Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Att lösa reglertekniska problem med Modelica

Examensarbete utfört i Reglerteknik
vid Tekniska högskolan i Linköping
av

Ahmed Ali Abdul-Amir

LiTH-ISY-EX–08/3973–SE

Linköping 2008

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Att lösa reglertekniska problem med Modelica

Examensarbete utfört i Reglerteknik
vid Tekniska högskolan i Linköping
av

Ahmed Ali Abdul-Amir

LiTH-ISY-EX–08/3973–SE

Handledare: **Dr. Johan Sjöberg**
ISY, Linköpings universitet

Examinator: **Prof. Torkel Glad**
ISY, Linköpings universitet

Linköping, 24 mars, 2008

Avdelning, Institution
Division, Department

Division of Automatic Control
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Datum
Date

2008-03-24

Språk
Language

☒ Svenska/Swedish
☐ Engelska/English
☐ _____

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
—

ISRN
LiTH-ISY-EX–08/3973–SE

Serietitel och serienummer **ISSN**
Title of series, numbering —

URL för elektronisk version
`http://www.control.isy.liu.se`
`http://www.ep.liu.se`

Titel
Title

Att lösa reglertekniska problem med Modelica

To use Modelica solving control problems

Författare
Author

Ahmed Ali Abdul-Amir

Sammanfattning
Abstract

Modelica is a multi-domain and equation-based modeling language. Modelica is based on object-oriented principles and non-causal modeling. The language is constructed to facilitate reuse and decompose models. The models and the model library can modified to design a new nonlinear components.

Object-oriented modeling is an excellent way to analyze and study large complex heterogeneous physical systems. The object-oriented modeling approach build on reusing and decomposition of models and non-causal modeling.

Modeling physical systems often leads to a DAE system with index 2 or 3. It is required to use automated symbolic manipulation of the DAE system to do the simulation.

Modelica need a compiler tool to run the simulation. Dymola is the dominating tool on the market. Through a graphic editor the user can easily model and simulate the physical system.

Nyckelord
Keywords

Modelica, Objektorienterad modellering, Icke-kausal modellering, Olinjära system och DAE.

Abstract

Modelica is a multi-domain and equation-based modeling language. Modelica is based on object-oriented principles and non-causal modeling. The language is constructed to facilitate reuse and decompose models. The models and the model library can modified to design a new nonlinear components.

Object-oriented modeling is an excellent way to analyze and study large complex heterogeneous physical systems. The object-oriented modeling approach build on reusing and decomposition of models and non-causal modeling.

Modeling physical systems often leads to a DAE system with index 2 or 3. It is required to use automated symbolic manipulation of the DAE system to do the simulation.

Modelica need a compiler tool to run the simulation. Dymola is the dominating tool on the market. Through a graphic editor the user can easily model and simulate the physical system.

Sammanfattning

Objektorienterad modellering är ett utmärkt sätt att analysera och modellera fysikaliska system. Den ger möjlighet att hantera stora, komplexa och blandade system.

Modelica är ett exempel på ett multidomän modelleringsspråk som är ekvationsbaserat och hanterar modeller från olika fysikaliska domäner. Det är baserat på principer från objektorientering och hanterar icke-kausala problem. Modelicas struktur gör att befintliga modeller kan delas upp i delmodeller som kan designas och testas oberoende av de andra delmodellerna. Det ger överskådlig bild av fysikaliska systemet. Modellbibliotek i Modelica kan enkelt modifieras för att designa nya komponenter.

Modellering av fysikaliska system, speciellt mekaniska eller mekatroniska system ger upphov till DAE system med index 2 eller 3. Det resulterar svårigheter vid simulering av modeller. En automatiserad symbolisk manipulering av DAE systemen behövs för att underlätta simuleringen.

För att simulera modeller skrivna i Modelica, behövs en kompilator och ett simuleringsverktyg. Dymola är det dominerande verktyget för dessa ändamål. Man har tillgång till Modelicas standardbibliotek via ett grafiskt gränssnitt. Vid simulering kontrolleras modellen så att Modelicas syntax är uppfylld. Sedan kompileras och simuleras modellen.

Tack

Jag vill tacka Allah, familjen, mina vänner, mina lärare och alla de som har stött mig genom åren.

Stor tack till min handledare på ISY, Johan Sjöberg, för engagemang och hjälp. Min examinator, Torkel Glad, vill jag också tacka för det intresse han har visat. Ett tack till Thomas Gustafsson för många bra reflektioner och tankar under opositionen.

Linköping, Mars 2008
Ahmed Ali Abdul-Amir
أحمد علي عبدالأمير

Innehåll

Kapitel 1

Inledning

1.1 Bakgrund

Institution för datavetenskap, IDA, på Linköpings universitet driver forskningsprojektet *OpenModelica* [3]. Syftet med projektet är att skapa en stabil miljö där Modelicaspråket [13] kan modelleras, kompileras och simuleras. OpenModelica består av en texteditor, en grafisk editor och en kompilator. Allt är baserat på fri programvara i form av binär- och källkod.

Dagens fysikaliska system är mer och mer komplexa och de består av delar från olika fysikaliska domäner. Modeller beskrivs ofta med differential-algebraiska ekvationer (DAE). Möjligheten att återanvända modeller med obestämd in- och utsignal är önskvärd. Dessa egenskaper har Modelicaspråket.

1.2 Mål

Målet med det här examensarbetet är att:

- Analysera, modellera och simulera ett antal reglertekniska problem.
- Belysa fördelen med objektorienterad modellering.
- Presentera Modelica som nästa generations modelleringsverktyg.
- Designa ett antal olinjära komponent och analysera dem.

1.3 Innehåll

- Kapitel 1 tar upp bakgrunden till examensarbetet. Målet med det och genomförandet.
- Kapitel 2 ger en introduktion till objektorienterad modellering och dess principer.

- Kapitel 3 handlar om Modelicaspråkets historia, struktur samt olika tekniker som Modelica använder sig av. Dessutom beskrivs DAE modeller och hanteringen av algebraiska loopar kort.
- Kapitel 4 tar upp en olinjär elektrisk krets som ett exempel och visar skillnaden mellan modellering i Modelica och Simulink.
- Kapitel 5 tar upp en DC-motor med roterande massor som ett exempel. Dels studeras hur den kan regleras, och dels hur en olinjär fjäder kan designas.
- Kapitel 6 tar upp hjulupphängning på en buss som ett exempel och där visas hur en olinjär fjäder/dämpare kan designas. Implementationen av en ny komponent *inerter* finns med.
- Kapital 7 är en sammanfattning och diskussion av examensarbetet.
- Bilaga A Modelicakod för alla modeller.

1.4 Genomförande

Arbetet har genomförts hos Institutionen för Systemteknik, ISY på Linköping universitet. Examinator var Torkel Glad och handledare var Johan Sjöberg. Dymola och Matlab verktyg var tillgängliga på institution för simulering och analys av problem.

Kapitel 2

Objektorienterad modellering

2.1 Inledning

Matematisk modellering handlar om att beskriva ett system med matematiska ekvationer och formler. Systemet kan vara ett fysikalisk system tex en bil, ett flygplan eller en kärnreaktor som måste simuleras och testas i en artificiell miljö innan den sättes i bruk. Det finns två typer av simulering- och modelleringsspråk idag.

- Domänspecifika simulering- och modelleringsspråk som ADAMS och SIMPACK för mekaniska system, Spice, Saber och VHDL-AMS för elektriska system och Flowmaster för hydrauliska system. Nackdelen är att de inte kan hantera multidomän system. System har ofta svårt eller omöjligt att modifiera de redan definierade komponenterna.

- Generella simulering- och modelleringsspråk som Simulink, ACSL, System Build etc. Dessa verktyg är inte lämpliga för fysikalisk modellering för att de använder blockdiagram och kausal modellering. Det krävs ofta extra arbete för att skriva om de matematiska ekvationerna innan simulering [1] [9].

Fysikaliska system har blivit större med tiden och behovet av ett nytt verktyg för modellering har ökat. Systemens komplexitet har växt och samma system kan bestå av ett antal olika domäner.

Airbus A380 tex. flygplanet är mer avancerad än äldre modeller och består av en rad delsystem tex. motorer, styr-, kyl- och navigeringsystem. Utveckling och testning av de olika delsystemen vill man göra oberoende av varandra. Annars tar det lång tid för att få en färdig produkt. Planets modell består av system från olika domäner, såsom elektrisk, mekanisk och hydraulisk.

Allt detta ledde till att man behövde ett nytt sätt att modellera system för att lösa de ovannämnda problem. Objektorienterad modellering är inte riktig samma

som objektorienterad programmering. Objektorienterad modellering är baserad på ett antal principer [6] [9].

- En modell kan delas i ett antal delar som kallas delmodeller, dessa kan i sin tur också delas.
- Delmodeller kopplade enligt den fysikaliska strukturen för den befintliga modellen.
- Modeller och delmodeller är deklarerade som klasser.
- Modeller ärver till delmodeller.
- Modellernas information är inkapslad. Bara genom väldefinierad gränssnitt sker utbyte av information.
- Modeller beskrivs av ekvationer.

2.2 Kausal och Icke-kausal modellering

Kausal modellering är baserad på algoritmer istället för ekvationer. Beräkningsvägen är redan bestämd och invariabeln måste vara känd för att beräkna utvariabeln.

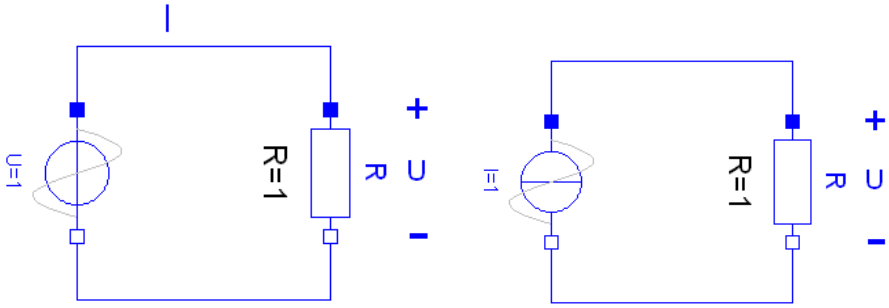
Icke-kausal modellering är baserad på att matematiska ekvationer är skrivna på sin naturliga form, dvs utan tilldelning sker. Ekvationer specificerar inte vilka in- och utvariabler som används. Skillnaden mellan de två sätten ser man på en tillståndsgraf, i icke-kausal modellering finns inga pilar medan pilar finns vid kausal modellering [9].

Fördelen med icke-kausal modellering är att utan villkor kunna återanvända modeller och möjlighet att välja vad som är in- och utvariabel, exempel Ohms lag:

Icke-kausal	Kausal
U=R*I	U:=R*I

För att beräkna ekvationen i det kausala fallet måste R och I vara kända innan U kan beräknas, alltså strömmen I är insignal och spänningen U är utsignal. Vid behov av en resistormodell med spänning U som insignal och ström I som utsignal måste vi designa en ny resistormodell.

Vid icke-kausala fallet räknas spänning U ut mha ekvation $U = R * I$ och strömmen I med ekvationen $I = R/U$. Dvs samma modell av resistor kan ha antingen I eller U som insignal, se figur 2.1.



Figur 2.1. Två olika resistormodeller med olika insignaler.

Kapitel 3

Modelica

3.1 Bakgrund

År 1996 började en grupp forskare från Europa och USA med datavetenskap bakgrund, med att utveckla ett nytt språk för modellering av stora, komplexa och heterogena fysikaliska system. Språket heter Modelica [13]. Idén med det nya språket var att enkelt kunna modifiera modeller och modellbibliotek samt tillämpa objektorienterade modelleringsprinciper. Två år senare kom första versionen, Modelica 1.0. Den senaste versionen Modelica 3.0 släpptes hösten 2007.

3.2 Struktur

Modelica är ett objektorienterat och ekvationsbaserat modelleringsspråk. Modelica har ett standard- och tilläggsbibliotek, se tabell 3.1. Modellbiblioteket kan modifieras genom att ändra de matematiska relationerna i modellens definition. Modellering sker på två sätt:

- Komponentbaserat: genom *dra och släpp* från biblioteket i den grafiska editorn.
- Ekvationsbaserat: genom att skriva de matematiska ekvationerna för modellen i sin naturliga form i en texteditor.

