Simuleringsalgoritmer

Finn Aakre Haugen, dosent
Høgskolen i Telemark

14. september 2015

1 Innledning

1.1 Hva er simulering?

Simulering av et system er beregning av tidsresponser vha. en matematisk modell av systemet. Responsene beregnes som en numerisk løsning av differensiallikningene som utgjør systemets modell. Den numeriske løsningen gir tidsutviklingen av variablene i modellen.

1.2 Hvorfor simulere?

Kapittel 12 i læreboka Reguleringsteknikk [Haugen, 2014] beskriver i detalj, og med eksempler, hvilken nytte vi kan av simulatorer. Her er en oppsummering:

- **Testing:** F.eks. kan regulatorer testes mot simulerte prosesser før de settes i drift mot den virkelige prosessen.
- **Regulatorinnstilling:** Du kan bruke eksperimentelle metoder for regulatorinnstilling, f.eks. Ziegler-Nichols svingemetode eller Skogestad-metoden, på en simulert prosess før regulatoren tas i bruk på den virkelige prosessen.
- **Analyse:** Med simuleringer kan en kjøre eksperimenter som av praktiske eller økonomiske grunner ellers ikke kan kjøres på det virkelige systemet. En kan se hvordan variable som en ellers ikke kan måle, utvikler seg. Dette kan gi ny innsikt og forståelse.

- **Opplæring og trening:** Simuleringer kan med stor fordel brukes i opplæring i studier og i operatørtrening.
- **Utforming (design):** En kan bruke en simulator til utforming av et planlagt virkelig system, som altså ennå ikke eksisterer!

1.3 Hva dokumentet omhandler

Dette dokumentet beskriver hvordan du kan utvikle simuleringsalgoritmer for dynamiske modeller (systemer), klare for programmering i et hvilket som helst programmeringsspråk. Modellene som dekkes i dette dokumentet, er av følgende to typer:

- 1. ordens differensiallikning (gjerne ulinær)
- Tidsforsinkelse

Det er min erfaring fra simulatorutvikling i mange ulike prosjekter at disse to modelltypene dekker de aller fleste anvendelser.

Hva hvis modellen er en 2. ordens differensiallikning[1], som jo er tilfelle for bevegelseslikningene for et legeme? Som vist senere i dokumentet, kan du utvikle en simuleringsalgoritme for en 2. ordens differensiallikning ved å skrive denne differensiallikningen som to 1. ordens differensiallikninger og så lage en simuleringsalgoritme for hver av dem.

Dokumentet viser ikke hvordan vi kan lage en simulator for et tilbakekoplet reguleringssystem. Hvis vi bruker LabVIEW, som jo er blokkdiagrambasert, er det nokså rett-fram å kombinere en PID-regulatorfunksjon (LabVIEWs innebygde eller en egenutviklet) med en simuleringsalgoritme for prosessen som skal reguleres, siden struktur av LabVIEWs blokkdiagram vil være som for reguleringssystemet. I et rent tekstbasert programmeringsspråk, som f.eks. Matlab, kan du implementere prosessimulatoren "øverst"i programmet og regulatoren nedenfor. Både prosessen og regulatoren må selvsagt implementeres inne i for- eller while-løkka.

[1] Differensiallikninger av orden høyere enn 2 har jeg aldri møtt.

2 Simuleringsalgoritme for 1. ordens differensiallikning

Anta at modellen som skal simuleres, består av følgende differensiallikning:

$$\dot{x}(t) = f(\cdot, t) \tag{1}$$

der uttrykket $f(\cdot, t)$ representer alt som står på høyresiden av differensiallikningen.t er tidsvariabelen.

Anta at vi skal utlede en simuleringsalgoritme som beregner verdien av x, som forresten kalles modellens tilstandsvariabel, som funksjon av t. En simuleringsalgoritme implementert i et program som kjører i en datamaskin, kan beregne x kun ved diskrete tidspunkter. Disse tidspunktene skal vi her betegne t_k der k er en tellevariabel eller tidsindeks for de diskrete tidspunktene. k er heltallig. Avstanden i tid mellom de diskrete tidspunktene kalles simuleringstidsskrittet, eller bare tidsskrittet, som vi her symboliserer med T_{sim}. Med andre ord: T_{sim} er oppløsningen langs simuleringstidsaksen. Figur 1 illusterer disse størrelsene. I figuren er tidsskrittet antatt å være 0,1 s. (Tallverdiene langs t- og x-aksene er her kun antatte verdier.)

