Feldetektering för diagnos med differentialgeometriska metoder

-en implementation i Mathematica

Examensarbete
utfört vid Fordonssystem

av
Anna Önnegren

Reg nr: LiTH-ISY-EX-3551-2004

18 september 2004

Feldetektering för diagnos med differentialgeometriska metoder

-en implementation i Mathematica

Examensarbete
utfört vid Fordonssystem,
Institution för systemteknik
på Linköpings universitet
av Anna Önnegren

Reg nr: LiTH-ISY-EX-3551-2004

Handledare: Erik Frisk
Linköpings universitet

Examinator:Erik Frisk
Linköpings universitet

Linköping, 18 september 2004

Avdelning, Institution
Division, Department

Vehicular Systems,
Dept. of Electrical Engineering
581 83 Linköping

Datum
Date

18 september 2004

Språk
Language
☒ Svenska/Swedish
☐ Engelska/English
☐ _____

Rapporttyp
Report category
☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
—

ISRN
LITH-ISY-EX-3551-2004

Serietitel och serienummer
Title of series, numbering

ISSN
—

URL för elektronisk version
http://www.vehicular.isy.liu.se
http://www.ep.liu.se/exjobb/isy/2004/3551/

Titel Feldetektering för diagnos med differentialgeometriska metoder
-en implementation i Mathematica

Title Fault detection for diagnosis with differential geometric methods
-an implementation in Mathematica

Författare
Author
Anna Önnegren

Sammanfattning
Abstract

Diagnosis means detection and isolation of faults. A model based diagnosis system is built on a mathematical model of the system. The difficulty when constructing the diagnosis system depends om how the model is formulated. In this report, a method is described that rewrites the model on such a form that the construction of the diagnosis algoritm is easy. The model is transformed by two state space transformations and the result will be a system on state space form where one part of the system becomes easy to supervise.

The main part of the report describes the procedure to create these transformations, which can be done in seven steps, based on differential geometric methods.

The aim of this masters thesis was to create an implementation in Mathematica (a computer tool for symbolic formula manipulation) of the creation of the two transformations and the system transformation. The created functions are described and examples of these are given.

A further aim was to evaluate if Mathematica could be a good support to rewrit a model. This was done by studying examples, and on the basis of the examples, identify difficult and easy steps.

The program has shown to be a good aid. Two of the seven steps have been identified as difficult and proposals for improvements have been given.

Nyckelord
Keywords
Diagnosis, Fault Detection, System Transformation, Differential Geometry

Sammanfattning

Diagnos innebär detektering och isolering av fel. Modellbaserad diagnos bygger på en matematisk modell av systemet. Att konstruera diagnossystemet är olika svårt beroende på hur modellen är formulerad. I den här rapporten beskrivs en metod som skriver om modellen på en form så att konstruktionen av diagnosalgoritmen blir enkel. Modellen transformeras med hjälp av två tillståndstransformationer och som resultat fås ett system på tillståndsform där en del av systemet blivit enkelt att övervaka.

Huvuddelen av rapporten beskriver tillvägagångssättet att skapa dessa transformationer, vilket kan göras i sju delsteg, baserat på differentialgeometriska metoder.

Examensarbetet gick ut på att skapa en implementation i Mathematica (ett datorverktyg för symbolisk formelmanipulation) av transformationsframtagningen samt systemtransformeringen. De skapade funktionerna beskrivs och exempel till dessa ges.

Ytterligare ett mål var att utvärdera om Mathematica kan vara ett bra stöd vid omskrivning av modellen. Detta gjordes geonom att studera exempel och utifrån exemplen identifiera svåra och enkla delsteg.

Programmet har visat sig fungera bra som ett hjälpmedel. Två av de sju delstegen har identifierats som svåra och förslag till förbättringar har där getts.

Nyckelord: Diagnos, Feldetektering, Systemtransformation, Differentialgeometri

Abstract

Diagnosis means detection and isolation of faults. A model based diagnosis system is built on a mathematical model of the system. The difficulty when constructing the diagnosis system depends om how the model is formulated. In this report, a method is described that rewrites the model on such a form that the construction of the diagnosis algoritm is easy. The model is transformed by two state space transformations and the result will be a system on state space form where one part of the system becomes easy to supervise.

The main part of the report describes the procedure to create these transformations, which can be done in seven steps, based on differential geometric methods.

The aim of this masters thesis was to create an implementation in Mathematica (a computer tool for symbolic formula manipulation) of the creation of the two transformations and the system transformation. The created functions are described and examples of these are given.

A further aim was to evaluate if Mathematica could be a good support to rewrit a model. This was done by studying examples, and on the basis of the examples, identify difficult and easy steps.

The program has shown to be a good aid. Two of the seven steps have been identified as difficult and proposals for improvements have been given.

Keywords: Diagnosis, Fault Detection, System Transformation, Differential Geometry

Innehåll

Kapitel 1

Inledning

Inom industrin är det viktigt att upptäcka när något har gått fel, vid vilken tidpunkt och i vilken komponent felet uppstod i, för att så snabbt som möjligt kunna åtgärda felet. En snabb åtgärd kan vara viktig ur flera aspekter, både säkerhetsmässiga och ekonomiska.

För att kunna avgöra i vilken komponent ett fel har uppstått i krävs att man utifrån känd data kan utläsa denna information. Den här möjligheten är inget man får automatiskt, utan systemet måste konstrueras så att dessa önskade egenskaper skapas. Kapitel 2 beskriver teorin bakom detta medan kapitel 3 översiktligt beskriver tillvägagångsättet, vilket innefattar 2 variabeltransformationer.

Kapitel 4 kommer sedan i mer detalj beskriva de matematiska stegen som måste utföras för att transformera systemet. Utgående från ett system på tillståndsform får man efter transformationen också ett system på tillståndsform. Teorin bakom hur man skapar denna nya tillståndsform är invecklad och kommer inte gås igenom i någon större utsträckning. Den behandlas vanligtvis inte på civilingenjörsutbildningar, utan dyker oftast upp på doktorandnivå. För den som vill fördjupa sig i varför dessa steg måste utföras hänvisas till artikeln som ligger till grund för detta examensarbete [1].

Syftet med examensarbetet är att skapa funktioner i Mathematica som skapar transformationerna samt transformerar systemet. Kapitel 5 illustrerar funktionsflödet medan kapitel 6 går igenom funktionerna med hjälp av ett exempel.

Därefter diskuteras de problem som uppkommit under arbetets gång samt möjlig utveckling av programmet. I Bilaga A finns de variabler och funktioner som används i programmet medan Bilaga B innehåller de definitioner och räkneoperationer som tas upp i arbetet. Sist i dokumentet, bilaga C, finns den kod som skapats i Mathematica. Förutom Mathematicas egna funktioner använder sig programmet av funktioner skapade på Fordonssystem vid Linköpings universitet. Vid intresse av dessa funktioner kontakta avdelningen.

Kapitel 2

Diagnos

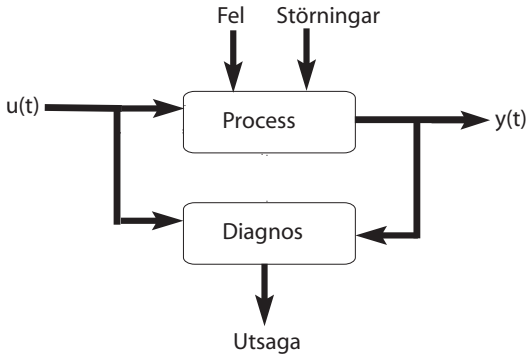
Om ett fel har uppstått i ett system vill man så snabbt som möjligt kunna avgöra i vilken komponent felet har uppstått i för att så fort som möjligt kunna rätta till felet och få systemet rullande igen. Att avgöra ifall det har uppstått något fel i systemet samt vanligtvis också vid vilken tidpunkt felet uppstod kallas för detektering av fel. Att lokalisera var felet har uppstått, exempelvis i vilken komponent, kallas för isolering. Diagnos innebär detektering och isolering av fel.