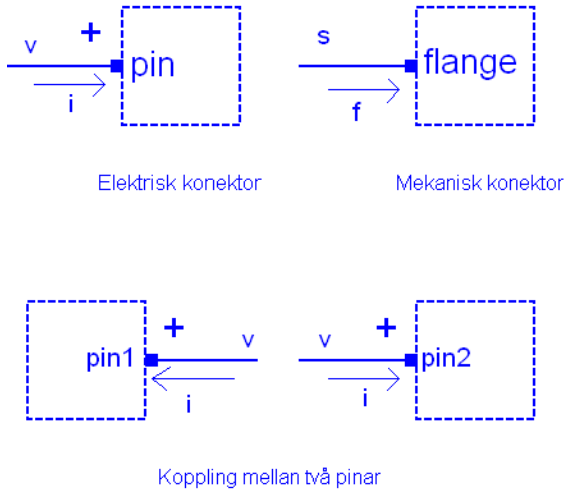
Modelica är ett multidomän språk. Det kan modellera elektriska kretsar, stelkroppsdynamik, hydraulik, termdynamik och kemiska reaktioner. Modelica hanterar icke-kausalitet och stödjer ett antal matematiska former såsom ODE, DAE, Petri nät och bindninggrafer.

Modeller är deklarerade som klasser med väldefinierad gränssnitt som kallas *konnektor* [9]. En konnektor innehåller två kvantiteter, intensitet e och flöde f. Varje domän har sin egen konnektor, i elektrisk domän heter den *Pin* och i mekanisk domän *Flange*, se figur 3.1. Pin och Flange är definierade enligt följande:

```
connector Pin "Pin of an electrical component"
```

```
  SI.Voltage v "Potential at the pin";
  flow SI.Current i "Current flowing into the pin";
end Pin;

connector Flange "1D translational flange"
  SI.Position s "absolute position of flange";
  flow SI.Force f "cut force directed into flange";
end Flange;
```



Figur 3.1. En elektrisk- och mekanisk konnektor samt koppling mellan två elektriska konnektorer.

Det är otillåtet att koppla ihop två komponenter från olika domäner. Energiöverföring mellan två olika domäner sker genom specifika komponenter. Interaktionen mellan två modeller bygger på de kända fysikaliska lagar som Newtons lagar för mekaniska rörelser och Kirchhoffs lagar för ellära. Kopplingen sker enligt två principer:

- Likhet mellan intensiteter e.
- Summa $= 0$ mellan flöden f.

Kopplingen i figur 3.1 implementeras med satsen `connect(pin1,pin2)` och ger, enligt de två principerna, följande:

```
pin1.v = pin2.v
pin1.i + pin2.i = 0
```

Fördelen med Modelica jämfört med blockdiagrambaserad modelleringsspråk som Simulink är att i Simulink sker dataflödet i en bestämd riktning. Detta kallas

kausal modellering och kräver normalt förhandsarbete för att beskriva problemet i ordinära differentialekvationer ODE form i explicit tillståndsform för att kunna lösa det, se ekvationssystem (3.4). I Modelica sker dataflödet i två riktningar och ingen variabel behöver räknas ut manuellt, så kallad *icke-kausal* modellering. Icke-kausalitet hanteras genom att modellen kan ha både intensiteten e såväl flödet f som in- och utsignal.

Den matematiska beskrivningen av modeller leder ibland till differential-algebraiska ekvationer DAE-form, se ekvationssystem (3.2). DAE-formen är mer generell än tillståndsformen och består av differential- och algebraiska ekvationer. Pendelmodellen i (3.1) är ett exempel på en DAE-form.

$$
\begin{aligned}
m\dot{v}_1 &= -Fsin(\varphi) && \text{(3.1a)}\\
m\dot{v}_2 &= Fcos(\varphi) - mg && \text{(3.1b)}\\
\dot{x}_1 &= v_1 && \text{(3.1c)}\\
\dot{x}_2 &= v_2 && \text{(3.1d)}\\
\dot{\varphi} &= \omega && \text{(3.1e)}\\
x_1 &= Lsin(\varphi) && \text{(3.1f)}\\
x_2 &= -Lcos(\varphi) && \text{(3.1g)}
\end{aligned}
$$

där L är längden på pendeln, m är massan, g acceleration, x_1 och x_2 är $x-y$ koordinaterna för spetsen på pendeln, φ är vinkeln. Modellen ovan kan skrivas på formen:

$$
\begin{aligned}
f(y(t), x(t), \dot{x}(t), u(t)) &= 0 && \text{(3.2a)}\\
g(y(t), x(t), u(t)) &= 0 && \text{(3.2b)}
\end{aligned}
$$

där $y(t) = \{\omega, F\}$, $x(t) = \{x_1, x_2, v_1, v_2, \varphi\}$, $u(t) = \{\ \}$ och

$$
f = \begin{cases} m\dot{v}_1 = -Fsin(\varphi) \\ m\dot{v}_2 = Fcos(\varphi) - mg \\ \dot{x}_1 = v_1 \\ \dot{x}_2 = v_2 \\ \dot{\varphi} = \omega \end{cases}, \; g = \begin{cases} x_1 = Lsin(\varphi) \\ x_2 = -Lcos(\varphi) \end{cases}
$$

Ett exempel på ODE form är Newton's kraftlag.

$$
\begin{aligned}
\dot{v}(t) &= \frac{F(t)}{m} && \text{(3.3a)}\\
\dot{s}(t) &= v(t) && \text{(3.3b)}
\end{aligned}
$$

En generell struktur för ordinära differential ekvationssystem är så kallad explicit tillståndsform:

$$
\begin{aligned}
\dot{x}(t) &= f(x(t), u(t)) && \text{(3.4a)}\\
y(t) &= g(x(t), u(t)) && \text{(3.4b)}
\end{aligned}
$$

där $u(t)$ är insignalerna $y(t)$ är utsignalerna och $x(t)$ är tillståndsvariablerna.

3.3 Verktyg

Det finns två kommersiella verktyg till Modelica. Dymola som utvecklas av Dynasim i Lund och MathModelica som utvecklas av MathCore i Linköping.

3.3.1 Dymola

Dymola (*Dynamic Modeling Laboratory*) är det mest dominerande simuleringsverktyg för Modelicaspråket. Dymola används mest för automation inom industrin. Historien går tillbaks till 1978 vid institutionen för reglerteknik på Lunds universitet då Hilding Elmqvist presenterade sin doktorsavhandling [7] om en metod för komponentbaserad modellering och simulering med icke-kausala matematiska ekvationer. Dymola består av följande:

- En Modelica kompilator med en *symbolic optimizer* för reducering av storleken på ekvationssystemet.
- En simuleringsmiljö med numerisk lösare för hybrida DAE modeller.
- Ett grafiskt gränssnitt där modeller kan kopplas ihop.
- En texteditor för redigering av modeller.

3.3.2 MathModelica

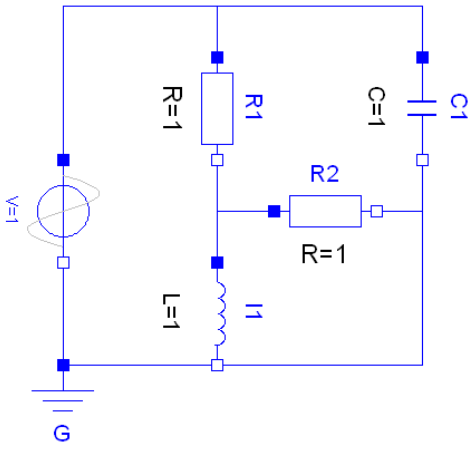
MathModelica är en interaktiv utvecklingsmiljö för avancerad modellering och simulering av fysikaliska system. Allt detta är baserad på *Mathematica*. MathModelica består av följande:

- En grafisk editor för design av modeller från ett komponentbibliotek.
- En *Notebook* för programmering, dokumentation och simulering med *Mathematica*.
- En simuleringsmiljö för simulering och kurvplottning.
- Som tillval finns ett gränssnitt för 3D grafik och ett gränssnitt till CAD-system.

3.4 Modelica kompilatorn

Kompilering av modeller i Modelica börjar med omskrivning av modellen till en så kallad *Flattened model*. Den omskrivna modellen innehåller alla ekvationer, variabler, parametrar och konstanter enligt [5]. Ett exempel är en elektrisk krets, se figur 3.2, som har en omskriven modell på tabell 3.2. Där står de enklaste ekvation form som beskriver modellen och vilken ekvation som tillhör vilken komponent. Man känner igen de två principerna för ihopkoppling modeller, likhet och summa = 0.

Därefter tillämpar kompilatorn ett antal metoder för att optimera modellen inför simulering.

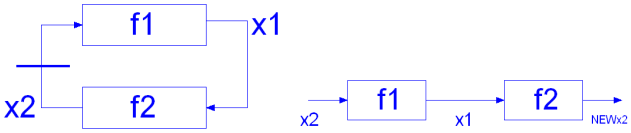


Figur 3.2. En elektrisk krets.

3.4.1 Minimering av antalet algebraiska loopar

En av fördelarna med Modelica är att den hanterar problemet med någon algebraiska loopar. En algebraisk loop [2] är en sluten slinga som inte innehåller integrator, dvs den saknar alltså dynamik. Ekvationssystemet (3.5) består av två ekvationer som kan illustreras med figur 3.3. Det syns tydligt att det inte går att lösa ut x_1 ur den första ekvationen innan x_2 har beräknats och tvärtom i den andra ekvationen, då det finns ett ömsesidigt beroende mellan dessa.

$$x_1 = f_1(x_2) \tag{3.5a}$$
$$x_2 = f_2(x_1) \tag{3.5b}$$



Figur 3.3. System med algebraisk loop före och efter *Tearing*.

Ett sätt att hantera problemet är att använda tekniken *Tearing* [8] i Modelica. Denna metod går ut på att bryta slingan så att beräkningsalgoritmen blir:

```
NEWx2 = INITx2
REPEAT
x2 = NEWx2
```

```
x1 = f1(x2)
NEWx2 = f2(x1)
UNTIL converged(NEWx2-x2)
```

Notera att iterationen bara utförs på x_2 och problemets dimension har reducerats till dimension 1. Den enda okända variabeln är x_2 och den olinjära ekvationen som ska lösas är:

$$f_2(f_1(x_1)) = x_2 \tag{3.6}$$

För större modeller med fler variabler används en grafteoretisk algoritm *Trajan´s algoritm* som går ut på att skriva ett ekvationssystem på formen *Block Lower Triangular* PLT för att bestämma vilken variabel som ska lösas ut för att minimera antalet algebraiska loopar [8].

3.4.2 Blandad symbolisk och numerisk DAE lösning

Ett DAE index är det minsta antalet derivationer som krävs för att $\dot{x}(t)$ ska kunna lösas ut som en funktion av $x(t)$, $y(t)$ och $u(t)$ dvs för att kunna skriva modellen på tillståndsform. En ODE har index 0 då den redan är på tillståndsform och därför behövs ingen derivering.

Många modeller, speciellt mekaniska eller mekatroniska modeller, har ett högt index, tex 2 eller 3. Det ger upphov till svårigheter vid numeriska beräkningar. Helst vill man ha en DAE med index max 1. En pendelmodell beskrivs av följande DAE system:

$$m\ddot{x}_1 = -\frac{x_1}{L}F \tag{3.7a}$$

$$m\ddot{x}_2 = -\frac{x_2}{L}F - mg \tag{3.7b}$$

$$x_1^2 + x_2^2 = L^2 \tag{3.7c}$$

För att reducera systemetsindex = 3, krävs det två deriveringar av $x_1^2+x_2^2 = L^2$ för att nå index = 1.

$$x_1^2 + x_2^2 = L^2 \ , \quad 2x_1\dot{x}_1 + 2x_2\dot{x}_2 = 0 \ , \quad 2\dot{x}_1\dot{x}_1 + 2\dot{x}_2\dot{x}_2 + 2x_1\ddot{x}_1 + 2x_2\ddot{x}_2 = 0$$

Modelica använder automatiserad symbolisk manipulering av DAE system för att underlätta simuleringen.