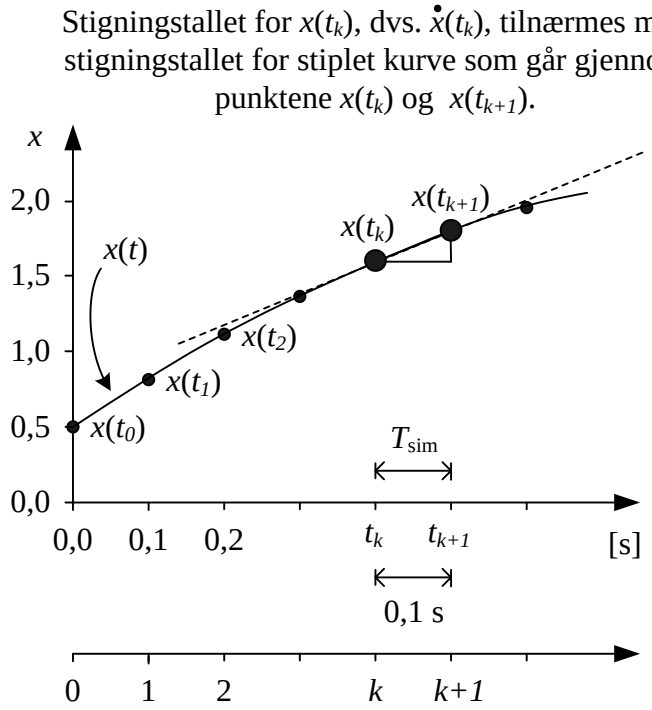
I figur 1 representerer den heltruktne kurven en kontinuerlig (eller analog) respons, $x(t)$. Vi skal finne en simuleringsalgoritme for beregning av den tilsvarende tidsdiskrete (eller digitale) responsen, $x(t_k)$. Det fins mange forskjellige simuleringsalgoritmer, og det fins mange metoder for utvikling av algoritmer. Vi skal her utlede den enkleste av alle simuleringsalgoritmer, nemlig Eulers foroveralgoritme, også kalt Eulers eksplisitte algoritme. Denne algoritmen kan utledes fra en numerisk tilnærmelse (eller approksimasjon) til integrasjon eller fra en numerisk tilnærmelse til tidsderivasjon. Av disse to utledningene velges her sistnevnte, siden den er den enkleste av de to.

Vi skal altså utlede Eulers foroveralgoritme ut fra en tilnærmelse til den tidsderiverte. Anta at modellen som skal simuleres, er gitt ved likn. (1). Eulers foroveralgoritme er basert på at $\dot{x}(t_k)$ tilnærmes med stigningstallet for den stiplede kurven som går igjennom punktene $x(t_k)$ og $x(t_{k+1})$, se figur 1. Matematisk uttrykt:

$$\dot{x}(t_k) \approx \frac{x(t_{k+1}) - x(t_k)}{T_{\text{sim}}} \tag{2}$$

Vi erstatter nå $\dot{x}(t_k)$ i likn. (1) med denne tilnærmelsen. Likn. (1) blir da

$$\frac{x(t_{k+1}) - x(t_k)}{T_{\text{sim}}} \approx f(\cdot, t_k) \tag{3}$$



Figur 1: Diskret tidsakse

Vi løser nå likn. (3) med hensyn på $x(t_{k+1})$ og har dermed kommet fram til Eulers foroveralgoritme (vi skriver nå for enkelthets skyld "="i stedet for "≈", men vi vet jo at det egentlig skal stå "≈"):

$$x(t_{k+1}) = x(t_k) + T_{\text{sim}} \cdot f(\cdot, t_k) \tag{4}$$

I likn. (4) representerer $f(\cdot, t_k)$ verdien av funksjonen f beregnet for tidspunkt t_k.

