,k} + y_{m,k-1} + y_{m,k-2} + y_{m,k-3} + y_{m,k-4}\right) \\ &= \frac{1}{5}\sum_{i=0}^{4} y_{m,k-i} \end{aligned}$$

Den generelle formelen for $y_{f,k}$ blir:

$$y_{f,k} = \frac{1}{N_f}\sum_{i=0}^{N_f-1} y_{m,k-i} \tag{19.4}$$

Når vi skal implementere (19.4), må vi ved hvert tidspunkt k legge inn den nyligste måleverdien og kaste ut den eldste. Summeringen kan vi godt bruke numpy.sum() til.

Merk at for en gitt N_f kan vi definere et tilsvarende tidsvindu, T_v [s]:

$$T_v = dt \cdot N_f \tag{19.5}$$

Eksempel: Med $N_f = 25$ og $dt = 0{,}01$ blir $T_v = 0{,}25$ s.

Eksempel 19.1 *Middelverdifiltering*

Program 19.1 simulerer et støyfylt sinusformet målesignal og filtrerer signalet med et middelverdifilter med følgende parameterverdier: $dt = 0.01$. $t_{start} = 0$. $t_{stop} = 20$ s. $N_f = 25$.

Figur 19.2 viser signalene.

http://techteach.no/python/files/prog_ma_filter.py

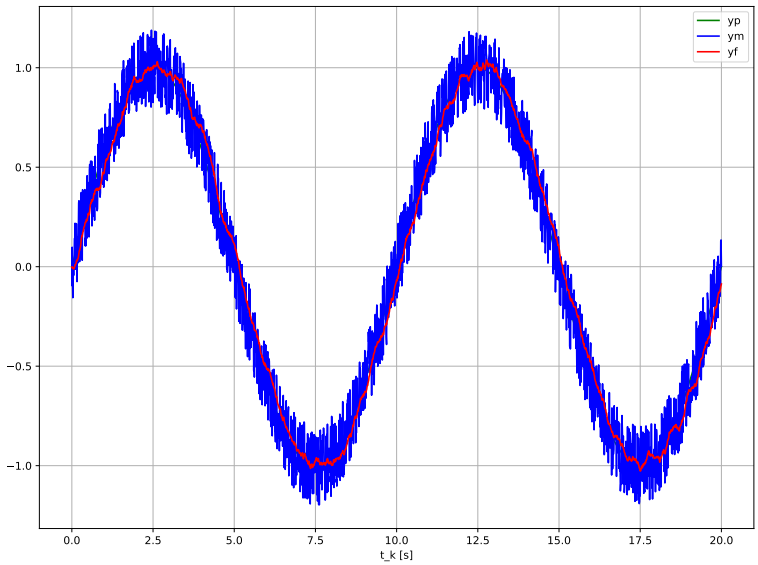
Listing 19.1: prog_ma_filter.py

```
# %% Import of packages:

import matplotlib.pyplot as plt
import numpy as np

# %% Filter function:
```

[2]moving average filter



Figur 19.2: Middelverdifiltrering av et støyfylt signussignal.

```
def fun_ma_filter(ym_k, ma_array, Nf):

    # Updating meas array with new measurement:
    ma_array = np.roll(ma_array,1)
    ma_array[0] = ym_k
    yf_k = np.sum(ma_array)/Nf

    return (yf_k, ma_array)

# %% Simulation time settings:

dt = 0.01  # [s]
t_start = 0  # [s]
t_stop = 20  # [s]
N_sim = int((t_stop - t_start)/dt) + 1  # Num time-steps

# %% Preallocation of arrays for plotting:

t_array = np.zeros(N_sim)
yp_array = np.zeros(N_sim)
yn_array = np.zeros(N_sim)
ym_array = np.zeros(N_sim)
yf_array = np.zeros(N_sim)

# %% Params of signals:

P = 10  # [s] Period of sine
a_yn = 0.2  # Ampl of uniformly distributed random noise
samples_yn = 1
```

```
# %% Filter param:

Nf = 25  # Filter length
ma_array = np.zeros(Nf)  # Array of measurements

# %% Simulation loop:

for k in range(0, N_sim):

    t_k = k*dt  # Time

    # Signals:
    yp_k = np.sin(2*np.pi*(1/P)*t_k)
    yn_k = np.random.uniform(-a_yn, a_yn, samples_yn)[0]
    ym_k = yp_k + yn_k

    # MA filter:
    (yf_k, ma_array) = fun_ma_filter(ym_k, ma_array, Nf)

    # Arrays for plotting:
    t_array[k] = t_k
    yp_array[k] = yp_k
    yn_array[k] = yn_k
    ym_array[k] = ym_k
    yf_array[k] = yf_k

# %% Plotting:

plt.close('all')
plt.figure(1, figsize=(12, 9))

plt.plot(t_array, yp_array, 'g', label='yp')
plt.plot(t_array, ym_array, 'b', label='ym')
plt.plot(t_array, yf_array, 'r', label='yf')
plt.legend()
plt.xlabel('t_k [s]')
plt.grid()

# plt.savefig('plot_ma_filt.pdf')
plt.show()
```