- *Pantelides Algoritm* används för att beräkna index och hur många gånger varje ekvation ska deriveras så att DAE systemet får index 1 eller 0.
- Metoden *dummy derivatives* används för överbestämda ekvationssystem (fler ekvationer än variabler).
- Dynamisk eller statisk tillståndsvariabel väljare för att få ett välbestämt system.
- När ekvationssystemet har manipulerats symboliskt, används en numerisk lösare, typ Euler eller Runge-Kutta för ODE och DASSL för DAE.

Modelica.UsersGuide	Användarhandledning.
Modelica.Blocks	Grunläggande in-och utsignal för reglerteknik packet.
Modelica.Constants	Matematiska och fysikaliska konstanter.
Modelica.Electrical	Bibliotek för elektriska modeller.
Modelica.Icons	Ikon definitioner.
Modelica.Math	Matematiska funktioner och matris operationer.
Modelica.Mechanics	Bibliotek för mekaniska modeller.
Modelica.Media	Bibliotek för medium modeller(luft, ideell gas och vatten).
Modelica.SIunits	Definitiner på typer och enheter enligt ISO 31-1992.
Modelica.StateGraph	Bibliotek för diskreta händelser och reaktiva system.
Modelica.Thermal	Bibliotek för termiska system.
Modelica.Utilities	Användbara funktioner för skript.

Tabell 3.1. Modelica standard bibliotek

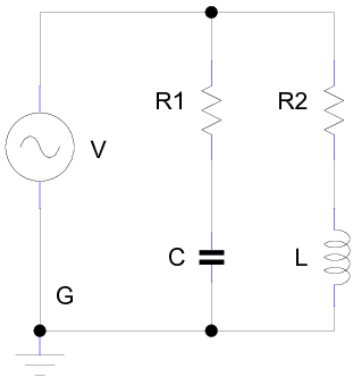
No.	Equation	Origin
1	R1.v=R1.p.v-R1.n.v	OnePort
2	0 = R1.p.i + R1.n.i	OnePort
3	R1.i = R1.p.i	OnePort
4	R1.R * R1.i = R1.v	Resistor
5	R2.v = R2.p.v - R2.n.v	OnePort
6	0 = R2.p.i + R2.n.i	OnePort
7	R2.i = R2.p.i	OnePort
8	R2.R * R2.i = R2.v	Resistor
9	C1.v = C1.p.v - C1.n.v	OnePort
10	0 = C1.p.i + C1.n.i	OnePort
11	C1.i = C1.p.i	OnePort
12	C1.i = C1.C * der(C1.v)	Capacitor
13	I1.v = I1.p.v - I1.n.v	OnePort
14	0 = I1.p.i + I1.n.i	OnePort
15	I1.i = I1.p.i	OnePort
16	I1.L * der(I1.i) = I1.v	Inductor
17	G.p.v = 0	Ground
18	V.v = V.p.v - V.n.v	OnePort
19	0 = V.p.i + V.n.i	OnePort
20	V.i = V.p.i	OnePort
21	V.src.outPort.signal_1 = V.src.p offset_1 +(iftime < V.src.p startTime_1 then0else V.src.p amplitude_1*sin(2 * V.src.pi * V.src.p freqHz_1 *(time-V.src.p startTime_1) + V.src.p phase_1))	SineVoltage
22	V.v = V.src.outPort.signal_1	SineVoltage
23	C1.n.i + R1.n.i + V.p.i = 0	Connect
24	R1.n.v = C1.n.v	Connect
25	V.p.v = C1.n.v	Connect
26	C1.p.i + G.p.i + I1.p.i + R2.n.i + V.n.i = 0	Connect
27	G.p.v = C1.p.v	Connect
28	I1.p.v = C1.p.v	Connect
29	R2.n.v = C1.p.v	Connect
30	V.n.v = C1.p.v	Connect
31	I1.n.i + R1.p.i + R2.p.i = 0	Connect
32	R1.p.v = I1.n.v	Connect
33	R2.p.v = I1.n.v	Connect

Tabell 3.2. Omskriven modell för figur 3.2

Kapitel 4

Elektrisk krets

4.1 Linjär krets

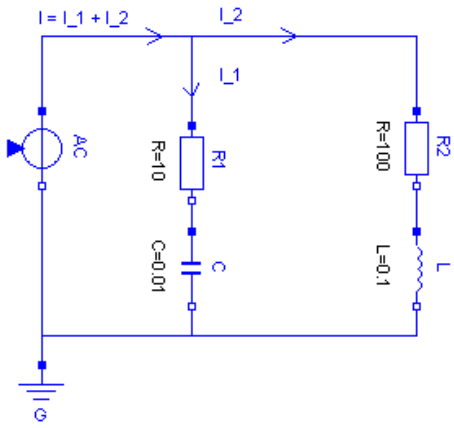


Figur 4.1. Linjär elektrisk krets

En linjär elektrisk krets ska modelleras. Linjär betyder att kretsen består av linjära komponenter såsom grundläggande elektriska komponenter tex motstånd, induktanser och kondensatorer. Kretsen ska designas med hjälp av två olika tekniker, objektorienterat (Modelica) och blockdiagrambaserat (Simulink). En jämförelse mellan de två ska göras för att visa för- och nackdelar med respektive teknik.

4.1.1 Modelica design

Det här linjära exemplet [12] illustrerar fördelarna med Modelica. Modellen består av ett antal elektriska standard komponenter, se figur 4.1. Det finns två resistorer *R1 R2*, en induktans L, en kapacitans C, en spänningskälla V och en jord G. De här komponenterna är tillgängliga i Modelicas standard bibliotek och kan med



Figur 4.2. Kretsmodellen skapad med Modelica

hjälp av den grafiska editorn enkelt kopplas ihop, se figur 4.2. Modelicakoden för modellen finns i Bilaga A, tabell A.1.

Koden består av två delar, *deklarations-* och *kopplingsdelen*. I den första delen står alla komponenter definierade och initierade, tex *R1*. Komponenten är av typen *Resistor* och har initierats till värdet *10*. Grundläggande elektriska komponenter ligger under biblioteket *Modelica.Electrical.Analog.Basic*

I den andra delen av koden beskrivs hur komponenterna är kopplade. Med satsen *connect* kopplar man två pinnar. *R1*:s positiva pin är kopplad till *R2*:s postiva pin genom satsen `connect(R1.p, R2.p)` osv. Typen *Resistor* definieras enligt följande:

```
model Resistor "Ideal linear electrical resistor"
  extends Interfaces.OnePort;
  parameter SI.Resistance R=1 "Resistance";
equation
  R*i = v;
end Resistor;
```