Siden $f(\cdot, t_k)$ er det samme som $\dot{x}(t_k)$, kan vi skrive Eulers foroveralgoritme som

$$\boxed{x(t_{k+1}) = x(t_k) + T_{\text{sim}} \cdot \dot{x}(t_k)} \tag{5}$$

hvilket gir litt enklere symbolbruk siden vi nå har droppet symbolet f.

Algoritme (5) skal beregnes for hvert tidsskritt i simuleringen. Initialtilstanden $x(t_0)$ må være kjent.

Algoritme (5) er en eksplisitt formel for ett-skritts prediksjonen $x(t_{k+1})$. Men vanligvis er det $x(t_k)$, som er verdien av x ved "nåtidspunktet"t_k,

som vi ønsker verdien av. $x(t_k)$ vil selvsagt også være tilgjengelig i algoritmen, nemlig som prediksjonen beregnet ved forrige tidsskritt. Dette kommer forhåpentligvis klart fram i pseudokoden i følgende eksempel.

Eksempel 1 ***Psuedokode for simuleringsalgoritme for 1. ordens differensiallikning***

Gitt modellen

$$a\dot{x}(t) = bx(t) + cu(t) \tag{6}$$

der $a = 0,5$, $b = -2$ og $c = 5$ er parametre. Initialtilstanden er $x(t_0) = 3$. u er en inngangsvariabel i form av en rampe med startverdi 0 og stigningstall (rate) $R = 0,1$.

Vi starter med å skrive modellen på standardformen, jf. likn. (1), dvs. at den tidsderiverte skal stå alene på venstre side:

$$\dot{x}(t) = \frac{b}{a}x(t) + \frac{c}{a}u(t) \tag{7}$$

Nedenfor er simuleringsalgoritmen implementert i pseudokode. Algoritmen kjører der syklisk i en for-løkke, men en while-løkke som kjører inntil en stoppbetingelse er oppfylt (f.eks. at brukeren har klikket på en stoppknapp), kan også brukes. Ved implementering i LabVIEW kan (bør) simuleringsalgoritmen programmeres i en Formula Node siden det kan være tungvint å realisere matematiske uttrykk vha. beregningsblokker. Husk at initialisering og preallokering av arrays for datalagring skal utføres kun n gang, jf. pseudokoden nedenfor.

```
%Initialisering:
a=0.5;
b=-2;
c=5;
R=0.1;
x_0=3;
x_k=x_0;

T_sim=0.01;
t_slutt=10;
N=t_slutt/T_sim; %Lengden av simuleringen i antall tidsskritt.

%Preallokkering av arrays for lagring av sim-data:
t=zeros(0,N);
```

```
u_tidsserie=zeros(0,N);
x_tidsserie=zeros(0,N);

for k_sim = 0 : N,
{      %Starten av simuleringssyklusen.
t_k=k_sim*T_sim;
u_k=R*t_k;
dot_x_k=(b/a)*x_k+(c/a)*u_k;
x_kp1=x_k+T_sim*dot_x_k;

%Lagring av verdier i arrays for plotting
%evt for eksport og analyse av data:
t(k_sim)=t_k;
u_tidsserie(k_sim)=u_k;
x_tidsserie(k_sim)=x_k;

%Indeksskift (forberedelse for neste tidsskritt):
x_k=x_kp1;
}      %Slutten av simuleringssyklusen.
figure(1)
plot(t,x_tidsserie)
figure(2)
plot(t,u_tidsserie)
```

Figur 2 viser inngangssignalet $u(t)$ og responsen i tilstandsvariabelen $x(t)$. (Simuleringsalgoritmen er programmert i Matlab, men er lik pseudokoden ovenfor.)