Kommentarer:

- numpy-funksjonen random.uniform() genererer støysignalet.
- Arrayen ma_array lagrer måleverdiene som skal brukes i filteret.
- Vi har implementert filteret i vår egenutviklede funksjon fun_ma_filter(). numpy-fuksjonen roll() brukes til å flytte arrayelementene ett trinn før vi inkluderer den nyligste måleverdien i arrayen. Filterverdien beregnes som np.sum()/len(ma_array).

- Plottet i figur 19.2 viser at støyen er dempet gjennom filteret.

[Slutt på eksempel 19.1]

19.2.2 Noen egenskaper ved middelverdifilteret

19.2.2.1 Filterets lavpassegenskaper

Middelverdifilteret er et lavpassfilter. Da må vi forvente at et støyfritt konstant målesignal komme ut uendret på filterutgangen. Blir det slik iht. filteralgoritmen (19.4)? La oss sett inn konstanten $y_m = Y_m$ inn i (19.4). Vi får da:

$$y_{f,k} = \frac{1}{N_f} \sum_{i=0}^{N_f-1} y_{m,k-i} = \frac{1}{N_f} \sum_{i=0}^{N_f-1} Y_m = \frac{1}{N_f} \cdot N_f \cdot Y_m = Y_m \tag{19.6}$$

Ja, forventningen er innfridd.

19.2.2.2 Tidsforskyvning gjennom filteret

Det kan vises at tidsforskyvningen av målesignalet gjennom filteret er ca:

$$\Delta t = \frac{dt \cdot N_f}{2} \tag{19.7}$$

Eksempel 19.2 *Tidsforskyvning gjennom filteret*

Vi kjører program 19.1 i eksempel 19.1 med $N_f = 200$ i stedet for 25. Vi setter målestøyen $y_n = 0$ for å se sinuskomponenten i målesignalet tydelig. Tidsforskyvningen bør iht. (19.7) da være:

$$\Delta t = \frac{0{,}01 \cdot 200}{2} = 1\text{s} \tag{19.8}$$

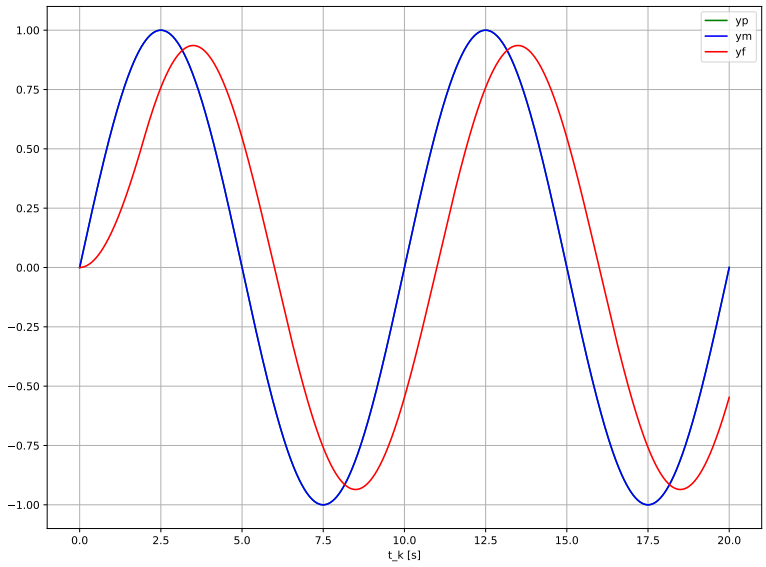
Figur 19.3 viser signalene. Vi leser av at forskyvningen er ca. 1 sek, som stemmer med (19.8).