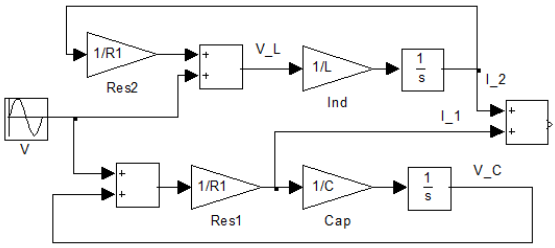
Man ser att typen ärver (*extends*) `Interfaces.OnePort` och parametern R definieras enligt `SI.Resistance`. Själva kopplingsdelen består av en enda ekvation, `R*i = v` som är Ohms lag. Tittar vi på `OnePort`s definition ges denna av:

```
partial model OnePort
  "Component with two electrical pins p and n and current i from p to n"
SI.Voltage v "Voltage drop between the two pins (= p.v - n.v)";
  SI.Current i "Current flowing from pin p to pin n";
  PositivePin p
  "Positive pin (potential p.v > n.v for positive voltage drop v)";
```

```
  NegativePin n "Negative pin";
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Vi har de två kvantiteterna spänningen v och strömmen i. Spänningen över komponenten är potentialskillnaden mellan den positiva och negativa ingången på komponenten, som framgår av ekvationen `v = p.v - n.v`. Den andra ekvationen är Kirchhoffs strömlag. Tredje ekvationen säger att strömmen som går igenom porten är den som kommer från den positiva ingången.

4.1.2 Simulink design



Figur 4.3. Kretsmodell skapad med Simulink

Modellering av kretsen i Simulink kräver att alla matematiska ekvationer mellan elementerna är kända vilket ger ekvationssystemet (4.1). För att sedan kunna simulera systemet i Simulink måste det skrivas om på tillståndsform med V_C och I_L som tillståndsvariabler, se ekvationssystem (4.2). Utsignalen i detta fall är I.

$$\frac{dV_C}{dt} = \frac{I_C}{C} \tag{4.1a}$$
$$\frac{dI_L}{dt} = \frac{V_L}{L} \tag{4.1b}$$
$$0 = V_{R_1} - R_1 I_{R_1} \tag{4.1c}$$
$$0 = V_{R_2} - R_2 I_{R_2} \tag{4.1d}$$
$$0 = I_{R_1} - I_C \tag{4.1e}$$
$$0 = I_{R_2} - I_L \tag{4.1f}$$
$$0 = V - V_{R_1} - V_C \tag{4.1g}$$
$$0 = V - V_{R_2} - V_L \tag{4.1h}$$
$$0 = I - I_{R_1} - I_{R_2} \tag{4.1i}$$

$$\frac{dV_C}{dt} = \frac{R_1(V - V_C)}{C} \tag{4.2a}$$

$$\frac{dI_L}{dt} = \frac{V - I_L R_2}{L} \tag{4.2b}$$

$$0 = I - I_L - R_1(V - V_C) \tag{4.2c}$$

När modellen är på tillståndsform kan den realiseras med Simulink. Modellen kommer att ser ut som i figur 4.3.

4.1.3 Observation

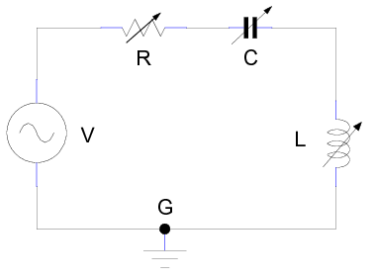
Utifrån Simulinksdesign av modellen som är blockdiagrambaserad med kausal modellering. Observera följande:

- Dataflödet sker i en bestämd riktning, *kausalitet*. För att beräkna V_L måste $R_2 I_2$ och V vara kända.
- Ekvationssystemet (4.1) måste skrivas om på tillståndsform enligt (4.2) för att kunna realiseras. För en sådan enkel och linjär modell tar det inte så mycket tid men för olinjära och stora modeller behövs mycket arbete och ibland är det omöjligt.
- Kretsschemat för komponenterna känns knappt igen. De olika komponenterna syns inte och kopplingen mellan blockschemat, figur 4.3 och kretsen i figur 4.1 försvinner.
- Modifiering av kretsen är svår. Ska man lägga till en ny komponent eller använda modellen som en delmodell i ett annat system måste hela ekvationssystemet skrivas om.
- Återanvändning med bestämd in- och utsignal av komponenter tex. Res_2, som representerar R_2, har strömmen I_2 som insignal och spänning V_{R_2} som utsignal medan Res_1 är tvärtom vilket betyder att vi behöver två olika komponenter för typen resistor, se figur 4.3.

4.2 Enkel olinjär krets

Vissa kretsar består av olinjära komponenter för att vissa system inte är linjära. Detta gör att en del av systemen är svåra eller omöjliga att beskriva på tillståndsform. Det ställer också hårda krav på modellerings- och simuleringsverktyg. I de senare exemplen syns fördelen med ekvationsbaserade och objektorienterade modelleringsverktyg som Modelica.

Kretsen består av en spänningskälla V, en olinjär induktans L, en olinjär kapacitans C, en olinjär resistans R och en jord G, se figur 4.4. De olinjära matematiska

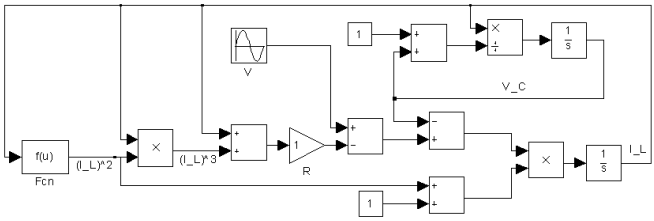


Figur 4.4. Enkel olinjär kretsmodell.

relationer mellan parameterna för varje komponent framgår av (4.3):

$$\frac{d\phi}{dt} = V_L \tag{4.3a}$$
$$\phi = \arctan(I_L) \tag{4.3b}$$
$$\frac{dV_C}{dt} = \frac{I_C}{1+V_C} \tag{4.3c}$$
$$V_R = R(I_R + I_R^3) \tag{4.3d}$$

4.2.1 Simulink design



Figur 4.5. Simulink design för den olinjära kretsmodellen i (4.3).

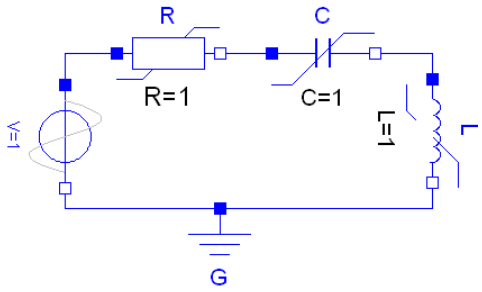
Om Kirchhoffs spänning- och strömlag används för hela kretsen och efter antal förenklingssteg fås ekvationssystemet som en tillståndsform, se ekvationssystem (4.4). Tillståndsvektorn är $x = \{I_L, V_C\}$ och insignalen är $u = \{V\}$.

$$\frac{dI_L}{dt} = (1+I_L^2)(V - V_C - R(I_L + I_L^3)) \tag{4.4a}$$
$$\frac{dV_C}{dt} = \frac{I_L}{1+V_C} \tag{4.4b}$$

Modellen ovan går att skriva på tillståndsform och på så sätt kan den realiseras med Simulink även om den består av olinjära komponenter. Detta går i det här

fallet och många andra fall men det kan vara svårt. Simulinkmodellen för kretsen syns i figur 4.5.

4.2.2 Modelica design



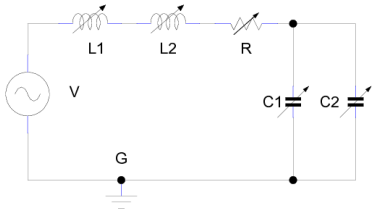
Figur 4.6. Modelica design för den olinjära kretsmodellen i (4.3).

Utifrån definitionen av varje olinjär komponent i (4.3), designas en ny olinjär modell och läggs till i modellbiblioteket. En olinjär resistans, $V_R = R(I_R + I_R^3)$ kan designas som en ny modell enligt Modelicakod i Bilaga A, tabell A.4. Sedan kopplar man ihop dem och simulerar modellen.

En realisering av modellen i Modelica syns i figur 4.6 vilket är mer tydlig och bunden till den fysikaliska kretsen än realiseringen i Simulink.

Antalet olinjära komponeneter ökas i nästa modell och vi ska se hur ökningen påverkar realiseringen av modellen.

4.3 Avancerad olinjär krets



Figur 4.7. Avacerad olinjär kretsmodell.

Den olinjära kretsen utökas genom att lägga till flera olinjära komponenter, se figur 4.7, där L_1 L_2 är olinjära induktanser C_1 C_2 olinjära kapacitanser och R olinjär resistans. Matematiska uttrycken för varje olinjär komponent framgår av (4.5).

$$
\begin{aligned}
V_R &= R(I_R + I_R^3) && (4.5a)\\
\frac{dI_{L_1}}{dt} &= V_{L_1} + V_{L_1}^3 && (4.5b)\\
\frac{dI_{L_2}}{dt} &= \arctan(V_{L_2}) + V_{L_2} && (4.5c)\\
\frac{dV_{C_i}}{dt} &= \frac{I_{C_i}}{1 + V_{C_i}},\ i = 1, 2 && (4.5d)
\end{aligned}
$$

4.3.1 Simulink design

Kirchhoffs lagar för hela modellen ger ekvationssystemet (4.6) och för att simulera modellen med Simulink krävs att (4.6) kan skrivas om på tillståndsform. Efter ett antal förenklingssteg fås (4.7) som inte är på tillståndsform och man kan inte göra göra ytterligare förenklingar. Sådana modeller kan inte realiseras med Simulink då verktyget kräver att modellen skrivas på tillståndsform innan den kan simuleras. En annan lösning skulle kunna vara är att kunna designa olinjär komponenter och lägga till modellbibliotek men den lösningen är omöjligt att tillämpa i Simulink.

$$
\begin{aligned}
0 &= V_{C_1} - V_{C_2} && (4.6a)\\
0 &= I_{L_1} - I_{L_2} && (4.6b)\\
0 &= I_{L_1} - I_R && (4.6c)\\
0 &= I_{L_1} - I_{C_1} - I_{C_2} && (4.6d)\\
0 &= V - V_{L_1} - V_{L_2} - V_{C_{1,2}} - V_R && (4.6e)\\
0 &= V_R - R(I_R + I_R^3) && (4.6f)\\
\frac{dI_{L_1}}{dt} &= V_{L_1} + V_{L_1}^3 && (4.6g)\\
\frac{dI_{L_2}}{dt} &= \arctan(V_{L_2}) + V_{L_2} && (4.6h)\\
\frac{dV_{C_1}}{dt} &= \frac{I_{C_1}}{1 + V_{C_1}} && (4.6i)\\
\frac{dV_{C_2}}{dt} &= \frac{I_{C_2}}{1 + V_{C_2}} && (4.6j)
\end{aligned}
$$

$$0 = V_{C_1} - V_{C_2} \tag{4.7a}$$
$$0 = I_{L_1} - I_{L_2} \tag{4.7b}$$
$$0 = I_{L_1} - I_R \tag{4.7c}$$
$$0 = I_{L_1} - I_{C_1} - I_{C_2} \tag{4.7d}$$
$$0 = V - V_{L_1} - V_{L_2} - V_{C_1} - V_R \tag{4.7e}$$
$$0 = V_R - R(I_{L_1} + I_{L_1}^3) \tag{4.7f}$$
$$0 = V_{L_1} + V_{L_1}^3 - \arctan(V_{L_2}) - V_{L_2} \tag{4.7g}$$
$$0 = \frac{I_{C_1} - I_{C_2}}{1 + V_{C_1}} \tag{4.7h}$$
$$\frac{dI_{L_1}}{dt} = \frac{1}{2}(\arctan(V - V_{L_1} - V_{C_1} + R(I_{L_1} + I_{L_1}^3)) + V_{L_1}^3 + V - V_{C_1} + R(I_{L_1} + I_{L_1}^3)) \tag{4.7i}$$
$$\frac{dV_{C_1}}{dt} = \frac{I_{L_1}}{2(1 + V_{C_1})} \tag{4.7j}$$

4.3.2 Modelica design

I Modelica kan detta lösas på två olika sätt.

Ekvationsbaserad lösning

Ett ekvationssystem som beskriver kretsen, (4.6) eller (4.7), väljs och implementeras som en modell mha en texteditor. Ekvationssystemet (4.6) väljs nedan som ett exempel på det.