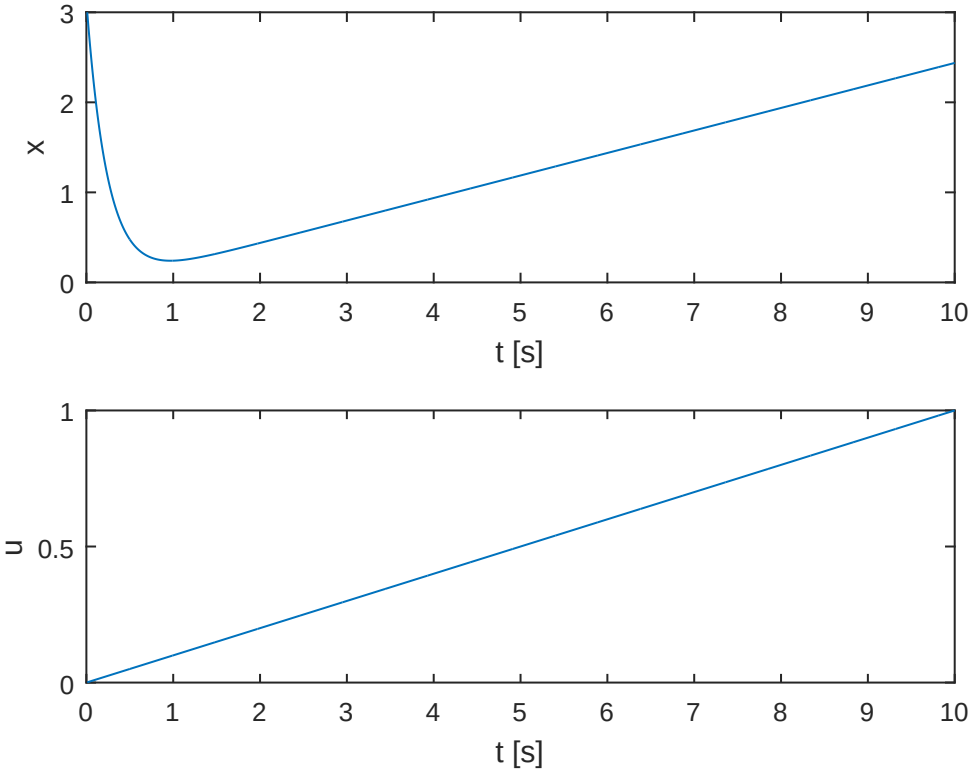
[Slutt på eksempel 1]

Simulering av 2. ordens differensiallikning Hvis modellen som skal simuleres, inneholder en 2. ordens differensiallikning, kan vi skrive denne differensiallikningen som to 1. ordens differensiallikninger og så lage en simuleringsalgoritme for hver av dem.

Et eksempel er modellen for et masse-fjær-demper-system med følgende modell, jf. eksempel 6.3 på side 214 i læreboka Reguleringsteknikk [Haugen, 2014]:

$$m\ddot{y}(t) = -D\dot{y}(t) - Ky(t) + F(t) \tag{8}$$

der m [kg] er legemets masse, D [N/(m/s)] er dempekonstanten, K [N/m] er fjærkonstanten, og F er ytre kraft på legemet. Initialtilstandene er $y(0)$ (posisjon) og $\dot{y}(0)$ (hastighet). Vi starter med å sørge for at den



Figur 2: Eksempel 1: Simulert inngangssignal $u(t)$ og responsen i tilstandsvariabelen $x(t)$

dobbelderiverte står alene på venstre side:

$$\ddot{y}(t) = \frac{1}{m}[-D\dot{y}(t) - Ky(t) + F(t)] \tag{9}$$

Så definerer vi følgende to tilstandsvariable:

$$x_1(t) = y(t) \text{ (posisjon)} \tag{10}$$

$$x_2(t) = \dot{y}(t) \text{ (hastighet)} \tag{11}$$

Fra disse definisjonene får vi

$$\dot{x}_1(t) = \dot{y}(t) = x_2(t) \text{ (hastighet)} \tag{12}$$

$$\dot{x}_2(t) = \ddot{y}(t) \text{ (akselerasjon)} \tag{13}$$

Den opprinnelige modellen, likn. (9), kan dermed representeres med følgende to 1. ordens differensiallikninger:

$$\dot{x}_1(t) = x_2(t) \text{ (hastighet)} \tag{14}$$

$$\dot{x}_2(t) = \frac{1}{m}[-Dx_2(t) - Kx_1(t) + F(t)] \text{ (akselerasjon)} \tag{15}$$

Vi kan nå lage en simuleringsalgoritme for den gitte modellen ved å anvende Eulers foroveralgoritme på både likn. (14) og likn. (15). Initialtilstandene er $x_1(0) = y(0)$ og $x_2(0) = \dot{y}(0)$.