[Slutt på eksempel 19.2]

Et alternativt begrep til tidsforskyvning er faseforskyvning, som uttrykker hvor mye fase- eller vinkelforskyvning et sinusformet signal får gjennom filteret. Vi går ikke nærmere inn på dette.

19.2.2.3 Støydempning gjennom filteret

Her er et interessant faktum: Anta prosessverdi $y_p = 0$. Anta at målestøyen y_n er tilfeldig (random) med standardavvik σ_n. Filterutgangen y_f får et standardavvik σ_f som typisk er



Figur 19.3: Middelverdifiltrering av et støyfylt signussignal.

mindre enn σ_n. Det kan vises at forholdet mellom standardavvikene er:

$$\frac{\sigma_f}{\sigma_n} \approx \frac{1}{\sqrt{N_f}} = \sqrt{\frac{dt}{T_v}} = S \tag{19.9}$$

som vi kan kalle støydempningen (S). Jo større filterlengde, jo større støydempning – bra!, men ikke glem at tidsforskyvningen da kan bli stor, jf. (19.7) – ikke bra!

Med andre ord: Filterlengden N_f må velges som et kompromiss mellom støydempning og tidsforskyvning (eller faseforskyvning). Hva bør N_f velges som? Det er ikke lett å gi et presist råd. Du kan kanskje prøve med 25 som en start. Da får du $S = 1/\sqrt{25} = 0{,}2$.

Støydempningen skal beregnes i en simulering i oppgave 19.1.

19.3 Tidskonstantfilter

19.3.1 Filteralgoritmen

En ingeniør kjenner sannsynligvis til denne matematiske modellen for et såkalt tidskonstantsystem:

$$Ty' = Ku - y \tag{19.10}$$

der T [s] er tidskonstanten og K er forsterkningen. u er systemets inngangssignal, og y er utgangssignalet.[3] Vi skal ikke analysere dette systemet her, men bare informere om at såkalte tidskonstantfiltere, som er mye brukt bl.a. innen praktisk signalbehandling og reguleringsteknikk, er basert på en filtermodell som er nettopp (19.10), da med $K = 1$. T kalles da filtertidskonstanten, og vi kan innføre symbolet T_f for den. Hvis vi også innfører symbolet y_m for filterinngangen (målingen) og y_f for filterutgangen, får vi følgende differensiallikning som definerer tidskonstantfilteret:

$$T_f y_f' = y_m - y_f \tag{19.11}$$

Vi skal nå finne en filteralgoritme ut fra (19.11). Vi kan diskretisere (19.11) på samme måte som vi diskretiserte differensiallikninger i kap. 15 om simulatorer. I det kapitlet brukte vi Eulers forover-integrasjon. Det viser seg at fagfolk som skal ha en programmerbar filteralgoritme som bygger på tidskonstantfilteret (19.11), i stedet bruker Eulers bakover-integrasjon. Vi gjør som dem.

Fra (19.11) får vi:

$$y_f' = \frac{1}{T_f}\left(y_m - y_f\right)$$

som ved tidspunkt k er:

$$y_{f,k}' = \frac{1}{T_f}\left(y_{m,k} - y_{f,k}\right)$$

Eulers bakover-integrasjon gir:

$$y_{f,k} \quad = \quad y_{f,k-1} + dt \cdot y_{f,k}' = y_{f,k-1} + dt \cdot \left[\frac{1}{T_f}\left(y_{m,k} - y_{f,k}\right)\right]$$

som løst mht. filterutgangen $y_{f,k}$ gir følgende filteralgoritmen for tidskonstantfilteret:

$$y_{f,k} = (1-a)\, y_{f,k-1} + a y_{m,k} \tag{19.12}$$

der vi har innført parameteren a, som er:

$$a = \frac{dt}{T_f + dt} \tag{19.13}$$

Filteralgoritmen (19.12) er en rekursiv filteralgoritme. I (19.12) trenger vi ikke lagre noe array av målesignalene, slik vi gjorde i program 19.1, men vi må lagre $y_{f,k}$ slik at vi, etter tidsskift nederst i programmets løkke, har $y_{f,k-1}$ tilgjengelig i neste iterasjon av filteralgoritmen.