Området diagnos har idag fått en stor utbredning och används bland annat inom övervakningssystem för bilar, flygplan och industrirobotar samt övervakningssystem för gasturbiner och kemiska processer. Manuell diagnos har funnits så länge det har funnits tekniska system medan automatisk diagnos började uppkomma när datorn blev ihopkopplad med systemen. I början av 1970-talet kom de första forskningsresultaten på modellbaserad diagnos. Ett av de första forskningsområdena var flygrelaterade system sponsrade av NASA.

När diagnossystemet är baserat på en explicit matematisk modell över processen som ska övervakas brukar man säga modellbaserad diagnos, vilket är den typ av diagnos som framöver kommer att behandlas. Modellbaserad diagnos illustreras i figur 2.1. Där kan man se att fel och störningar kommer in och påverkar processen. Om exempelvis både utsignalen $y(t)$ och insignalen $u(t)$ är kända variabler kan man ta reda på felens och störningarnas påverkan genom bland annat skapandet av så kallade residualer enligt (2.1), där $\hat{y}(t)$ är den skattade (beräknade) signalen. Ett system med kända in- och utsignaler har man bland annat om insignalen skickas ifrån en dator och utsignalen tas emot av en dator.

$$r(t) = y(t) - \hat{y}(t) \tag{2.1}$$

Om ett fel uppstår kommer residualen att bli nollskild, se figur 2.2. Residualerna kan användas för att detektera fel.



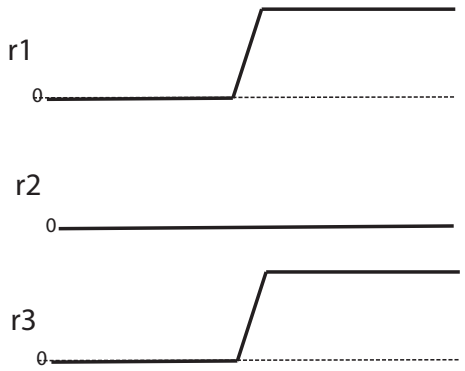
Figur 2.1: *Modellbaserad diagnos. Fel och störningar kommer in och påverkar processen. Med hjälp av kända variabler kan man avgöra ifall ett fel har uppstått i systemet.*

Figur 2.2: *En schematisk bild av residualsignalen. När ett fel uppstår kommer signalen att bli nollskild.*

På ovan nämnda sätt kan man skapa en uppsättning av residualer och sammanställa dem enligt tabell 2.1. Här står o för att residualen inte är känslig för felet medan x står för att den är känslig för felet. Tittar man på r_2 (residual 2) så påverkas den av f_1 och f_3 (fel 1 och fel 3), medan den inte påverkas av f_2 (fel 2). Denna egenskap att vara okänslig för f_2 är inget som uppkommer automatiskt, utan något man försöker åstadkomma. Genom att se f_2 som en störning och därefter göra systemet okänsligt för störningar är det sätt som framöver kommer att presenteras. Residualerna för f_2 ser ut enligt figur 2.3.

Tabell 2.1: *Med hjälp av residualkombinationen kan man utläsa vilket fel som uppstått*

	f1	f2	f3
r1	o	x	x
r2	x	o	x
r3	x	x	o



Figur 2.3: *En schematisk bild av signalerna för residualkombinationen för fel 2. Residual 1 och 3 blir nollskilda, medan residual 2 inte påverkas.*

Detta sätt att göra residualerna okänsliga för vissa fel möjliggör isolering, genom att jämföra residualens signatur med motsvarande kolumn i tabell 2.1. Fördelen med isolering av fel är att man med hjälp av residualkombinationen kan utläsa vilket fel som påverkar processen. Om man utifrån figur 2.3 ska utläsa vilket fel som har uppstått så får man gå in i tabell 2.1 och se vilka möjliga fel det skulle kunna vara om residualkombinationen visar att r_1 och r_3 är nollskilda medan r_2 är konstant noll. Enda möjligheten i detta faller är att f_2 kan ha uppstått.

Anledningen till att man vill göra systemet okänsligt för störningar är att störningarna påverkar residualerna vilka då kan bli svårare att tolka.

Kapitel 3

Principer för metoden

Som nämndes i föregående kapitel vill vi kunna generera residualer som är okänsliga för störningar. I den metod jag har studerat görs detta genom att gå från en modellbeskrivning till en annan modellbeskrivning där dessa önskvärda egenskaper finns. Det finns flera olika metoder att utföra detta på. En av dessa metoder kommer att presenteras, varav resten av dokumentet kommer att behandla. Metoden beskrivs i artikeln [1] medan den bakomliggande teorin och matematiken beskrivs i boken [2]. Den modellstruktur som den här metoden betraktar är följande.

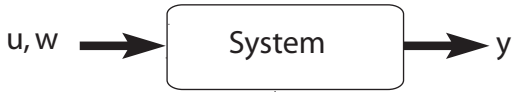
$$\begin{aligned} \dot{x} &= f(x) + g(x)u + l(x)m + p(x)w \\ y &= h(x) \end{aligned} \tag{3.1}$$

Tillstånden x definieras i ett område X runt origo i rummet $\mathbb{R}^n$, insignalen $u \in \mathbb{R}^a$, felet $m \in \mathbb{R}^b$, störningen $w \in \mathbb{R}^d$ och utsignalen $y \in \mathbb{R}^q$. De a kolumnerna av $g(x)$ (dvs $g_1(x)$, ..., $g_a(x)$), $f(x)$, de b kolumnerna av $l(x)$ och de d kolumnerna av $p(x)$ samt $h(x)$ är mjuka vektorfält, dvs. deriverbara tillräckligt många gånger. För övrigt är $f(0) = 0$ och $h(0) = 0$.

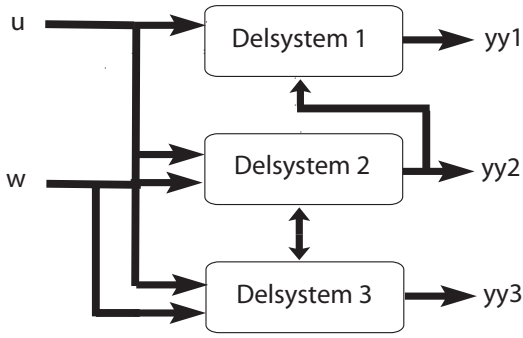
Skapandet av den störningsavkopplade modellbeskrivningen kan göras genom två variabeltransformationer, en som transformerar tillstånden ($\Phi(x)$) och en som transformerar utsignalen ($\Psi(y)$). Man går då ifrån ett system enligt figur 3.1 till ett system enligt figur 3.2. Därefter kan man plocka ut den delen av systemet som är avkopplat ifrån störningarna, framöver kallat kvotsystemet, enligt figur 3.3. I figurerna står u för insignalen, w för störningen, y för den ursprungliga utsignalen och yy står för den transformerade utsignalen. Enligt tabell 2.1 ville man även få systemet okänsligt för vissa fel. Dessa fel innesluts i störningen w.

Kvotsystemt kan vara av olika storlek, exempelvis skulle en variant kunna vara att kvotsystemet är tomt, medan systemet delades upp i delsystem 2 och

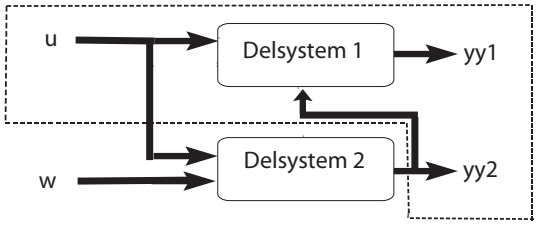
3. Då kvotsystemet är den delen av systemet som är separerad från störningar så skulle inte denna variant vara av något intresse. Det man istället vill skapa är det största möjliga kvotsystemet, dvs. ett så stort system som möjligt som inte påverkas av störningar.