3 Simulering av tidsforsinkelse

Tidsforsinkelse kalles også noen ganger transportforsinkelse eller dødtid.

Ved simulering av reguleringssystemer er det viktig at eventuelle tidsforsinkelser er inkludert i simulatoren. Grunnen er at tidsforsinkelser i prosessen har stor betydning for stabilitetsforholdene i reguleringssystemet. Hvis du har stilt inn en regulator for en prosess *uten* tidsforsinkelse, mens det faktisk *er* en tidsforsinkelse i den virkelige prosessen, er det en mulighet for at reguleringssystemet for den virkelige prosessen blir ustabilt!

Tidsforsinkelser kan simuleres med et array. Hvis tidsforsinkelsen er τ [s] og simuleringstidsskrittet er T_{sim}, kan arrayets lengde, N_τ, settes lik

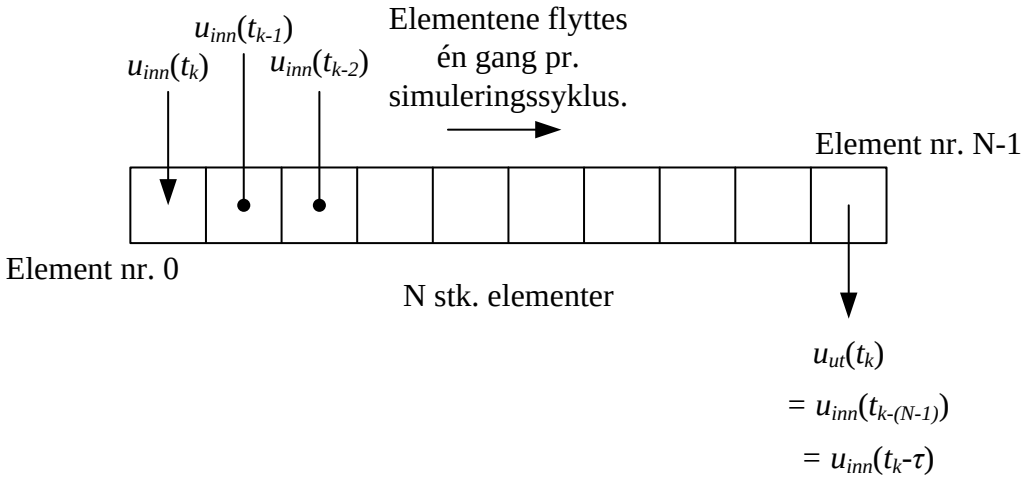
$$N_\tau = \frac{\tau}{T_{\text{sim}}} \tag{16}$$

eller avrundet oppover til nærmeste heltall dersom T_{sim} ikke går opp i τ.

Figur 3 illustrerer hvordan en tidsforsinkelse kan realiseres vha. et array. Rekkefølgen av operasjonene i hver syklus av simuleringsalgoritmen er:

1. Det tidsforsinkede signalet, $u_{\text{ut}}(t) = u_{\text{inn}}(t - \tau)$, kan hentes ut som det siste elementet i arrayet.
2. Signalet som skal tidsforsinkes, $u_{\text{inn}}(t)$, legges inn i det første element i arrayet.
3. Alle elementene flyttes ett trinn (i retning av økende arrayindeks).