I praksis vil $dt \ll T_f$, så a vil være ganske nær 1. Et eksempel med typiske verdier: $dt = 0{,}1$ s og $T_f = 5$ s. Da blir $a \approx 0{,}02$ og $(1-a) \approx 0{,}98$. Filteralgoritmen (19.12) ser i så fall slik ut:

$$y_{f,k} = 0{,}98 y_{f,k-1} + 0{,}02 y_{m,k} \tag{19.14}$$

Ser du at (19.12) vil fungere som et lavpassfilter? Du kan anta $a \approx 0{,}02$. [4]

Oppgave 19.2 dreier seg om å programmere et tidskonstantfilter.

[3] Dette er en modell som kan representere de dynamiske egenskapene til mange forskjellige slags systemer – elektriske, mekaniske, biologiske m.m.

[4] Algoritmen har mindre vekt på den støyfylte målingen $y_{m,k}$ enn på den filterutgangen $y_{f,k}$. Derfor kan vi regne med at filteret demper støyen.

19.3.2 Noen egenskaper ved filteret

19.3.2.1 Filterets lavpassegenskaper

La oss sett inn konstanten $y_m = Y_m$, som jo er et til de grader lavfrekvent signal, inn i (19.12), og anta at filterutgangen blir en konstant verdi, Y_f. (19.12) blir da:

$$Y_f = (1 - a)\, Y_f + aY_m \tag{19.15}$$

som gir:

$$Y_f = Y_m$$

Utgangen blir altså lik inngangen, statisk. Altså fungerer filteret som et *lavpass*filter, som forventet.

19.3.2.2 Tidsforskyvning gjennom filteret

Det kan vises at tidsforskyvningen av målesignalet gjennom et tidskonstantfilteret er ca.:

$$\Delta t = T_f \tag{19.16}$$

19.3.2.3 Støydempning gjennom filteret

I kap. 19.2.2.3 presenterte vi en formel for forholdet mellom standardavvikene av filterutgangen og filterinngangen (gitt av prosessverdien er null, slik at det kun er målstøy på inngangen). Det kan vises at forholdet mellom standardavvikene for tidskonstantfilteret er:

$$\frac{\sigma_f}{\sigma_n} \approx \sqrt{\frac{dt}{2T_f}} = S \tag{19.17}$$

der S er støydempningen. Jo større T_f, jo større støydempning – bra!, men ikke glem at tidsforskyvningen da kan bli stor, jf. (19.16) – ikke bra!

19.3.2.4 En sammenheng mellom tidskonstantfilter og middelverdifilter

La oss kreve at de to støydemningene (19.17) og (19.9) skal være like:

$$S = \frac{\sigma_f}{\sigma_n} \approx \sqrt{\frac{dt}{2T_f}} = \sqrt{\frac{dt}{T_v}} \tag{19.18}$$

som tilsier at tidskonstantfilteret og middelverdifilteret har like statistiske egenskaper dersom:

$$T_v = 2T_f \tag{19.19}$$

(gitt at de har samme dt,selvsagt). Med andre ord: Det spiller ikke så stor rolle hvilken av filteralgoritmene du implementerer – du kan få dem til å oppføre seg temmelig likt dersom (19.19) er oppfylt.