Figur 3.1: *Det ursprungliga systemet. Fel och störningar påverkar hela systemet.*



Figur 3.2: *Det transformerade systemet. Fel och störningar påverkar bara en del av systemet.*



Figur 3.3: *Kvotsystemet. Den delen av systemet som inte påverkas av någon störning.*

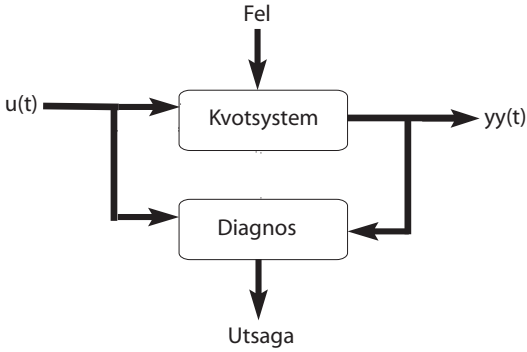
Efter transformeringen har systemet fått en struktur enligt

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, z_2) + g_1(z_1, z_2)u \\ \dot{z}_2 &= f_2(z_1, z_2, z_3) + g_2(z_1, z_2, z_3)u \\ \dot{z}_3 &= f_3(z_1, z_2, z_3) + g_3(z_1, z_2, z_3)u \\ yy_1 &= h_1(z_1) \\ yy_2 &= z_2 \end{aligned} \tag{3.2}$$

där z är de transformerade tillstånden. Om man utifrån detta tar ut $\dot{z}_1$ och yy_1 samt byter ut z_2 mot yy_2 så får man ut kvotsystemet

$$\begin{aligned} \dot{z}_1 &= f_1(z_1, yy_2) + g_1(z_1, yy_2)u \\ yy_1 &= h_1(z_1). \end{aligned} \tag{3.3}$$

Som tidigare nämnts vill man skapa residualer som sedan ses som utsignaler från systemet. Dessa residualer ska vara känsliga för vissa fel men inte för andra, samt okänsliga för störningar. Om de fel som inte ska påverka residualen ses som störningar så uppfyller kvotsystemet dessa krav, se figur 3.4. Detta ska jämföras med figur 2.1 som var ursprungsmodellen. Notera att till skillnad mot figur 2.1 så finns ingen störning kvar som påverkar kvotsystemet.



Figur 3.4: *Modellbaserad diagnos av kvotsystemet. Felet är den enda okända variabeln som påverkar processen.*

Kapitel 4

Matematisk beskrivning av modellen

I det här kapitlet följer en beskrivning av tillvägagångssättet för att kunna skapa transformationerna, transformera systemet samt ge ut kvotsystemet. Skapandet av transformationerna görs i 7 steg enligt tabell 4.1. Först tar man fram en distribution Δ och använder sedan den för att ta fram transformationerna.

Tabell 4.1: *De steg som ska gås igenom för att skapa transformationerna*

1)	Räkna fram distributionen Δ.	Enkelt
2)	Kontrollera att l inte ingår i Δ.	Enkelt
3)	Fyll ut matrisen som representerar Δ till en kvadratisk inverterbar matris.	Enkelt
4)	Ta fram Φ_1 genom att lösa partiella differentialekvationer. $Span\{dh\} = \Delta^{\perp}$.	Svårt
5)	Hitta Ψ_1 genom att lösa ekvationen $\Delta^{\perp} \cap span\{dh\} = span\{d(\Psi_1 \circ h)\}$	Svårt
6)	Fyll ut $\Psi = \{\Psi_1(y), H2y\}$ så att Ψ blir mjuk och inverterbar.	Enkelt
7)	Fyll ut $\Phi = \{\Phi_1(x), H2h(x), \Phi_3(x)\}$ så att Φ blir mjuk och inverterbar.	Enkelt

4.1 Skapa distributionen

Enligt steg 1 i tabell 4.1 tar man fram distributionen Δ, vilken beräknas enligt algoritm (4.1)

$$\begin{aligned} \Delta_0 &= \bar{P} \\ \Delta_{k+1} &= \bar{\Delta}_k + \sum_{i=0}^{a} [g_i, \bar{\Delta}_k \cap Ker\{dh\}] \end{aligned} \tag{4.1}$$

där $P = span\{p\}$ och $\bar{P}$ står för involutiva höljet av P. Likaså står $\bar{\Delta}_k$ för involutiva höljet av Δ_k. Summationen $\sum_{i=1}^{a}[g_i, \bar{\Delta}_k \cap Ker\{dh\}]$ bidrar till att det transformerade systemet kommer att bli villkorligt invariant. Villkorligt invariant eller f,h-invariant innebär att $[g_i, \Delta \cap Ker\{dh\}] \in \Delta$ ska vara uppfyllt för alla $i = 0, \ldots, a$ där $g_0 = f$.

Antag att x_0 är en icke-singulär punkt med en icke-singulär omgivning runt vilken man vill göra transformationen. Då kommer iterationen att stanna efter max n steg, där n är antalet tillstånd. Iterationen stannar då summationen i algoritm (4.1) inte längre tillför någon ny dimension, dvs.

$$\Delta_{k^*+1} = \bar{\Delta}_k^* \tag{4.2}$$

Om istället den angivna punkten är singulär finns det inga garantier för att iterationen någonsin kommer att stanna. Efter n steg har man däremot skapat en distribution som gäller för den del av definitionsmängden som inte innehåller singulära punkter. På grund av detta är programmet skapat för att maximalt iterera n gånger. Matrisen som representerar Δ har nu maximalt n kolumner.

Fördelen med att bygga upp Δ på det här sättet är att man inte i förväg behöver definiera något koordinatsystem, utan koordinatsystemet byggs upp av kolonnvektorerna i matrisen som representerar Δ.

Om man antar att punkten x_0 samt en omgivning runt denna är icke-singulär så kommer Δ efter iterationen ha följande egenskaper:

- span$\{p\} \subset \Delta$
- span$\{l\} \not\subseteq \Delta$
- Δ är villkorligt invariant, involutivt och icke-singulärt

Ett ytterligare antagande som görs för att kunna hitta transformationerna är följande:

- Det finns en funktion Ψ_1 så att $\Delta^{\perp} \cap span\{dh\} = span\{d(\Psi \circ h)\}$ där $\Delta^{\perp}$ är annihilatorn till Δ

Skapandet av Δ borde alltid fungera varav detta steg anses enkelt. Det som skulle kunna vara en svårighet är att lägga till dimensioner till Δ (plusstecknet i algoritm (4.1)), vilket innebär att lösa ett linjärt ekvationssystem. I de fall som testats har dock inga problem uppkommit.

Distributionen Δ innehåller de delar som man vill göra systemet okänsligt för, nämligen störningarna. Eftersom man vill göra kvotsystemet så stort som möjligt vill man göra Δ så litet som möjligt. För att felen ska bli detekterbara så ska $l \notin \Delta$, varav man i steg 2 av tabell 4.1 kontrollerar detta. Om $l(x)$ skulle ligga i Δ så skulle inte ett system med kriterierna, att störningen inte ska påverka systemet medan felen ska göra det, kunna skapas. Därmed kan man lika gärna sluta här.

Om $l(x)$ däremot skulle tillhöra $\Delta^{\perp}$ (dvs. annihilatorn till Δ) så existerar ett sådant system och därmed vill vi hitta transformationerna samt transformera systemet.