```
model Nonlinear71
  Modelica.SIunits.Voltage VC1;
  Modelica.SIunits.Voltage VC2;
  Modelica.SIunits.Voltage VL1;
  Modelica.SIunits.Voltage VL2;
  Modelica.SIunits.Voltage VR;
  Modelica.SIunits.Current IC1;
  Modelica.SIunits.Current IC2;
  Modelica.SIunits.Current IL1;
  Modelica.SIunits.Current IL2;
  Modelica.SIunits.Current IR;
  parameter Real R=1;
equation
  0 = VC1 - VC2;
  0 = IL1 - IL2;
  0 = IL1 - IR;
  0 = IL1 - IC1 - IC2;
  0 = sin(time) - VL1 - VL2 - VC1 - VR;
```

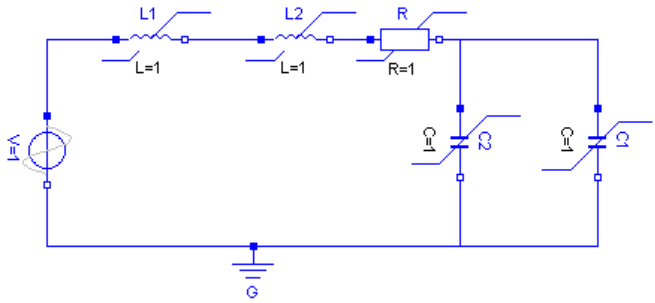
```
  0 = VR - R*(IR + IR^3);
  der(IL1) = VL1^3 + VL1;
  der(IL2) = arctan(VL2) + VL2;
  der(VC1) = IC1/(1 + VC1);
  der(VC2) = IC2/(1 + VC2);
end Nonlinear71;
```

Detta är en metod som enkelt hanterar det olinjära fallet och ingen omskrivning av modellekvationer på tillståndsform behövs.

Komponentsbaserad lösning

Komponenterna är redan designade i förra exemplet, Enkel olinjär krets. Sedan kopplar man ihop dem som i figur 4.8 och simulerar modellen.

Figur 4.9 visar spänningen V som insignal och strömmen i den olinjära kapacitansen C_2 som utsignal. Modelicakod för hela modellen finns i Bilaga A, tabell A.3 och A.4.



Figur 4.8. Modelica design för den olinjära kretsmodellen i (4.5).

4.4 Slutsats

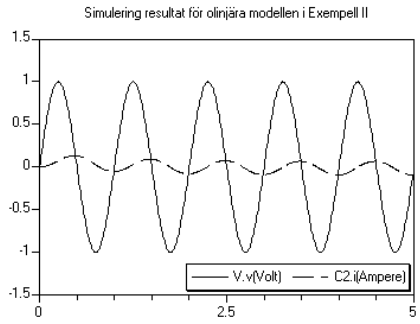
Utifrån Modelicas komponentsbaserade modellering av modeller som är objektorienterad med icke-kausal modellering kan man dra följande slutsatser:

- Dataflödet sker i två riktningar dvs *icke-kausalt*. In- och utsignal behöver inte specificeras.
- Inget förhandsarbete för att härleda de ekvationer som ska simuleras. Modelica stödjer både ODE- och DAE-form.
- Modelicas realisering har direkt koppling till den fysikaliska kretsen. Stora och komplexa system blir mer överskådliga.

- Modifiering av kretsen är inte svårt. Nya komponenter kan läggas till utan problem eller hela modellen kan ingå som en delmodell i ny supermodell.
- Återanvändning av komponenter tex. L_1 i figur 4.7 kan läggas till även om det ger en annan kausalitet för vissa modeller.
- Nya olinjära komponenter kan designas och läggas till modellbibliotek.

En nackdel med Dymola verktyget är att det tillämpar den grafteoretiska algoritmen *Pantelides algoritm*, för att bestämma vilken tillståndsvariabel som ska lösas först. Pantelides algoritm fungerar i många praktiska situationer. Dock kan den i vissa fall misslyckas med att indexreducera korrekt. För den olinjära elektriska kretsen i figur 4.7 upptäcktes felmeddelande vid kontroll av modellen. För vissa konfigurationer av C_1, C_2 osv fås en modell som inte ger något felmeddelande vid kontroll, men som inte går att simulera. En lösning var då att byta plats på de komponenter som ger felmeddelandet.

En mer exakt metod för lösning av DAE system är *Kunkel-Mehrmanns algoritm* [10]. Nackdelen med metoden är att den är komplicerad.

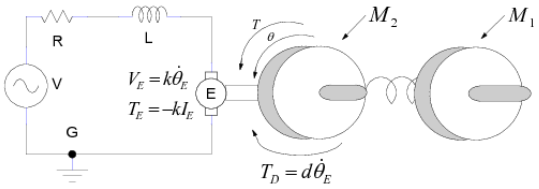


Figur 4.9. Simulering av den olinjära kretsmodellen i (4.5).

Kapitel 5

DC-motor

5.1 Teori



Figur 5.1. En DC-motor

Ett klassisk exempel på reglerteknik är styrning av en DC-motor. Motorn är ett typisk exempel på en multidomän modell då den innehåller både en elektrisk och en mekanisk del [4] .

En DC-motor består av en spänningsgenerator V, en resistans R, en induktans L och en elektromotorisk kraft E, se figur 5.1. Till motorn kopplas två roterade massor M_1 och M_2. En ideal fjäder/dämpare, olinjär fjäder eller fjäder/dämpare med glapp kan kopplas mellan M_1 och M_2. Till andra änden av M_1 kopplas en störning signal W i form av ett belastande moment. Massan M_2 har en friktion med friktionskonstant d. Insignalen är spänningen V och utsignalen är momentet T för fjädern i den komponent som är kopplad mellan massorna M_1 och M_2, se figur 5.5.

Relationen mellan V, R och L framgår av (5.1).

$$V = V_R + V_L + V_E \tag{5.1a}$$
$$V_R = RI_R \tag{5.1b}$$
$$V_L = L\frac{dI_L}{dt} \tag{5.1c}$$
$$I = I_R \tag{5.1d}$$
$$I_R = I_L \tag{5.1e}$$

Relationen mellan V, I, vinkelhastigheten $\dot{\theta}$ och momentet $\dot{\theta}$ för E framgår av (5.2).

$$T_E = -kI_E \tag{5.2}$$
$$V_E = k\dot{\theta}_E \tag{5.3}$$

En elektromotorisk kraft E motsvarar en gyrator i en bindningsgraf [11]. Ekvationssystemet (5.2) visar hur insignalerna, spänning och ström resulterar i ett moment och en vinkelhastighet som utsignaler. Friktionen på M_2 modelleras som en dämpare med dämpningskonstant d.

$$T_D = d\dot{\theta}_D \tag{5.4}$$

Relationen mellan tröghetsmomentet J, vinkelacceleration $\ddot{\theta}$ och momentet T för M_2 framgår av (5.5).

$$J_{M_2}\ddot{\theta}_{M_2} = T_{in} + T_{ut} \tag{5.5}$$

Ekvation (5.6) klargör hur komponenten som kopplas mellan massorna M_1 och M_2 får sin vinkel och moment. Momenten T beräknas enligt (5.6), linjär fjäder/dämpare komponent, där c är fjäderkonstant och d är dämpningskonstant.

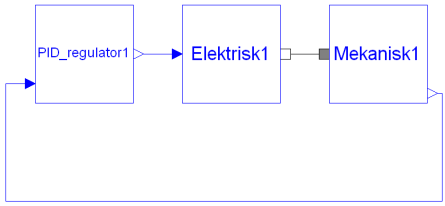
$$T = c\,\theta + d\,\dot{\theta} \tag{5.6}$$

5.2 Design och implementering

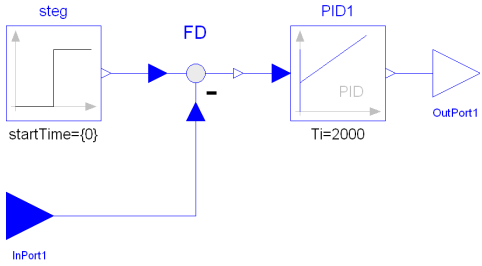
En PID regulator ska designas så att stigtiden är mindre än 4 s med en översläng som är mindre än 5% då utsignalen är momentet T.

Hela modellen kan delas upp som tre delmodeller. En PID-regulator, en elektrisk och en mekanisk del, se figur 5.2. Det ger bättre kontroll och och gör det lättare att analysera problemet och lokalisera eventuella fel. Antal delmodeller ger oss ett mått på modellens komplexitet. Ju högre komplexitet, desto fler delmodeller behövs. Vid behov delar man också delmodeller dock inte i detta fallet, ty de är tillräckligt enkla.

Delmodellens gränssnitt består av två signaler, en insignal och en utsignal. Den är designad som en svart låda där modellens information är inkapslad. Alla modellerna designas oberoende av varandra. Vad som är viktigt är att ett väldefinierat gränssnitt fås. Modelicakoden för DC-motorn finns i Bilaga A, tabell A.5.



Figur 5.2. DC-modell skapad med Modelica



Figur 5.3. PID-regulator delmodell

5.2.1 PID-regulator delmodell

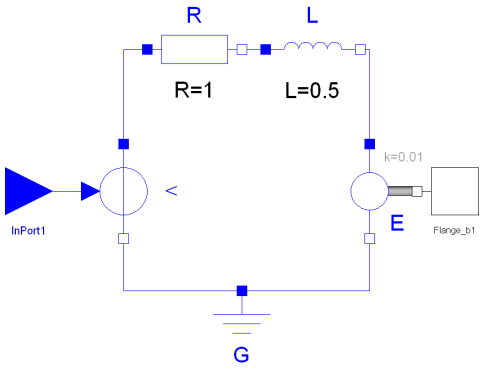
Ekvationen för PID-regulatorn ges av:

$$u = k\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\dot{e}(t)\right) \tag{5.7}$$

där k är förstärkningen, T_i är konstant för den integrerande delen och T_d konstanten för den deriverande delen. En PID-regulator har redan designats i Modelica och den finns i biblioteket `Modelica.Blocks.Continuous`. Hela delmodellen är designad genom att plocka och koppla ihop de komponenter som behövs. De är redan definierade och kan hämtas från Modelicas standardbibliotek. Delmodellen, se figur 5.3, består av ett steg *steg* en återkoppling *FD*, en PID komponent *PID1* samt in- och utsignal *InPort1* och *OutPort1*. Modelicakoden för delmodellen finns i Bilaga A, tabell A.6.

5.2.2 Elektrisk delmodell

Elektriska delmodellen består av ett antal elektriska komponenter, vilka återfinns i biblioteket `Modelica.Electrical.Analog.Basic`. Delmodellens gränssnitt består

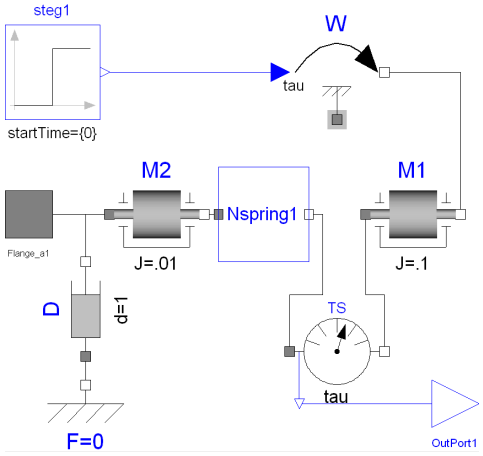


Figur 5.4. Elektrisk delmodell

av en insignal, *InPort1* från delmodell PID-regulator och en utsignal *Flangeb1* i form av moment som kopplas till den mekaniska delmodellen. Modelicakoden för delmodellen finns i Bilaga A, tabell A.7.

5.2.3 Mekanisk delmodell

Den mekanisk delmodellen, se figur 5.5, består av en dämpare D, två massor M_1, M_1, en olinjär fjäder *Nspring1*, en momentsensor TS, en störning moment W samt in- och utsignal *Flange-b1 OutPort1*. Alla komponenter utom den olinjära fjädern *Nspring1* finns i biblioteket `Modelica.Mechanics.Rotational`. Modelicakoden för delmodellen finns i Bilaga A, tabell A.8.



Figur 5.5. Mekanisk delmodell

- Olinjär fjäder.
  En linjär fjäder definieras enligt Modelica syntax:

```
model Spring "Linear 1D rotational spring"
  extends Interfaces.Compliant;
  parameter Real c(final unit="N.m/rad", final min=0) "Spring constant";
  parameter SI.Angle phi_rel0=0 "Unstretched spring angle";
equation
  tau = c*(phi_rel - phi_rel0);
end Spring;
```

  där c är en fjäderkonstant, *phirel0* är vinkeln för den osträckta fjädern och *phirel*är den nuvarande vinkeln på fjädern. Koden kan modifieras till en olinjär fjäder enligt:

$$T = c_1\,\theta^3 + c_2\,\theta$$

  genom att modefiera ekvationen:`tau = c*(phi_rel - phi_rel0)` till
  `tau = c1*(phi_rel - phi_rel0)^3 + c2*(phi_rel - phi_rel0)`
  där c_1 och c_2 är fjäderkonstanter.

  Olinjäriteten valdes i det här fallet som ett tredjegradspolynom. Modelica-koden för fjädern finns i Bilaga A, tabell A.13. `Nspring1` kan ersättas av ideal fjäder/dämpare eller fjäder/dämpare med glapp. Modellen testas med olika värden på regulatorns parameter och för värdena `c1=0.1 N.m/rad`, `c2=0.4 N.m/rad`, `k=450`, `Ti=10000` och `Td=0.3` fås ett stegsvar enligt figur 5.6.

- Ideal Fjäder/Dämpare.

  Komponenten är en fjäder parallellkopplad med en dämpare. Matematiska relationen mellan dess parametrar, moment T och vinkel hastighet $\dot{\theta}$ är:

$$T = c\,\theta + d\,\dot{\theta}$$

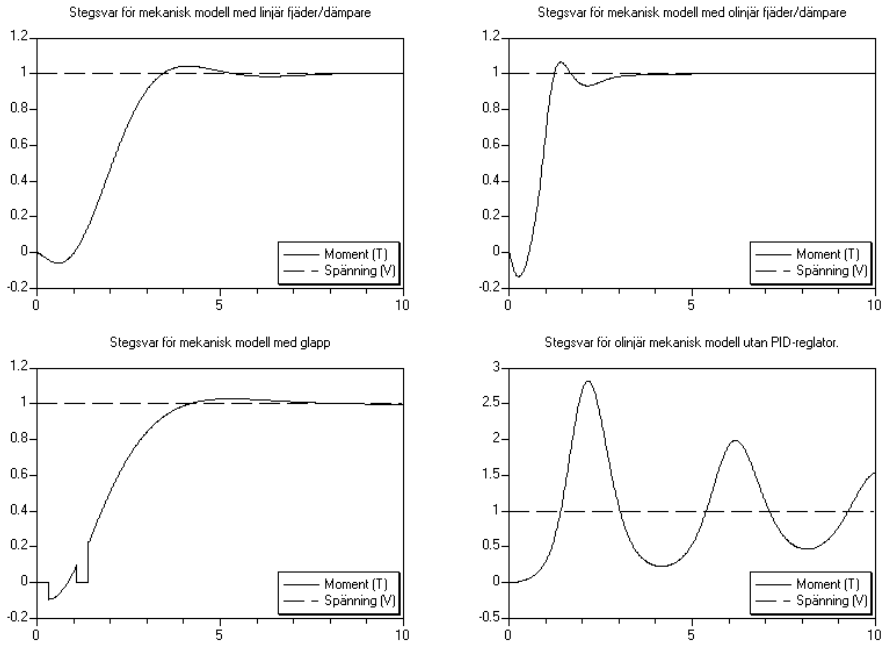
  Med parametervärdena `c=0.1 N.m/rad` och `d=0.01 N.m.s/rad` fås ett stegsvar enligt figur 5.6.

- Fjäder/Dämpare med glapp.

  Glapp kommer från slitage och den gör att komponenten inte reagerar på små ändringar inom ett visst område.

$$T = \begin{cases} c\left(\theta - \frac{b}{2}\right) + d\,\dot{\theta} & \theta > \frac{b}{2} \\ c\left(\theta + \frac{b}{2}\right) + d\,\dot{\theta} & \theta < \frac{-b}{2} \\ c\,\theta + d\,\dot{\theta} & \text{F.Ö.} \end{cases}$$

  Med parametervärdena `c=0.1 N.m/rad`, `d=0.05 N.m.s/rad` och `b=1 rad` fås ett stegsvar enligt figur 5.6. I simuleringen syns hur störningen W börjar verka innan regulatorn kompenserar.



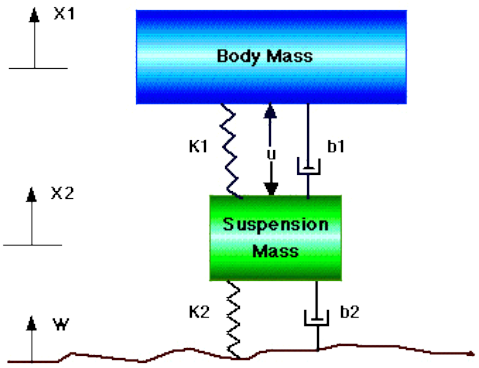
Figur 5.6. Stegsvar för fyra fall av DC-motor.

Kapitel 6

Hjulupphängning på en buss

6.1 Teori

I det har exemplet behandlas en fjädring på en buss [4]. För att förenkla problemet analyseras 1/4-bussmodell, dvs en modell för varje hjul, se figur 6.1. Modellen



Figur 6.1. En 1/4 buss modell enligt [4]

består av:

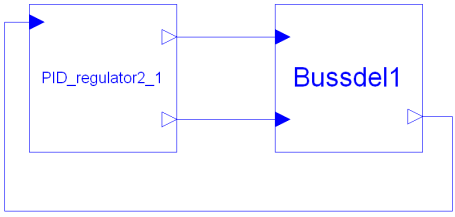
- Bussens massa (Body Mass) $m_1 = 2500kg$
- Bussens position x_1
- Däckens massa (Suspension Mass) $m_2 = 320kg$
- Däckens position x_2
- Fjäderkonstant $k_1 = 8000N/m$

- Fjäderkonstant $k_2 = 500000N/m$
- Dämpningskonstant $b_1 = 350Ns/m$
- Dämpningskonstant $b_2 = 150020Ns/m$
- Styrsignal u i form av kraft.
- Störningsignal w i form av position.
- En mätsignal, förändringen i fjäderns längd.

6.2 Design och implementering

Ur bekvämlighet- och stabilitetssynpunkt synpunkt ska en PID-regulator designas. Regulatorn ska eliminera svängningar som uppstår när bussen kör på ojämn väg. När bussen kör ner i en grop kommer bussen att oscillera och det måste dämpas ut.

Först delas modellen in i två delar, se figur 6.2, en regulator- och en bussdel. Regulatordelen består, som i DC-motor exemplet, av komponenterna *PID*, *återkoppling* och *steg*.

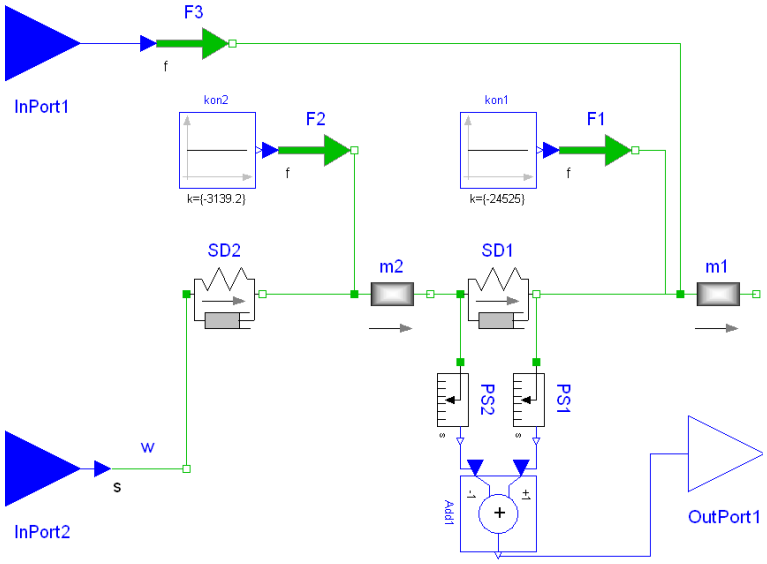


Figur 6.2. 1/4 buss modell skapad med Modelica

6.2.1 Linjär fjäder

Bussdelen, se figur 6.3, består av massorna m_1 och m_2, linjära fjäder/dämpare SD_1 och SD_2. Den innehåller också F_1 och F_2 som simulerar gravitationskraften som påverkar m_1 respektive m_2. En utsignal $OutPort1$ som ska återkopplas är differensen i positionen på SD_1: portar. Ingångarna $InPort1$ och $InPort2$ kommer från regulatordelen och simulerar styrsignalen u som simuleras av F_3 och en störningsignal w som är ett steg.

Det intressanta ligger i att återkoppla längdförändringen $SD1.s_{rel}$ för den linjära fjädern i SD_1. Man testar några värden på regulatorns parametrar och till slut fås ett bra resultat som syns i figur 6.4. Det syns hur störningen dämpas och försvinner inom en sekund.



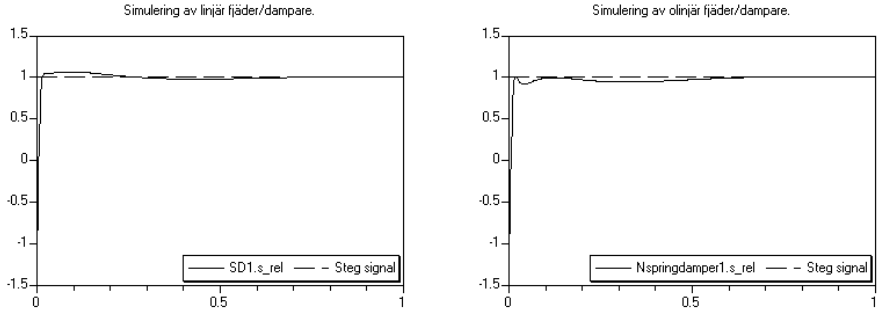
Figur 6.3. Bussdelen för modellen i figur 6.2

6.2.2 Olinjär fjäder

En olinjär fjäder kopplas till modellen genom att modifiera komponenten SD_1. Den nya definitionen av SD_1 ges av:

```
model NonLinearSpringDamper
  "Linear 1D translational spring and damper in parallel"
  extends Interfaces.Compliant;
  parameter SI.Position s_rel0=0 "unstretched spring length";
  parameter Real c(
    final unit="N/m",
    final min=0) = 1 "spring constant";
  parameter Real d(
    final unit="N/(m/s)",
    final min=0) = 1 "damping constant";
  SI.Velocity v_rel "relative velocity between flange_a and flange_b";
equation
  v_rel = der(s_rel);
  f = f=c1*(s_rel-s_rel0)+c2*(s_rel-s_rel0)^3+d*v_rel;
end NonLinearSpringDamper;
```

där c_1 och c_2 är fjäderkonstanter. Den nya olinjära komponenten kan ersätta den linjära SD_1 i figur 6.3 och simuleras på samma sätt som i det linjära fallet.



Figur 6.4. Linjär och olinjär fjäder i bussmodellen i figur 6.3

6.3 Inerter

Det går att dra analogier mellan den elektriska och den mekaniska domänen [15]. En massa, med lite restriktion, är analogt med en kapacitans, se tabell 6.1. Relationen mellan en massa m, kraft F och hastighet v i mekanisk domän framgår av Newtons andra lag:

$$m\dot{v} = F$$

Det medför att accelerationen $a = \dot{v}$ för massan m beräknas relativt ett fixt koordinatsystem, dvs $m(\dot{v} - 0) = F$. En massa är analogt med en *jordad* kapacitans. För att realisera en mekanisk komponent som är analog med en *ojordad* kapacitans, designas en ideal *inerter*. En inerter [14] är en mekanisk tvåport-komponent som har egenskapen att kraften F som verkar på portarna är proportionell mot den relativa accelerationen mellan portarna. En proportionell konstant b kallas *inertans* med enheten kilogram.

$$F = b\,(\dot{v}_2 - \dot{v}_1)$$

där v_1 och v_2 är hastigheten på portarna. Inerters potentiella energin är:

$$E_P = \frac{1}{2} b\,(v_2 - v_1)^2$$

Att designa en inerter i Modelica är enkelt. Det är bara att skapa en ny komponent enligt koden nedan:

```
model Inerter "Linear 1D translational inerter"
  extends Modelica.Mechanics.Translational.Interfaces.Compliant;
  Modelica.SIunits.Velocity v_rel "relative velocity";
  parameter Real b(final unit="kg", final min=0) = 0 "inertance";
equation
  v_rel = der(s_rel);
```

Mekanik	Ellära
Kraft	Ström
Hastighet	Spänning
Fjäder	Induktans
Dämpare	Resistans
Massa	Kapacitans

Tabell 6.1. Fysiska analogier mellan elektriska och mekaniska domäner