19.3.2.5 Hvordan skal vi stille inn filtertidskonstanten?

Vi kan betrakte filtertidskonstanten T_f i (19.13) som den justerbare filterparameteren. Vi har allerede sett at jo større T_f, jo større støydempning (bra), men samtidig blir tidsforskyvningen gjennom filteret større (ikke bra). Svaret på hva T_f skal stilles inn på, blir da et kompromiss der du må ta hensyn til lokale forhold. F.eks. bør et filter som skal filtrere en rotasjonsmåling på en motor, helt klart ha en mindre T_f enn for et filter som filtrerer en temperaturmåling i en biogassreaktor (som er en temmelig treg prosess). Noe prøving og feiling må du regne med, men hvis du ikke har peiling på hva du skal starte med, så kan du jo starte med 2 sekunder!

19.4 Høypassfilter

Lavpassfiltere brukes til å dempe – helst: stoppe – uønskede, høyfrekvente signaler i et målesignal. Da kan du sikkert tenke deg hva høypassfiltere brukes til ... Ja, til å stoppe lavfrekvente signaler. Et stjerneeksempel på et lavfrekvent signal er en konstant eller en bias. Et høypassfilter kan altså brukes til å fjerne en slik konstant fra et signal.

Det er mange måter å uvikle høypassfilter-algoritmer på. Vi skal gjøre det slik: Vi har allerede laget lavpassfiltere – både et middelverdifilter og et tidskonstantfilter. La oss bruke tidskonstantfilteret. Hvis vi kan fjerne den lavfrekvente delen av et målesignal, må vel kun den høyfrekvente delen stå igjen? Vi satser på at det stemmer. Vi kan da lage et høypassfilter slik:

$$y_{\mathrm{HP},f,k} \quad = \quad y_{m,k} - y_{\mathrm{LP},f,k} \tag{19.20}$$

der:

$$y_{\mathrm{LP},f,k} = (1-a)\, y_{\mathrm{LP},f,k-1} + a y_{m,k} \tag{19.21}$$

der:

$$a = \frac{dt}{T_f + dt} \tag{19.22}$$

Til beroligelse for dem som måtte kjenne til alternative måter å utlede et første ordens høypassfilter på: Det kan vises at vi får den samme filteralgoritmen, dvs. (19.20)–(19.21), dersom vi starter med en modellbasert definisjon av et første ordens høypassfilter og så diskretiserer modellen med Euler bakover-derivasjon.

Parameteren å stille på i høypassfilteret er filtertidskonstanten T_f i (19.13). Vi nøyer oss her med å si at det du må regne med å finne fram til en passende T_f-verdi med prøving-og-feiling. Har du ikke peiling på hva du skal starte med, så kan du jo starte med $T_f = 2$ sekunder.

Oppgave 19.3 dreier seg om å programmere et høypassfilter.

19.5 Oppgaver til kapittel 19

Oppgave 19.1 *Støydempning*

Du skal her sjekke om formel (19.9) for støydempningen holder.

Ta utgangspunkt i program 19.1 i eksempel 19.1. Modfiser programmet iht. følgende:

- $y_p = 0$ (konstant).
- $dt = 0.001$. $t_{start} = 0$. t_{stop} =100 s. Dette gir relativt lange dataserier (1 million verdier), hvilket vil bidra til at forholdene i dette eksperimentet ligger nær de teoretiske antakelsene som ligger bak (19.9)
- $N_f = 10$.
- Plotting kan droppes.
- Implementer (19.9). Du kan beregne standardavvikene for y_m og y_f med numpy-funksjonen std().

Kjør programmet. Holder (19.9) i dette «eksperimentet»?

Oppgave 19.2 *Programmering av tidskonstantfilter*

Oppgave a

Ta utgangspunkt i program 19.1 i eksempel 19.1. Programmet implementerer filteralgoritmen (19.4). Modifiser programmet ved at filteralgoritmen i stedet blir et tidskonstantfilter, og kjør programmet.

Oppgave b

Les av tidsforskyvningen Δt mellom innsignalet og utsignalet i plottet som du får i oppgave a. Stemmer den med den teoretiske verdien gitt ved (19.16)?

Oppgave 19.3 *Programmering av høypassfilter*

Programmer høypassfilteret (19.20)–(19.21), og kjør en simulering med følgende spesifikasjoner:

- dt = 0,01 s. T_f = 1 s.
- Prosessverdien y_p er en konstant med verdi 1.