4.2 Transformationerna

Nu har det blivit dags att skapa transformationerna. Enligt den metod jag följer ska transformationerna se ut enligt (4.3) och (4.4) där (4.3) transformerar tillstånden och (4.4) transformerar utsignalen.

$$\Phi(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \Phi_1(x) \\ H_2 h(x) \\ \Phi_3(x) \end{pmatrix} \tag{4.3}$$

$$\Psi(y) = \begin{pmatrix} yy_1 \\ yy_2 \end{pmatrix} = \begin{pmatrix} \Psi_1(y) \\ H_2 y \end{pmatrix} \tag{4.4}$$

Först gör man en transformation av hela systemet med hjälp av $\Phi(x)$ varefter man transformerar utsignalerna med hjälp av $\Psi(y)$.

4.2.1 Transformation av tillstånden

I ett första steg vid beräkningen av Φ beräknar man $\Phi_1(x)$. Detta görs genom att lösa partiella differentialekvationer enligt ekvation (4.5). Sedan fyller man ut med $H_2 h(x)$ samt en godtyckligt vald funktion $\Phi_3(x)$ så att avbildningen Φ blir inverterbar.

För att hitta $\Phi_1(x)$ ska man räkna fram de $n - d$ oberoende lösningarna till den partiella differentialekvationen (4.5), där d är dimensionen av Δ

$$\frac{\partial \lambda_i}{\partial x} F(x) = 0 \tag{4.5}$$

Matrisen $F(x) = [f_1, f_2, \ldots, f_d]$ medan $span\{f_i\} = \Delta$. Ekvation (4.5) löses enligt en specifik metod som beskrivs i [2]. Metoden går i stort ut på att man först fyller ut matrisen som representerar Δ till en kvadratisk matris, enligt steg 3 i tabell 4.1. Denna utfyllnad är trivial och går alltid att utföra. Genom att sedan följa flöden kan man få fram dessa λ som är de $n - d$ oberoende lösningarna.

Problemet att hitta $n - d$ oberoende lösningar av differentialekvationen kan bli omformulerat till att en icke-singulär distribution Δ ska ha en annihilator $\Delta^{\perp}$ vilken spänns upp av exakta differentialer enligt

$$\Delta^{\perp} = Span\{d\lambda_1 \dots d\lambda_{n-d}\}. \tag{4.6}$$

Vidare så kan man enligt den här metoden skapa Φ_1 enligt

$$\Delta^{\perp} = Span\{d\Phi_1\}. \tag{4.7}$$

Jämför man ekvationerna 4.6 och 4.7 kan man se att Φ_1 byggs upp av dessa λ. Beräknandet av Φ_1 svarar mot steg 4 i tabell 4.1. Vid detta steg skulle man kunna få problem att lösa ut Φ_1 varav detta steg anses svårt. En av svårigheterna innefattar en invertering av en avbildning, se avsnitt 7.1

Skapandet av H_2 beskrivs närmare i avsnitt 4.2.2. Matrisen H_2 multipliceras sedan med y.

Matrisen $\Phi_3(x)$ ska nu väljas så att matrisens radvektorer är linjärt oberoende av varandra, dvs. så att avbildningen $\Phi(x)$ blir inverterbar. Storleken av $\Phi_3(x)$ är $n - n_1 - n_2$ där n är antalet tillstånd, n_1 är antalet rader i $\Phi_1(x)$ medan n_2 är antalet rader $H_2h(x)$. Avbildningen $\Phi_3(x)$ kan alltid väljas som någon eller några x-variabler. Man får dock inte välja någon x-variabel som ensamt linjärt ingår i Φ_1 eller $H_2h(x)$, eller med andra ord differentialen till Φ ska ha full rang. Utfyllnaden av Φ (steg 7 i tabell 4.1) är ett trivialt steg som alltid går att utföra.

4.2.2 Transformation av utsignalen

Transformationen $yy = \Psi(y)$ löses enligt (4.4). I ett första steg ska Ψ_1 beräknas ur ekvation (4.8)

$$\Delta^{\perp} \cap span\{dh\} \quad = \quad span\{d(\Psi_1 \circ h)\} \tag{4.8}$$

där $y = h(x)$. I det linjära fallet går det alltid att hitta transformationen, medan det i det olinjära fallet inte finns några garantier för att transformationen existerar. Skapandet av Ψ_1 beskrivs närmare i avsnitt 7.3. Detta motsvarar steg 5 i tabell 4.1, vilket kan ställa till en del problem då Ψ_1 inte alltid kan skapas.

Matrisen H_2y ska nu väljas så att matrisens radvektorer är linjärt oberoende av varandra samt så att matrisen blir inverterbar. Storleken av H_2y är n_2, medan storleken av Ψ_1 är $q - n_2$ där q är antalet utsignaler. Matrisen H_2y kan alltid väljas som någon eller några y-variabler. Man får dock inte välja någon y-variabel som ensamt linjärt ingår i Ψ_1, eller med andra ord, differentialen till Ψ ska ha full rang. Denna utfyllnad är trivial och går alltid att utföra (steg 6 i tabell 4.1).

I och med skapandet av H_2y har man också skapat H_2 vilken används vid beräkningarna av tillståndstransformationen Φ. Matrisen H_2 är någon eller några rader i enhetsmatrisen av storleken $n_2 \times n_2$.

4.3 Kvotsystemet

Efter att ha transformerat systemet har vi fått ett system på formen (3.2) och genom att ta ut $\dot{z}_1$ och yy_1 samt byta ut z_2 mot yy_2 så får man ut kvotsystemet (3.3). Kvotsystemet är den delen som ur diagnostisk synvinkel är intressant, eftersom den innehåller de delar som går att övervaka.

Kapitel 5

Funktionsflöde

Syftet med examensarbetet var att göra en implementation i Mathematica som skapar transformationerna, transformerar systemet samt ger ut kvotsystemet. I bilaga A finns alla variabler och funktioner uppställda. Nedan följer en beskrivning av när de olika funktionerna används samt vilka som är relaterade till varandra.

Om allt går som det ska är det enklast att använda sig av nedanstående funktioner för att räkna fram det transformerade systemet, ta ut den delen som tillhör kvotsystemet (resultatet fås i matrisform) alternativt skriva ut kvotsystemet på tillståndsform med struktur enligt tillståndsform (3.3).

$$FaultDetectionTransformation[xvar, yvar, g, p, l, h, x0]$$

$$FDTQuotientSystem[resultFDT]$$
$$FDTPrintQuotientSystem[resultFDT, xvar, g, l, p]$$

Eftersom beräkningarna innehåller en del svårigheter så kommer inte alltid Mathematica att kunna utföra alla stegen som nämndes i tabell 4.1. Funktionen FaultDetectionTransformation kan därför delas upp i flera delsteg så att användaren kan hjälpa Mathematica där den får problem, genom att räkna ut vissa steg för hand.

En första uppdelning av FaultDetectionTransformation är en funktion som räknar ut transformationerna och en funktion som transformerar systemet. Om man har transformerat systemet med hjälp av $TransformSystem$ måste man använda sig av nedanstående funktioner för att ta fram kvotsystemet (resultatet fås i matrisform) alternativt skriva ut kvotsystemet på tillståndsform

med struktur enligt tillståndsform (3.3).

$ComputeTransformations[xvar, yvar, g, p, l, h, x0]$
$TransformSystem[\Psi, \Phi, D\Phi_1, DH2, xvar, yvar, g, p, l, h, x0]$

$TSQuotientSystem[D\Psi_1, DH2, D\Phi_1, D\Phi_3, resultTS]$
$TSPrintQuotientSystem[D\Psi_1, DH2, D\Phi_1, D\Phi_3, resultTS, g, l, p]$

En andra uppdelning är att dela upp skapandet av transformationen i de 7 delsteg som nämndes i tabell 4.1, varefter man kan transformera systemet med hjälp av $TransformSystem$.