```
  f = b*der(v_rel);
end Inerter;
```

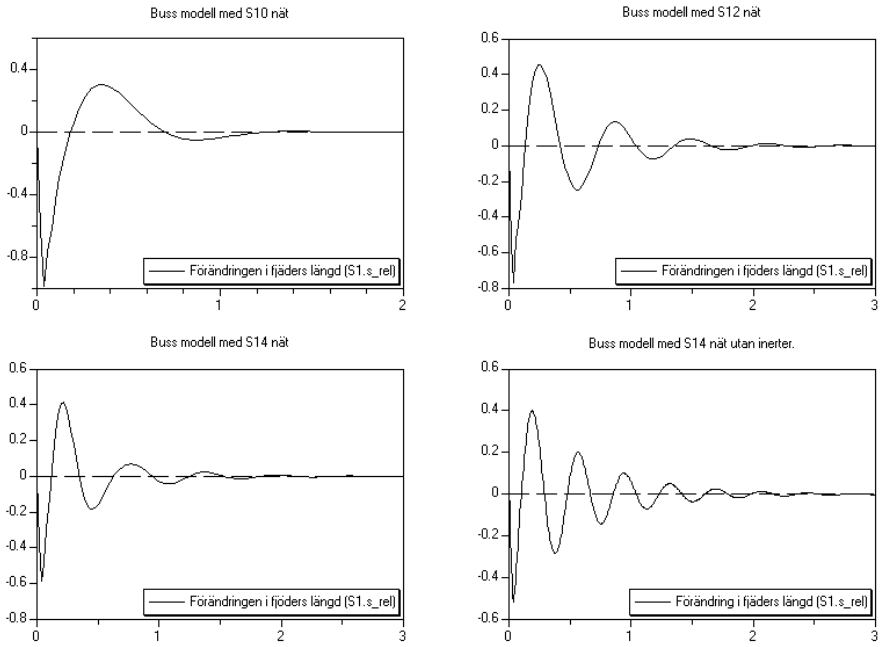
Som ett exempel kan 1/4-bussmodellen studeras. Tre olika konfigurationer, S_{10}, S_{12} och S_{14}, har utvärderats. De olika konfigurationer kan ses i figur 6.6. Konfigurationen, som består av ett antal komponenter som inerter, fjäder och dämpare, kopplas till modellen i figur 6.7. Hur en konfiguration ska designas beror på vilket värde fjäderkonstanten c har i S_1, för en djupare studie se [14] [15].

- $10\ kN/m < c < 18\ kN/m$.
  Designa ett nät som består av en inerter som är parallell med en dämpare och båda två är seriekopplade med en fjäder och en dämpare, se figur 6.6.
  För $c = 12\ kN/m$ bör man välja $d = 177.7\ kN/m$ för $Spring1$, $c = 250574\ Ns/m$ för $Damper2$, $c = 1900\ Ns/m$ för $Damper1$ och $b = 14.3\ kg$ för $Inerter1$.

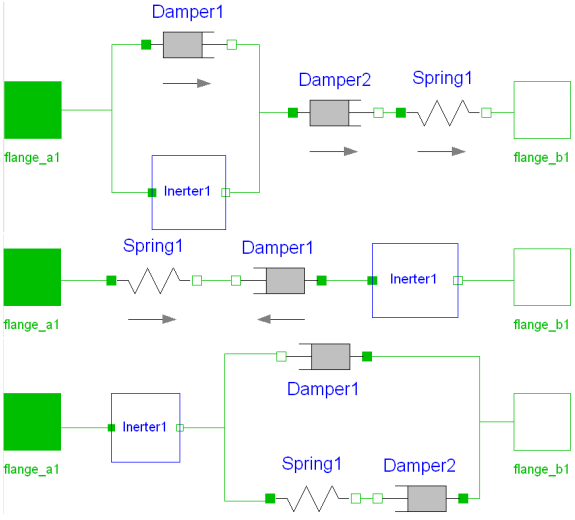
- $20\ kN/m < c < 65\ kN/m$.
  Designa ett nät som består av en inerter som är seriekopplad med en fjäder och en dämpare, se figur 6.6.
  För $c = 50\ kN/m$ bör man välja $d = 288.5\ kN/m$ för $Spring1$, $c = 2857\ Ns/m$ för $Damper2$ och $b = 297\ kg$ för $Inerter1$.

- $66\ kN/m < c < 120\ kN/m$.
  Designa ett nät som ser ut som i figur 6.6.
  För $c = 90\ kN/m$ bör man välja $d = 38.7\ kN/m$ för $Spring1$, $c = 29794\ Ns/m$ för $Damper2$, $c = 3306\ Ns/m$ för $Damper1$ och $b = 283\ kg$ för $Inerter1$.

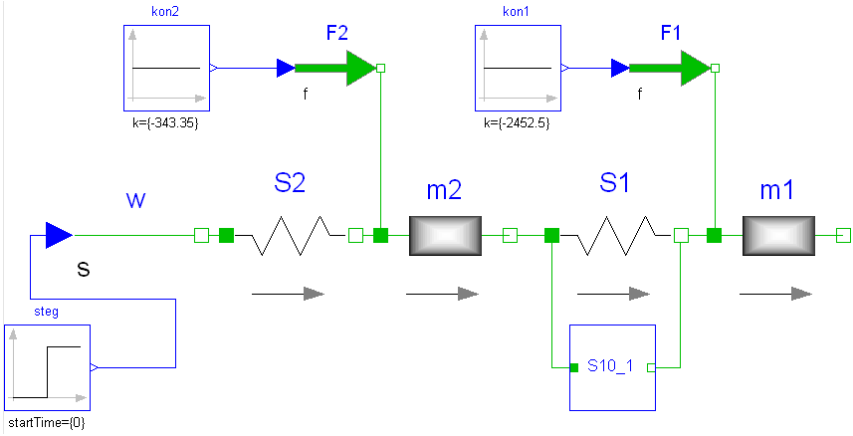
Resultatet för de tre fallen ges av figur 6.5, där bussmodellen störs med ett steg. Steget ska simulera en farthinder eller en grop som utsätter bussen för svängningar. De kopplade näten S_{10}, S_{12} och S_{14} kommer att arbeta för eliminera de störningar som uppstår så att bussen stabiliserar sig efter kort tid.



Figur 6.5. Svängningar i bussmodeller



Figur 6.6. S_{10}, S_{12} och S_{14} konfigurationer.



Figur 6.7. Bussmodellen med en S_{10} konfiguration.

Kapitel 7

Sammanfattning

- Kapitel 2: Matematisk modellering berörs samt ett antal simuleringsproblem har presenterats. Objektorienterad modellering som är ett nytt sätt att modellera fysikaliska modeller beskrivs. Principer för objektorienterad modellering såsom hierarki, ärvning och icke-kausalitet belystes. Dagens simulering och modellering språk/verktyg som domänspecifika simulerings- och modelleringsspråk och generella simuleringsverktyg har presenterats. Fördelar respektive nackdelar med några av dagens domänspecifika simulerings- och modelleringsspråk. Begreppet icke-kausal modellering av matematiska modeller har analyserats och fördelen med det jämfört med en kausal modellering.
- Kapitel 3: Detta kapitel handlar om Modelica som är ett objektorienterat och Ekvationbaserat modelleringsspråk, strukturen och historia för det. Kommersiella simuleringsverktyg som Dymola och MathModelica för att kompilera språket. Konnektor, algebraisk loop och index reducering beskrivits och de olika teknik som Modelica använder sig av för öka prestanda vid simulering.
- Kapitel 4: En elektrisk krets med antal grundläggande komponenter presenterats som ett exempel på en modell. Modellering av kretsen i Modelica och i Simulink har jämförts. Det syns hur Modelica design har fördelen gällande enkelhet, överskådlighet och naturlighet i det linjära fallet. Vad gäller olinjära fallet ser vi svårhet med realisering i Simulink modellen. Vissa olinjära system är svåra att modellera i Simulink, ty de kan inte skrivas om på tillståndsform. Några olinjära elektriska komponenter som resistans, kapacitans och induktans har designats mha Modelica dessutom presenteras några simulering resultat av olinjära kretsen.
- Kapitel 5: Analysring av en DC-motor som är ett typiskt exempel på styrning av en multidomän modell. Modellen har delats upp i tre delmodeller och simulerats i Modelica. Dessutom har en PID-regulator och en olinjär fjäder/dämpare komponent designats. Ett antal simulering resultat av tre fall, linjär fjäder/dämpare, olinjär fjäder/dämpare och fjäder/dämpare med glapp har presenterats.

- Kapitel 6: Hjulupphängning på en buss har studerats i det här kapitlet. Modellen som simulerar av ett 1/4 buss system har designats mha Modelica språket dessutom en PID-regulator för kunna stabilisera modellen. Olinjäritet har undersökt i modellen också genom att realiseringen av en olinjär fjäder. Modelicas styrka har återigen bevisats vid design av en ny mekanisk komponent *inerter* och hur den kan vara en del av en krets för dämpa oscillationer i modellen. Ett antal konfigurationer har designats, S_{10}, S_{12} och S_{14}. Simuleringsresultat av de olika fall har presenterats.

7.1 Modelica

Objektorienterad modellering är ett utmärkt sätt att analysera fysikaliska problem. Den kan hantera stora och komplexa fysikaliska system. Modelica är ett multidomän programmeringsspråk och ekvationbaserad modelleringsspråk. Den är baserad på objektorientering principer och hanterar icke-kausala problem. Fördelen med design i Modelica är:

- Modelica är ett språk som stödjer multidomän och ekvation-baserad modellering.
- Dataflödet sker i två riktningar *icke-kausalitet*. In- och utsignal behöver inte specificeras.
- Man kan dela modellen i ett antal delmodeller.
- Design speglar det verkliga fysikaliska system.
- Matematiska ekvationer kan skrivas i sin naturliga form.
- Designen av delmodeller sker oberoende av varandra.
- Delmodeller kan modifieras eller bytas ut samt själva befintliga modellen kan ingå som delmodell i en ny supermodell.
- Man kan modeifera en komponent från Modelica standard bibliotek och döpa den eller skapa ett helt nytt komponent som `Nspring1`.

7.2 Dymola

Dymola är ett användarvänligt simuleringsverktyg för Modelica. Man har tillgång till Modelicas standardbibliotek via ett grafiskt gränssnitt. Vid simulering kontrolleras modellen så att Modelica syntax är godkänd sedan kompileras modellen för att få modellen på sk *Flattened form*. Efter kontrollen och kompilering simuleras modellen.

En nackdel med Dymola verktyg är den tillämpar den grafteoretiska algoritmen *Pantelides algoritm*, för att bestämma vilken tillståndsvariabel som ska lösas först. Pantelides algoritm fungerar men den är inte exakt.

En mer exakt metod för lösning av DAE system är *Kunkel-Mehrmann algoritm* [10]. Nackdelen med metoden är den komplicerad och kräver en avancerad programvara.

Litteraturförteckning

[1] http://www.cds.caltech.edu/oldweb/courses/1999-2000/cds221/tummescheit-24mar00.pdf.

[2] http://www.control.lth.se/ fumodell/lectures00/lecture7.pdf.

[3] http://www.ida.liu.se/labs/pelab/modelica/openmodelica.html.

[4] http://www.library.cmu.edu/ctms/.

[5] P. Aronsson. *Automatic parallelization of equation-based simulation programs.* Department of Computer and Information Science, Linköping University, 2006. ISBN 9185523682.

[6] W. Borutzky. *Relation between bond graph based and object-oriented physical systems modeling.* SCS Publishing, 1999.

[7] H. Elmqvist. *A Structured Model Language for Large Continuous Systems.* Department of Automatic Control, Lund University, Sweden, 1978.

[8] H. Elmqvist and M. Otter. *Methods for Tearing Systems of Equations in Object-Oriented Modelling.* Proceedings of the European Simulation Multiconference, pp. 326-332, Barcelona, Spain, 1994.

[9] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* 2003. ISBN 0-471-471631.

[10] P. Kunkel and V. Mehrmann. *Differential-Algebraic Equations : Analysis and Numerical Solution.* Zürich-European Mathematical Society, 2006. ISBN 3-03719-017-5.

[11] L. Ljung and T. Glad. *Modellbygge och simulering.* 2 edition, 2004.

[12] S.E. Mattsson and H. Elmqvist. *Modelica-An International Effort to Design the Next Generation Modelling Language.* 7th IFAC Symp. on Computer Aided Control Systems Design, CACSD'97, Gent, Belgium, 28-30 April, 1997.

[13] Modelica and the Modelica Association. http://www.modelica.org.

[14] C. Papageorgiou and M.C. Smith. *Positive Real Synthesis Using Matrix Inequalities for Mechanical Networks: Application to Vehicle Suspension*. IEEE Transactions on Control Systems Technology, Vol. 14, No. 3, 2006.

[15] M.C. Smith. *Synthesis of Mechanical Networks: The Inerter*. IEEE Transactions on Automatic Control, Vol. 47, No. 10, 2002.

Bilaga A

Modelica kod

```
Model citcuit
  Modelica.Electrical.Analog.Basic.Resistor R1(R=10);
  Modelica.Electrical.Analog.Basic.Resistor R2(R=100);
  Modelica.Electrical.Analog.Basic.Inductor L(L=0.1);
  Modelica.Electrical.Analog.Basic.Capacitor C(R=0.01);
  Modelica.Electrical.Analog.Basic.Ground G;
  Modelica.Electrical.Analog.Sources.SignalVoltage AC;
equation
  connect(AC.n, C.n);
  connect(G.p, AC.n);
  connect(R1.n, C.p);
  connect(AC.p, R1.p);
  connect(L.p, R2.n);
  connect(R1.p, R2.p);
  connect(L.n, C.n);
end circuit;
```

Tabell A.1. Modelicakoden för kretsmodellen i figur 4.1

```
model Nonlinear71
  Modelica.SIunits.Voltage VC1;
  Modelica.SIunits.Voltage VC2;
  Modelica.SIunits.Voltage VL1;
  Modelica.SIunits.Voltage VL2;
  Modelica.SIunits.Voltage VR;
  Modelica.SIunits.Current IC1;
  Modelica.SIunits.Current IC2;
  Modelica.SIunits.Current IL1;
  Modelica.SIunits.Current IL2;
  Modelica.SIunits.Current IR;
  parameter Real R=1;
equation
  0 = VC1 - VC2;
  0 = IL1 - IL2;
  0 = IL1 - IR;
  0 = IL1 - IC1 - IC2;
  0 = sin(time) - VL1 - VL2 - VC1 - VR;
  0 = VR - R*(IR + IR^3);
  der(IL1) = VL1^3 + VL1;
  der(IL2) = arctan(VL2) + VL2;
  der(VC1) = IC1/(1 + VC1);
  der(VC2) = IC2/(1 + VC2);
end Nonlinear71;
```

Tabell A.2. Modelicakoden för kretsmodellen i ekvation 4.6

```
model Nonlinear
  NonLinearInductor1 L1;
  NonLinearInductor2 L2;
  NonLinearCapacitor1 C1;
  NonLinearCapacitor2 C2;
  NonLinearResistor R;
  Modelica.Electrical.Analog.Basic.Ground G;
  Modelica.Electrical.Analog.Sources.SineVoltage V;
equation
  connect(L1.n, L2.p);
  connect(V.p, L1.p);
  connect(V.n, G.p);
  connect(R.p, L2.n);
  connect(R.n, C1.p);
  connect(C2.n, C1.n);
  connect(G.p, C2.n);
  connect(C2.p, R.n);
end Nonlinear;
```

Tabell A.3. Modelicakoden för kretsmodellen i figur 4.7.

```
model NonLinearCapacitor1 "non linear electrical capacitor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
equation
  i/(1 + v) = der(v);
end NonLinearCapacitor1;

model NonLinearCapacitor2 "non linear electrical capacitor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
equation
  i/(2 + v) = der(v);
end NonLinearCapacitor2;

model NonLinearInductor1 "non linear electrical iductor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
equation
  der(i) = v^3 + v;
end NonLinearInductor1;

model NonLinearInductor2 "non linear electrical iductor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
equation
  der(i) = arctan(v) + v;
end NonLinearInductor2;

model NonLinearResistor "non linear electrical resistor"
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
equation
  (i + i^3) = v;
end NonLinearResistor;
```

Tabell A.4. Modelicakoden för olinjära komponenter *C1, C2, L1, L2* och R i figur 4.7.