Eksempel 2 ***Implementering av tidsforsinkelse i MathScript Node i LabVIEW***



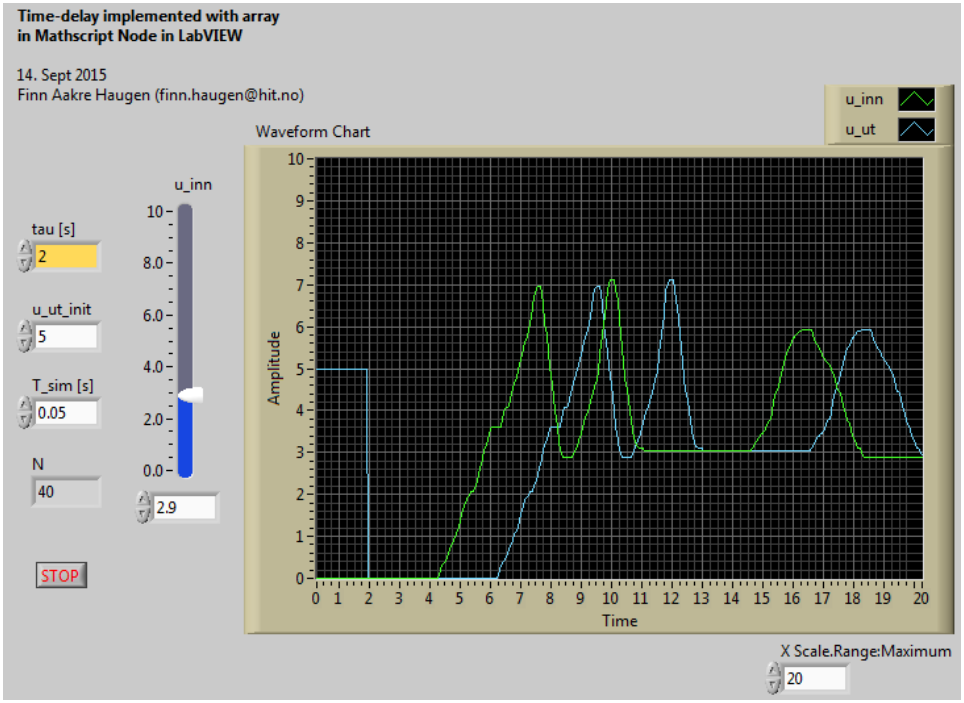
Figur 3: Realisering av en tidsforsinkelse vha. et array.

Figur 4 viser frontpanelet for et LabVIEW-program som simulerer en tidsforsinkelse. Vi ser at inngangssignalet u_{inn} er tidsforsinket med 2 sek. Figur 5 viser LabVIEW-programmets blokkdiagram.

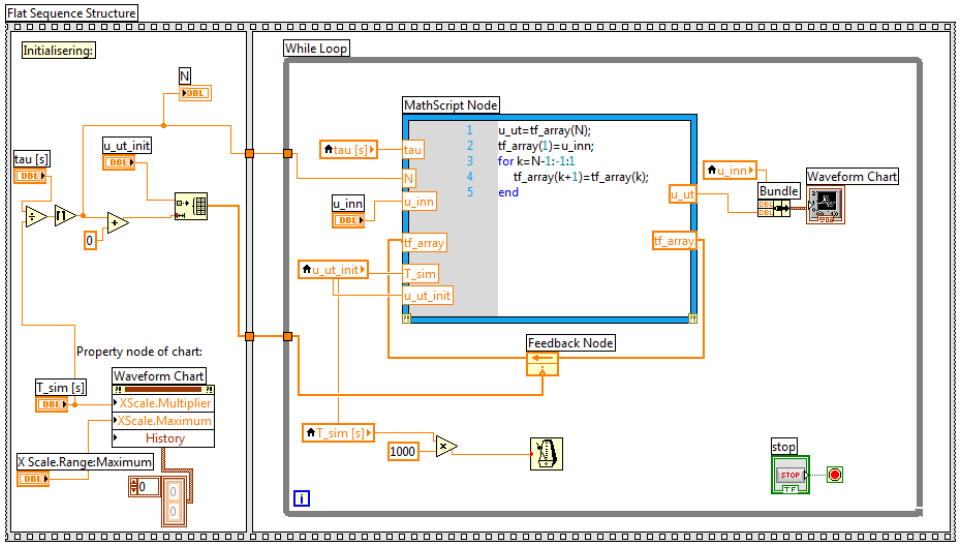
[Slutt på eksempel 2]

Referanser

[Haugen, 2014] Haugen, F. A. (2014). *Reguleringsteknikk, 2. utgave.* Fagbokforlaget.



Figur 4: Frontpanelet for et LabVIEW-program som simulerer en tidsforsinkelse.



Figur 5: Blokkdiagrammet for et LabVIEW-program som simulerer en tidsforsinkelse.