$\Delta Solve[p, h, g, xvar]$
$FaultDetectableQ[\Delta, l]$
$CreateExtension[\Delta, x0, xvar]$
$\Phi 1Solve[\Delta, F, x0, xvar]$
$\Psi 1Solve[\Delta, h, x0, xvar, yvar]$
$\Psi andH2Solve[\Psi_1, yvar]$
$\Phi Solve[\Phi_1, H2, h, xvar]$
$TransformSystem[\Psi, \Phi, D\Phi_1, DH2, xvar, yvar, g, p, l, h, x0]$

$TSQuotientSystem[D\Psi_1, DH2, D\Phi_1, D\Phi_3, resultTS]$
$TSPrintQuotientSystem[D\Psi_1, DH2, D\Phi_1, D\Phi_3, resultTS, g, l, p]$

Om man vill kontrollera om transformationsmatriserna uppfyller de kriterier man hade när man skapade transformationerna kan nedanstående funktion användas.

$CheckTransformations[resultComputeTransformations]$

Funktionen testar om Δ är singulär, om l tillhör Δ, om $d\Phi_1\Delta = 0$, om villkoret $\Delta^{\perp} \cap span\{dh\} = span\{d(\Psi_1 \circ h)\}$ är uppfyllt samt om Ψ och Φ har rätt dimensioner.

Kapitel 6

Exempel

I det här kapitlet kommer ett exempel att visas. Eftersom programmet är skrivet i Mathematica (ett datorverktyg för symbolisk formelmanipulation) kommer en del Mathematica-specifika kommandon att beskrivas. Alla kommandon kommer att visas med kursiv stil, med svaren direkt efterföljande varefter en förklarande text finnes. Ett semikolon i slutet av en rad innebär att svaret inte skrivs ut.

Det här exemplet är hämtat från [1] och visar hur man designar ett detektionsfilter för att kunna upptäcka fel i en av styrmotorerna baserad på en punktmassemodell av en satellit.

6.1 Definiera systemet

Modellen ser ut enligt följande:

$$
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= x_1 x_4 - \frac{\theta_1}{x_1^2} + \theta_2 u_1 + p \\
\dot{x}_3 &= x_3 \\
\dot{x}_4 &= -\frac{2 x_2 x_4}{x_1} + \frac{\theta_2}{x_1} u_2 + \frac{\theta_2}{x_1} 1 \\
y_1 &= x_1 \\
y_2 &= x_2 \\
y_3 &= x_3
\end{aligned}
$$

Här står (x_1, x_3) för satellitens position i polära koordinater, x_2 för radiella hastigheten och x_4 för vinkelhastigheten, varav signalerna x_1, x_3 och x_4 är mätbara.

I bilaga A finns de variabler som ska definieras. Det skulle kunna se ut enligt följande:

$$\begin{aligned} xvar &= \{x_1, x_2, x_3, x_4\}; \\ yvar &= \{y_1, y_2, y_3\}; \\ f &= \{x_2, x_1 x_4^2 - \theta/x_1^2, x_4, -2x_2 x_4/x_1\}; \\ g_1 &= \{0, \theta_2, 0, 0\}; \\ g_2 &= \{0, 0, 0, \theta_2/x_1\}; \\ g &= \{f, g_1, g_2\}; \\ p &= \{\{0, 1, 0, 0\}\}; \\ l &= \{\{0, 0, 0, \theta_2/x_1\}\}; \\ h &= \{x_1, x_3, x_4\}; \\ x_0 &= \{1, 0, 0, 0\}; \end{aligned}$$

$xvar$ är tillstånden, $yvar$ är utsignalen, $g = \{f, g_1, g_2\}$, p är störningen, l är felen, $y = h(x)$ och x_0 är begynnelsevillkoret.

För att kunna isolera felen ifrån varandra krävs att det är färre fel än vad det är insignaler. För att inte störningarna ska påverka utsignalen krävs att l inte tillhör Δ. Vektorn x_0 talar om runt vilken punkt man ska utföra transformationen och måste väljas så att den blir icke-singulär.

6.2 Ladda paket

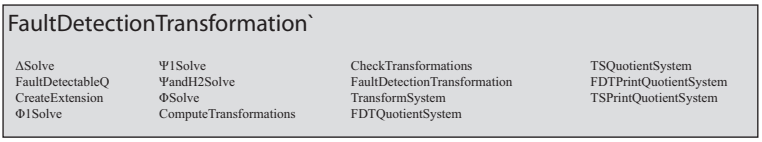
För att kunna köra de funktioner som nämndes i kapitel 5 måste paketet FaultDetectionTransformation laddas in, vilket görs med kommandot:

$$Needs[FaultDetectionTransformation`"]$$

För att se de funktioner man kan använda sig av skriver man:

$$?FaultDetectionTransformation`*$$

Då får man upp en lista över alla funktionerna som finns tillgängliga, se figur 6.1. Klickar man på namnen kommer det upp en förklarande text vad funktionen gör och vad den vill ha för indata.



Figur 6.1: *De funktioner som paketet FaultDetectionTransformation innehåller.*

6.3 Beräknandet av transformationerna

Beräknandet av transformationerna sker i 7 steg enligt tabell 4.1. Varje funktion går igenom ett av dessa steg.

1. Skapa Δ.

$$\{\Delta, \Delta singular\} = \Delta Solve[p, h, g, xvar]$$

$$\Delta = span(f_1, f_2) = span \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \frac{2x_4}{x_1} & 0 \end{pmatrix}$$

$$\Delta singular = False$$

Här kan man se att Δ byggs upp av två linjärt oberoende vektorer. I detta fall är $\Delta = \{f_1, f_2\}$ där $f_1 \neq \alpha f_2$ där α är någon konstant. Vektorerna är också linjärt oberoende runt punkten x_o vilket ΔSingular kontrollerar. I detta fall var $x_0 = \{1, 0, 0, 0\}$.

$$(f_1, f_2)|_{x=x_0} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

2. Kontrollera så inte l ligger i Δ.

$$FaultDetectableQ = FaultDetectableQ[\Delta, l]$$

$$FaultDetectableQ = False$$

Här kan man se att

$$(f_1, f_2, l) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ \frac{2x_4}{x_1} & 0 & \frac{\theta_2}{x_1} \end{pmatrix}$$

är linjärt oberoende av varandra, dvs. l kan inte skrivas som en linjärkombination av f_1 och f_2 ($l \neq \alpha f_1 + \beta f_2$ där α och β är konstanter).

3. Fyll ut matrisen som representerar Δ så att en kvadratisk matris skapas.

$$F_{ext} = CreateExtension[\Delta, x0, xvar]$$

$$F_{ext} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{2x_4}{x_1} & 0 & 0 & 1 \end{pmatrix}$$

Här har matrisen som representerar Δ blivit utfyllt till en kvadratisk matris med hjälp av rader från enhetsmatrisen av storleken 4×4. Radvektorerna väljs så att utfyllnaden får full rang i punkten x_0.