```
model DC-motor
  Mekanisk Mekanisk1;
  PID_regulator PID_regulator1;
  Elektrisk Elektrisk1;
equation
  connect(PID_regulator1.OutPort1, Elektrisk1.InPort1);
  connect(Elektrisk1.Flange_b1, Mekanisk1.Flange_a1);
  connect(Mekanisk1.OutPort1, PID_regulator1.InPort1);
end DC-motor;
```

Tabell A.5. Modelicakoden för DC-motorn i figur 5.2

```
model PID_regulator
  Modelica.Blocks.Continuous.PID PID1(
    k=750,
    Ti=2000,
    Td=.2,
    Nd=100);
  Modelica.Blocks.Math.Feedback FD;
  Modelica.Blocks.Sources.Step steg;
  Modelica.Blocks.Interfaces.InPort InPort1;
  Modelica.Blocks.Interfaces.OutPort OutPort1;
equation
  connect(FD.outPort, PID1.inPort);
  connect(steg.outPort, FD.inPort1);
  connect(FD.inPort2, InPort1);
  connect(PID1.outPort, OutPort1);
end PID_regulator;
```

Tabell A.6. Modelicakoden för PID-regulator i figur 5.3

```
model Elektrisk
  Modelica.Electrical.Analog.Basic.Resistor R(R=1);
  Modelica.Electrical.Analog.Basic.Inductor L(L=0.5);
  Modelica.Electrical.Analog.Basic.EMF E(k=0.01);
  Modelica.Electrical.Analog.Basic.Ground G;
  Modelica.Electrical.Analog.Sources.SignalVoltage V;
  Modelica.Blocks.Interfaces.InPort InPort1;
  Modelica.Mechanics.Rotational.Interfaces.Flange_b Flange_b1;
equation
  connect(R.n, L.p);
  connect(G.p, E.n);
  connect(V.n, G.p);
  connect(V.p, R.p);
  connect(L.n, E.p);
  connect(V.inPort, InPort1);
  connect(E.flange_b, Flange_b1);
end Elektrisk;
```

Tabell A.7. Modelicakoden för elektrisk delmodell i figur 5.4

```
model Mekanisk
  Modelica.Mechanics.Rotational.Inertia J1(J=.1);
  Modelica.Mechanics.Rotational.Inertia J2(J=.01);
  Modelica.Blocks.Sources.Step steg1;
  Modelica.Mechanics.Rotational.Torque W;
  Modelica.Mechanics.Rotational.Fixed F;
  Modelica.Mechanics.Rotational.Damper D(d=1);
  Modelica.Mechanics.Rotational.Sensors.TorqueSensor TS;
  Modelica.Mechanics.Rotational.Interfaces.Flange_a Flange_a1;
  Modelica.Blocks.Interfaces.OutPort OutPort1;
  Nspring Nspring1;
equation
  connect(J1.flange_b, W.flange_b);
  connect(F.flange_b, D.flange_a);
  connect(D.flange_b, J2.flange_a);
  connect(steg1.outPort, W.inPort);
  connect(TS.flange_b, J1.flange_a);
  connect(TS.outPort, OutPort1);
  connect(Flange_a1, J2.flange_a);
  connect(Nspring1.flange_b, TS.flange_a);
  connect(Nspring1.flange_a, J2.flange_b);
end Mekanisk;
```

Tabell A.8. Modelicakoden för mekanisk delmodell i figur 5.5

```
model Bussmodell
  PID_regulator2 PID_regulator2_1;
  Bussdel Bussdel1;
equation
  connect(PID_regulator2_1.OutPort1, Bussdel1.InPort1);
  connect(PID_regulator2_1.OutPort2, Bussdel1.InPort2);
  connect(Bussdel1.OutPort1, PID_regulator2_1.InPort1);
end Bussmodell;
```

Tabell A.9. Modelicakoden för figur 6.2

```
model PID_regulator2
  Modelica.Blocks.Continuous.PID PID1(
    Nd=100,
    k=832100,
    Ti=1.33,
    Td=.25);
  Modelica.Blocks.Math.Feedback FD;
  Modelica.Blocks.Interfaces.InPort InPort1;
  Modelica.Blocks.Interfaces.OutPort OutPort1;
  Modelica.Blocks.Interfaces.OutPort OutPort2;
  Modelica.Blocks.Sources.Step Step1;
equation
  connect(FD.outPort, PID1.inPort);
  connect(PID1.outPort, OutPort1);
  connect(Step1.outPort, OutPort2);
  connect(Step1.outPort, FD.inPort1);
  connect(FD.inPort2, InPort1);
end PID_regulator2;
```

Tabell A.10. Modelicakoden för PID-regulator i figur 6.2

```
model Bussdel
  Modelica.Mechanics.Translational.SpringDamper SD1(
    c=80000,
    d=350,
    s_rel0=9.81*m1.m/SD1.c);
  Modelica.Mechanics.Translational.SlidingMass m1(m=2500);
  Modelica.Mechanics.Translational.Force F1;
  Modelica.Blocks.Sources.Constant kon1(k={-24525});
  Modelica.Mechanics.Translational.Force F3;
  Modelica.Mechanics.Translational.Sensors.PositionSensor PS1;
  Modelica.Blocks.Interfaces.OutPort OutPort1;
  Modelica.Blocks.Interfaces.InPort InPort1;
  Modelica.Mechanics.Translational.SpringDamper SD2(
    c=500000,
    d=15020,
    s_rel0=9.81*(m2.m + m1.m)/SD2.c);
  Modelica.Mechanics.Translational.Position p;
  Modelica.Mechanics.Translational.SlidingMass m2(m=320);
  Modelica.Mechanics.Translational.Force F2;
  Modelica.Blocks.Sources.Constant kon2(k={-3139.2});
  Modelica.Mechanics.Translational.Sensors.PositionSensor PS2;
  Modelica.Blocks.Interfaces.InPort InPort2;
  Modelica.Blocks.Math.Add Add1(k1=-1);
equation
  connect(SD1.flange_b, m1.flange_a);
  connect(kon1.outPort, F1.inPort);
  connect(F1.flange_b, m1.flange_a);
  connect(F3.flange_b, m1.flange_a);
  connect(m1.flange_b, PS1.flange_a);
  connect(F3.inPort, InPort1);
  connect(p.flange_b, SD2.flange_a);
  connect(SD2.flange_b, m2.flange_a);
  connect(F2.flange_b, m2.flange_a);
  connect(m2.flange_b, PS2.flange_a);
  connect(kon2.outPort, F2.inPort);
  connect(p.inPort, InPort2);
  connect(m2.flange_b, SD1.flange_a);
  connect(PS2.outPort, Add1.inPort1);
  connect(PS1.outPort, Add1.inPort2);
  connect(Add1.outPort, OutPort1);
end Bussdel;
```

Tabell A.11. Modelicakoden för figur 6.3

```
model Nspringdamper
  "Non-linear 1D translational spring and damper in parallel"
  Modelica.Mechanics.Translational.Interfaces.Compliant;
  parameter Modelica.SIunits.Position s_rel0=0 "unstretched spring length";
  parameter Real c1(
    final unit="N/m",
    final min=0) = 1 "1st spring constant";
  parameter Real c2(
    final unit="N/m",
    final min=0) = 1 "2nd spring constant";
  parameter Real d(
    final unit="N/(m/s)",
    final min=0) = 1 "damping constant";
  Modelica.SIunits.Velocity v_rel "relative velocity between flange_a and flan
equation
  v_rel = der(s_rel);
  f = c1*(s_rel-s_rel0)+c2*(s_rel - s_rel0)^3+d*v_rel;
end Nspringdamper;
```

Tabell A.12. Modelicakoden för olinjär fjäder/dampare

```
model Nspring "Non-Linear 1D rotational spring"
  extends Modelica.Mechanics.Rotational.Interfaces.Compliant;
  parameter Real c1;
  parameter Real c2(final unit="N.m/rad", final min=0) "c1*x^3+c2*x";
  parameter Modelica.SIunits.Angle phi_rel0=0 "Unstretched spring angle";
equation
    tau = c1*(phi_rel - phi_rel0)^3 + c2*(phi_rel - phi_rel0);
end Nspring;
```

Tabell A.13. Modelicakoden för olinjär fjäder i figur 5.5

```
model S10
  Inerter Inerter1(b=14.3);
  Modelica.Mechanics.Translational.Damper Damper1(d=1900);
  Modelica.Mechanics.Translational.Damper Damper2(d=250574);
  Modelica.Mechanics.Translational.Spring Spring1(c=177700);
  Modelica.Mechanics.Translational.Interfaces.Flange_a flange_a1;
  Modelica.Mechanics.Translational.Interfaces.Flange_b flange_b1;
equation
  connect(Damper1.flange_a, Inerter1.flange_a);
  connect(Inerter1.flange_a, flange_a1);
  connect(Spring1.flange_b, flange_b1);
  connect(Inerter1.flange_b, Damper1.flange_b);
  connect(Damper2.flange_a, Inerter1.flange_b);
  connect(Damper2.flange_b, Spring1.flange_a);
end S10;
```

Tabell A.14. Modelicakoden för S10 i figur 6.6

```
model S12
  Modelica.Mechanics.Translational.Spring Spring1(c=288500);
  Modelica.Mechanics.Translational.Damper Damper1(d=2857);
  Inerter Inerter1(b=297);
  Modelica.Mechanics.Translational.Interfaces.Flange_b flange_b1;
  Modelica.Mechanics.Translational.Interfaces.Flange_a flange_a1;
equation
  connect(Spring1.flange_b, Damper1.flange_b);
  connect(Inerter1.flange_b, flange_b1);
  connect(Spring1.flange_a, flange_a1);
  connect(Damper1.flange_a, Inerter1.flange_a);
end S12;
```

Tabell A.15. Modelicakoden för S12 i figur 6.6

```
model S14
  Inerter Inerter1(b=283);
  Modelica.Mechanics.Translational.Damper Damper1(d=3306);
  Modelica.Mechanics.Translational.Damper Damper2(d=29794);
  Modelica.Mechanics.Translational.Spring Spring1(c=38700);
  Modelica.Mechanics.Translational.Interfaces.Flange_b flange_b1;
  Modelica.Mechanics.Translational.Interfaces.Flange_a flange_a1;
equation
  connect(Spring1.flange_a, Inerter1.flange_b);
  connect(Inerter1.flange_a, flange_a1);
  connect(Damper1.flange_b, Spring1.flange_a);
  connect(Damper2.flange_b, Spring1.flange_b);
  connect(Damper1.flange_a, Damper2.flange_a);
  connect(Damper2.flange_a, flange_b1);
end S14;
```

Tabell A.16. Modelicakoden för S14 i figur 6.6

```
model BUSS_10
  Modelica.Mechanics.Translational.SlidingMass m1(m=250);
  Modelica.Mechanics.Translational.Force F2;
  Modelica.Mechanics.Translational.Force F1;
  Modelica.Blocks.Sources.Step steg(height={1});
  Modelica.Blocks.Sources.Constant kon2(k={-343.35});
  Modelica.Blocks.Sources.Constant kon1(k={-2452.5});
  Modelica.Mechanics.Translational.Spring S2(c=150000,
   s_rel0=9.81*(m2.m + m1.m)/S2.c);
  Modelica.Mechanics.Translational.Spring S1(s_rel0=9.81*m1.m/S1.c, c=12000);
  Modelica.Mechanics.Translational.SlidingMass m2(m=35);
  Modelica.Mechanics.Translational.Position w;
  S10 S10_1;
equation
  connect(kon1.outPort, F1.inPort);
  connect(kon2.outPort, F2.inPort);
  connect(F1.flange_b, m1.flange_a);
  connect(S1.flange_b, m1.flange_a);
  connect(m2.flange_b, S1.flange_a);
  connect(S2.flange_b, m2.flange_a);
  connect(F2.flange_b, m2.flange_a);
  connect(steg.outPort, w.inPort);
  connect(S10_1.flange_b1, S1.flange_b);
  connect(S10_1.flange_a1, S1.flange_a);
  connect(w.flange_b, S2.flange_a);
end BUSS_10;
```

Tabell A.17. Modelicakoden för bussmodellen i figur 6.7