- Målestøyen er et uniformt fordelt tilfeldig (random) signal mellom ± 0,02. Du kan bruke numpy-funksjonen random.uniform() til å generere støysignalet.
- Plott filterets inngang og utgang i samme plott.

Tips: Du kan ta utgangspunkt i program
http://techteach.no/python/files/prog_timeconstant_filter.py som implementerer et tidskonstantfilter.

Ser du at filteret virker som et høypassfilter?

19.6 Løsninger til kapittel 19

Løsning til oppgave 19.1

Program http://techteach.no/python/files/prog_ma_filter_noise_damping.py viser en løsning.

Resultat av en programkjøring:

```
S = std_yf/std_ym = 0.09932068513955186
S = 1/np.sqrt(Nf) = 0.1
```

Vi må kunne si at (19.9) holder veldig bra.

Løsning til oppgave 19.2

Løsning til oppgave a

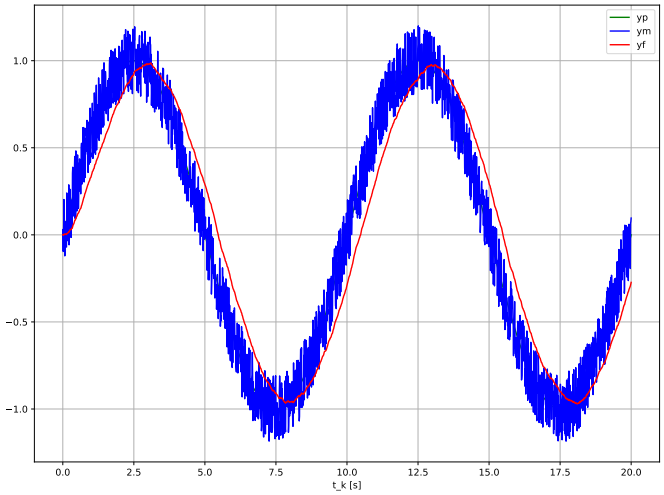
Program http://techteach.no/python/files/prog_timeconstant_filter.py implementerer filteret med den rekursive filteralgoritmen (19.12). Figur 19.4 viser resultatet av programkjøringen.

Løsning til oppgave b

Vi ser av tidsforskyvningen mellom innsignalet og utsignalet i plottet i figur 19.4 til å være ca. 0,5 s. Iht. (19.16) skal den være $\Delta t = T_f =$ 0,5s. Resultatene stemmer godt.

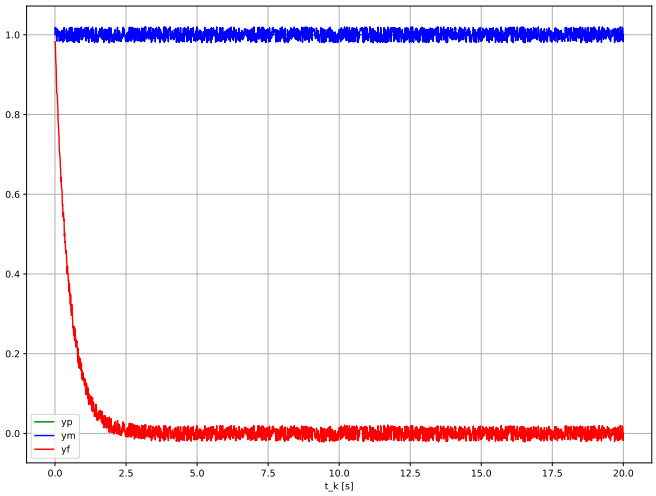
Løsning til oppgave 19.3

Program http://techteach.no/python/files/prog_hp_filter.py implementer filteret med den rekursive filteralgoritmen (19.12). Figur 19.5 viser resultatet av programkjøringen. Vi ser at filterert fungerer som et høypassfilter: Filterutgangen (rød kurve) går mot null, så



Figur 19.4: Filtrering med tidskonstantfilter.

konstantverdien (en ekstremt lavfrekvent komponent) i prosessverdien er fjernet gjennom filteret.



Figur 19.5: Filtrering med høypassfilter.