4. Skapa Φ_1.

$$\Phi_1 = \Phi_1 Solve[\Delta, F, x0, xvar]$$

$$\Phi_1 = \begin{pmatrix} x_3 \\ x_1^2 x_4 \end{pmatrix}$$

Matrisen Φ_1 har skapats genom att räkna fram de $n-d$ oberoende lösningarna till den partiella differentialekvationen

$$\frac{\partial \lambda_i}{\partial x} F(x) = 0$$

där $F = \{f_1, f_2\}$. I detta fallet är $n = 4$ och $d = 2$. Genom att sätta in λ i ekvatonen kan vi se att λ är en lösning.

$$\frac{\partial \lambda_1}{\partial x} F(x) = (0,0,1,0) \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \frac{2x_4}{x_1} & 0 \end{pmatrix} = 0$$

$$\frac{\partial \lambda_2}{\partial x} F(x) = (2x_1x_4, 0, 0, x_1^2) \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \frac{2x_4}{x_1} & 0 \end{pmatrix} = 0$$

5. Skapa Ψ_1.

$$\Psi_1 = \Psi_1 Solve[\Delta, h, x0, xvar, yvar]$$

$$\Psi_1 = \begin{pmatrix} y_2 \\ y_1^2 y_3 \end{pmatrix}$$

Ψ_1 har lösts ut från ekvation $\Delta^\perp \cap span\{dh\} = span\{d(\Psi_1 \circ h)\}$. Annihilatorn till Δ (dvs. $\Delta^\perp$) spänns upp av vektorer som är vinkelräta mot Δ. Vi söker alltså vektorer som inte är en linjärkombination av vektorerna som spänner upp Δ.

$$\Delta = span \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \frac{2x_4}{x_1} & 0 \end{pmatrix}$$

$$\Delta^\perp = span \begin{pmatrix} 0 & \frac{2x_4}{x_1} \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Snittet mellan $\Delta^\perp$ och dh innebär de vektorer som spänner upp dimensioner som finns med i både $\Delta^\perp$ och dh. I detta fall kan vektorerna i $\Delta^\perp$ byggas upp av vektorerna i dh. Därmed är $\Delta^\perp \cap span\{dh\} = \Delta^\perp$.

$$\Delta^{\perp} = span \begin{pmatrix} 0 & \frac{2x_4}{x_1} \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$span\{dh\} = span \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Delta^{\perp} \cap span\{dh\} = \Delta^{\perp} = span \begin{pmatrix} 0 & \frac{2x_4}{x_1} \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Det som ska lösas är funktionen $\Delta^{\perp} \cap span\{dh\} = span\{d\gamma(x)\}$ där $\gamma(x) = \Psi_1 \circ h(x)$. Detta har samma lösningsmetod som lösningen av Φ_1 ty $\Delta^{\perp} = span\{d\Phi_1(x)\}$. Eftersom snittet $\Delta^{\perp} \cap span\{dh\}$ i detta fall var $\Delta^{\perp}$ så blir $\gamma = \Phi_1$. Avbildningen $\Psi_1(y)$ fås sedan genom att sätta $x_1 = y_1, x_3 = y_2$ och $x_4 = y_3$ eftersom $h(x) = \{x_1, x_3, x_4\}$.

$$\gamma(x) = \Psi_1 \circ h(x) = \Psi_1(h(x))$$

$$\gamma(x) = \begin{pmatrix} x_3 \\ x_1^2 x_4 \end{pmatrix} = \Psi_1(y_1, y_2, y_3)|_{y_1=x_1, y_2=x_3, y_3=x_4}$$

$$\Psi_1(y) = \begin{pmatrix} y_2 \\ y_1^2 y_3 \end{pmatrix}$$

6. Skapa Ψ och H_2.

$$\{\Psi, H_2\} = \Psi and H2Solve[\Psi_1, yvar]$$

$$\Psi(y) = \begin{pmatrix} y_2 \\ y_1^2 y_3 \\ y_1 \end{pmatrix}$$

$$H_2 = (1, 0, 0)$$

Genom att fylla ut Ψ_1 med $H_2 y$ så skapas Ψ. Utfyllnaden av Ψ_1 ska göras så att de ingående termerna blir linjärt oberoende av varandra samt så att Ψ får

samma dimension som utsignalerna vilket i detta fall är $\{y_1, y_2, y_3\}$. Vektorn H_2 är derivatan av $H_2 y$ med avseende på y-variablerna, i detta fallet $\frac{\partial y_1}{\partial y}$.

7. Skapa Φ.

$$\Phi = \Phi Solve[\Phi_1, H2, h, xvar]$$

$$\Phi = \begin{pmatrix} x_3 \\ x_1^2 x_4 \\ x_1 \\ x_2 \end{pmatrix}$$

Genom att fylla ut Φ_1 med $H_2 h(x)$ och Φ_3 så skapas Φ. Utfyllnaden av Φ_1 ska göras så att de ingående termerna blir linjärt oberoende av varandra samt så att Φ får samma dimensionen som tillståndsvariablerna vilket i detta fall är $\{x_1, x_2, x_3, x_4\}$.

Funktionen

$$\{\Delta, \Delta Singular, FaultDetectedQ, F, \Phi_1, \Psi_1, \Psi, H2, \Phi\} = ComputeTransformations[xvar, yvar, g, p, l, h, x0];$$

löser alla dessa 7 steg i en följd.

De två transformationerna $\Phi(x)$ och $\Psi(x)$ har nu skapats.

$$\Phi = \begin{pmatrix} x_3 \\ x_1^2 x_4 \\ x_1 \\ x_2 \end{pmatrix}$$

$$\Psi(y) = \begin{pmatrix} y_2 \\ y_1^2 y_3 \\ y_1 \end{pmatrix}$$

6.4 Transformation av systemet

Transformerar man systemet med hjälp av transformationerna $\Phi(x)$ och $\Psi(x)$ så får man en tillståndsform enligt följande:

$$
\begin{aligned}
\dot{z}_{11} &= \frac{z_{12}}{z_2^2} \\
\dot{z}_{12} &= z_2\theta_2 u_2 + z_2\theta_2 m \\
\dot{z}_2 &= z_3 \\
\dot{z}_3 &= \frac{z_{12}^2 - z_2\theta_1}{z_2^3} + \theta_2 u_1 + w \\
yy_{11} &= z_{11} \\
yy_{12} &= z_{12} \\
yy_2 &= z_2
\end{aligned}
$$

Transformationen kan göras med hjälp av funktionen

$$
\begin{aligned}
&\{ff, ggllpp, yytemp, yy\} = \\
&TransformSystem[\Psi, \Phi, D\Phi_1, DH2, xvar, yvar, g, p, l, h, x0]
\end{aligned}
$$

$$
ff = \begin{pmatrix} \frac{z_{12}}{z_2^2} \\ 0 \\ z_3 \\ \frac{z_{12}^2 - z_2\theta_1}{z_2^3} \end{pmatrix}
$$

$$
ggllpp = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & z_2\theta_2 & z_2\theta_2 & 0 \\ 0 & 0 & 0 & 0 \\ \theta_2 & 0 & 0 & 1 \end{pmatrix}
$$

$$
yytemp = \begin{pmatrix} z_2 \\ z_{11} \\ \frac{z_{12}}{z_2^2} \end{pmatrix}
$$

$$
yy = \begin{pmatrix} z_{11} \\ z_{12} \\ z_2 \end{pmatrix}.
$$

Vektorn ff är $f(x)$ efter tillståndstransformationen medan $ggllpp$ är $g(x)$, $l(x)$ och $m(x)$ efter tillståndstransformationen. Vektorn $yytemp$ innehåller utsignalen efter tillståndstransformationen, men innan utsignalstransformationen, medan yy är utsignalen efter utsignalstransformationen. Funktionen

$$\{\Delta, \Delta Singular, FaultDetectedQ, F, \Phi_1, \Psi_1, \Psi, H2, \Phi,$$
$$ff, ggllpp, yytemp, yy\} =$$
$$FaultDetectionTransformation[xvar, yvar, g, p, l, h, x0];$$

både skapar transformationen och transformerar systemet. Den ger ut samma resultat som $TransformSystem$, men ger även ut resultatet från de 7 delstegen vid framtagningen av transformationerna.

6.5 Framtagning av kvotsystemet

Utifrån det transformerade systemet kan man nu ta fram kvotsystemet, vilket i detta fall blir:

$$\dot{z}_{11} = \frac{z_{12}}{yy_2^2}$$
$$\dot{z}_{12} = z_2\theta_2 u_2 + z_2\theta_2 m$$
$$yy_{11} = z_{11}$$
$$yy_{12} = z_{12}$$

Detta kan göras med hjälp av 2 varianter på funktioner. En som ger ut kvotsystemet i matrisform och en som skriver ut kvotsystemet på tillståndsform. Om man har transformerat systemet med hjälp av $TransformSystem$ ska man använda funktionerna

$$\{ff, ggllpp, yy\} =$$
$$TSQuotientSystem[D\Psi_1, DH2, D\Phi_1, D\Phi_3, resultTS]$$

$$ff = \begin{pmatrix} \frac{z_{12}}{z_2^2} \\ 0 \end{pmatrix}$$
$$ggllpp = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & z_2\theta_2 & z_2\theta_2 & 0 \end{pmatrix}$$
$$yy = \begin{pmatrix} z_{11} \\ z_{12} \end{pmatrix}$$

eller

$$TSPrintQuotientSystem[D\Psi_1, DH2, D\Phi_1, D\Phi_3, resultTS, g, l, p];$$

och om man har använt funktionen $FaultDetectionTransformation$ för transformeringen så ska man använda funktionerna

$$FDTQuotientSystem[resultFDT];$$

eller

$$FDTPrintQuotientSystem[resultFDT, xvar, g, l, p];$$

vid framtagning av kvotsystemet. Resultaten ser här likadana ut som hos funktionerna $TSQuotientSystem$ och $TSPrintQuotientSystem$.

I ovan nämnda funktioner står $resultTS$ för resultatet från funktionen $TransformSystem$ medan $resultFDT$ står för resultatet från funktionen $FaultDetectionTransformation$.

6.6 Kontrollera transformationen

Med hjälp av funktionen

$$CheckTransformations[resultComputeTransformations]$$

$$Seams\ correct.$$

kan man kontrollera om transformationerna uppfyller villkoren för att kunna skapa ett system där felen är separerade och störningarna inte påverkar. Funktionen kontrollerar om Δ är singulär, om l tillhör Δ, om $d\Phi_1\Delta = 0$, om villkoret $\Delta^{\perp} \cap span\{dh\} = span\{d(\Psi_1 \circ h)\}$ är uppfyllt samt om Ψ och Φ har rätt dimensioner. Dessa tester gicks igenom vid förklaringarna till de 7 delstegen i avsnitt 6.3.

Kapitel 7

Problem som uppkommit

Det här kapitlet tar upp de problem som uppkommit under arbetets gång med att skapa ett program i Mathematica för att ta fram transformationerna Φ och Ψ, transformera systemet samt ge ut kvotsystemet.

7.1 FrobSolve och DSolve

$FrobSolve$ är en egenutvecklad funktion (på Fordonssystem vid Linköpings universitet) som räknar ut lösningarna till den partiella differentialekvationen (4.5). Funktionen används bland annat vid skapandet av Φ_1. Som ett första steg räknar man ut flöden $x(t) = \Phi_t^f(x^o)$ vilka löser ett antal ordinära differentialekvationer $\dot{x} = f(x)$ med initialvillkoret $x(0) = x^0$ [2]. Detta är generellt ett svårt problem. Därefter följer man flödena och får då ut avbildningen Φ_1^{-1}. Därefter ska avbildningen Φ_1^{-1} inverteras för att få ut Φ_1. Detta är ekvivalent med att lösa ett generellt olinjärt ekvationssystem, vilket också kan vara ett svårt problem.

På grund av detta fanns en misstanke om att funktionen skulle kunna få svårigheter att lösa de partiella differentialekvationerna som fördes in till funktionen. Därför gjordes en del försök att hitta alternativa möjligheter. En variant som testades var att lösa partiella differentialekvationer med hjälp av Mathematicas inbyggda funktion $DSolve$. Det visade sig dock att $DSolve$ inte klarade av att ta emot ett system av partiella differentialekvationer, vilket var ett krav vid problemställningen. Under den begränsade testningen som gjorts har däremot $FrobSolve$ klarat av att lösa de partiella differentialekvationerna.

7.2 En icke-singulär punkt antas ha en icke-singulär omgivning

För att kontrollera om en vektor tillhör en distribution används funktionen $DistMemberQ$, egenutvecklad på Fordonssystem vid Linköpings universitet. På några ställen har dock inte denna metod kunnat användas bland annat på grund av att $DistMemberQ$ inte tar hänsyn till runt vilken punkt detta ska kontrolleras. Ett sådant ställe är då distributionen Δ kontrolleras så att dess vektorer är linjärt oberoende runt punkten x_0. Här har istället gjorts en kontroll så att distributionens rang är samma som antalet vektorer, dvs. ingen av vektorerna får kunna skrivas som en linjärkombination av de övriga vektorerna i punkten x_0. Denna metod gör bara en kontroll så att punkten x_0 är icke-singulär, men borde egentligen även kontrollera så att ett område runt punkten är icke-singulärt. Här görs ett antagande att det som gäller för punkten även gäller för ett område runt denna.

7.3 Problem att skapa Ψ_1

Ett grundantagande för att den metod som beskrivits ska fungera är att det existerar ett Ψ_1 så att

$$\Delta^{\perp} \cap span\{dh\} = span\{d(\Psi_1 \circ h)\}$$

blir uppfyllt [1], där Δ är den distributionen som bestämdes i steg 1 av tabell 4.1. I fallet med ett linjärt system så existerar alltid ett sådant Ψ_1. Om vi sätter $h(x) = Cx$ och $(\Psi_1 \circ h) = \Psi_1 h(x) = \Psi_1 Cx$ så kan ekvationen skrivas om enligt

$$\Delta^{\perp} \cap Im\, C = Im\, (\Psi_1 C)$$

eftersom $d(Cx) = C$ och $d(\Psi_1 Cx) = \Psi_1 C$. Här kan man se att $\Delta^{\perp} \cap Im\, C$ är en delmängd av $Im\, C$, eftersom snittet av 2 distributioner aldrig är större än den ena distributionen dvs.

$$\Delta^{\perp} \cap Im\, C \ \subseteq \ Im\, C.$$

Snittet $\Delta^{\perp} \cap ImC$ spänns alltså upp av en linjärkombination av C:s rader varav Ψ_1 alltid går att lösa.

Om man följer samma resonemang i det olinjära fallet så gäller

$$\Delta^{\perp} \cap span\{dh\} = span\{d(\Psi_1 \circ h)\} \subseteq span\{dh\}$$

samt

$$d(\Psi_1 \circ h) = \frac{\partial}{\partial x}(\Psi_1 \circ h) = \frac{\partial}{\partial x}\Psi_1(h(x)) = \frac{\partial \Psi_1}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial \Psi_1}{\partial y}dh(x).$$

Detta ekvationssystem går alltid att lösa för $\frac{\partial \Psi_1}{\partial y}$ eftersom

$$span\{\frac{\partial \Psi_1}{\partial y}dh(x)\} \subseteq span\{dh\}.$$

Däremot är det inte självklart att det går att lösa ut $\Psi_1(y)$ ifrån $\frac{\partial \Psi_1}{\partial y}$. Kravet för detta är att det existerar en potential för ett vektorfält vilket har strikta existenskrav [2].

På grund av detta har en annan väg valts. Om variablerna som ingår i $\Psi_1(h(x))$ (kalla dem för x') går att lösa ut ur $y = h(x)$, dvs. lösa ekvationssystemet $y = h(x)$ med avseende på x'-variablerna, har man ett tillräckligt villkor för att hitta Ψ_1. Detta kan omformuleras i att $\frac{\partial h}{\partial x'}$ ska ha full kolumnrang. Denna metod ställer dock onödigt starka krav vilket kan leda till att metoden inte hittar någon lösning trots att det existerar en. Det har visat sig att den här metoden inte fungerar i det generella linjära fallet varav en utveckling av systemet skulle vara möjlig.

Kapitel 8

Utvecklingsmöjligheter och slutsatser

I det här kapitlet diskuteras möjligheten till utveckling av programmet, sådant som inte hunnits med under arbetets gång.

8.1 Förbättra skapandet av Ψ_1

Problemet vid skapandet av Ψ_1 beskrevs i avsnitt 7.3. Den metod som är implementerad fungerar inte ens i det generella linjära fallet. Däremot kunde man bevisa att Ψ_1 alltid är lösbart i det linjära fallet. På grund av detta skulle man kunna göra en implementering av det generella linjära fallet och låta användaren avgöra vilken lösningsmetod som passar bäst för det specifika systemet.

8.2 Ange singulariteten

I det fall då delar av definitionsmängden innehåller singulära punkter skulle området för denna singularitet kunna anges. Om matrisen som representerar distributionen Δ är kvadratisk är detta enkelt då singulariteten fås genom att ta determinanten av matrisen som representerar distributionen, se exempel 1.

Exempel 1:

$$\begin{aligned} \Delta &= span\left\{ \begin{pmatrix} 1 \\ x_1 \end{pmatrix}, \begin{pmatrix} x_1 \\ 1-x_2 \end{pmatrix} \right\} \\ det\left\| \begin{matrix} 1 & x_1 \\ x_1 & 1-x_2 \end{matrix} \right\| &= 1(1-x_2) - x_1^2 = 1 - x_2 - x_1^2 \end{aligned}$$

I detta fall är punkterna på kurvan $1 - x_2 - x_1^2$ singulära. Är inte matrisen kvadratisk så blir problemställningen mer komplicerad. En generell metod för detta skulle kunna vara en möjlig utveckling av programmet.

Anledningen till att man vill kunna finna alla singulära punkter beror på att den metod som beskrivs i [1] gör ett antagande att transformationen sker runt en icke-singulär punkt. Om man vet vilka punkter som är singulära kan man enkelt kontrollera om punkten man valt är icke-singulär samt kunna hjälpa användaren att hitta en punkt som är icke-singulär.

8.3 Ett steg som inte är implementerat

Efter skapandet av Δ enligt avsnitt 4.1 säger metoden enligt [1] att iteration (8.1) ska köras. Denna iteration liknar iterationen vid skapandet av Δ och innehåller också liknande operationer.

$$\begin{aligned} \Omega_0 &= \Theta \cap span\{dh\} \\ \Omega_{k+1} &= \Theta \cap \left(\sum_{i=0}^{a} [L_{g_i} \Omega_k + span\{dh\} \right) \end{aligned} \tag{8.1}$$

Iterationen stannar efter max k^* steg där $k^* \leq n - 1$ och $k^* \leq k$ så att $\Omega_k = \Omega_{k^*}$. Därefter kan man enligt tidigare skapa en kvadratisk matris och sedan hitta transformationerna. Anledningen till att man använder sig av även detta steg är för att kvotsystemet kan innehålla delar som inte går att övervaka. Använder man även denna iteration får man ut den delen av kvotsystemet som i princip är observerbart. Det linjära fallet kommer bli observerbart medan det olinjära fallet i specialfall kan innehålla delar som inte är observerbara trots att 8.1 har körts [1].

8.4 Linjärt system

Vid fallet då vi har ett linjärt system blir beräkningarna enklare i vissa delar av programmet. Bland annat blir beräkningarna av $\Phi(x)$ enklare då $\Delta^{\perp} = d\Phi_1(x)$ samt $\Psi_1(y)$ kommer alltid att existera, vilket nämndes i avsnitt 8.1. På grund av detta borde användaren kunna ange om systemet är linjärt eller inte, alternativt att programmet automatiskt kontrollerar detta.

8.5 Punkten X_0

Programmet är skapat så att punkten x_0 måste anges. En förbättring av programmet skulle vara om användaren själv fick bestämma om han eller hon vill använda sig av punkten eller inte. Om inte x_0 anges ses den som en symbol av Mathematica. Med hjälp av ett test om x_0 är en symbol eller inte på

specifika ställen i programmet och sedan olika lösningsmetoder skulle kunna vara en tänkbar utveckling av programmet.

8.6 Införa domänkunskap

Vid exempelvis ekvationslösningar, transformationer av system samt vid lösningar av partiella differentialekvationener är det en fördel att kunna förenkla uttryck så långt som möjligt. Genom att införa domänkunskaper som exempelvis att radien alltid är större än noll ($x > 0$) kan man förenkla uttryck som $\sqrt{x^2} - x$.

8.7 Förbättringar av funktionen FrobSolve

Funktionen $FrobSolve$ som diskuterades i avsnitt 7.1 innehöll två steg som kunde vara problematiska, beräkning av flöden samt invertering av en avbildning. Inverteringen skulle kunna förbättras genom att man tittar på ett lokalt område runt en punkt istället för att inverteringen ska gälla globalt.

8.8 Slutsatser

Mathematica har visat sig i denna applikation vara ett bra hjälpmedel. Även om programmet inte alltid klarar av att automatiskt skapa transformationerna så ger programmet ett bra stöd, vilket har visat sig genom exempel. De flesta stegen vid skapandet av transformationerna samt transformeringen gick att göra en enkel och rättfram implementation av, men i vissa steg har problematik identifierats och förslag på förbättringar getts. Om man ska förbättra programmet skulle jag tycka att skapandet av Ψ_1 är den viktigaste förbättringen då detta steg är det mest problematiska. Något som skulle vara enkelt att implementera och samtidigt användbart är att särskilja linjära och olinjära system, eftersom transformationerna alltid går att skapa för linjära system.

Litteraturförteckning

[1] De Persis, Claudio & Isidori, Alberto: *G*eometric Approach to Nonlinear Fault Detection and Isolation, *IEEE Transactions om automatic control, VOL. 46, NO. 6* (2001).

[2] Isidori, Alberto: *Nonlinear Control Systems*, 3rd ed. London, U.K.: Spring-Verlag (1995).

[3] Glad, Torkel & Ljung, Lennart: *Reglerteknik. Grundläggande teori*, Lund: Studentlilltteratur (1989).

[4] Nyberg, Mattias & Frisk, Erik: *Model Based Diagnos of Technical Processes* (2003).

Bilaga A

Variabler och funktioner

A.1 Variabler

xvar	vektor	tillstånd
yvar	vektor	utsignal
g	matris	g = $\{f, g_1, g_2\}$
p	matris	störning
l	matris	fel
h	vektor	$y = h(x)$
x0	vektor	begynnelsepunkt

A.2 Funktioner

ΔSolve[p, h, g, xvar]
FaultDetectableQ[Δ, l]
CreateExtension[Δ, x0, xvar]
Φ1Solve[Δ, F, x0, xvar]
Ψ1Solve[Δ, h, x0, xvar, yvar]
ΨandH2Solve[Ψ_1, yvar]
ΦSolve[Φ_1, H2, h, xvar]

ComputeTransformations[xvar, yvar, g, p, l, h, x0]
CheckTransformations[resultComputeTransformations]

FaultDetectionTransformation[xvar, yvar, g,p,l,h,x0]
TransformSystem[Ψ, Φ, DΦ_1,DH2,xvar, yvar, g,p,l,h,x0]

FDTQuotientSystem[resultFDT]
TSQuotientSystem[DΨ_1, DH2, DΦ_1, DΦ_3,resultTS]
FDTPrintQuotientSystem[resultFDT, xvar, g, l, p]
TSPrintQuotientSystem[DΨ_1, DH2, DΦ_1, DΦ_3,resultTS, g, l, p]

Bilaga B

Definitioner och räkneoperationer

Följande är en lista över de definitioner och räkneoperationer som används i dokumentet, tillsammans med en kort förklarande text. En del av dessa definitioner kan vara komplicerade samt svåra att förstå, där finns hänvisningar för närmare studier.

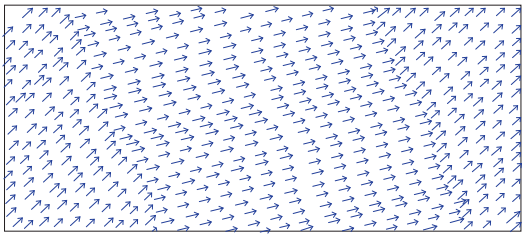
B.1 Definitioner

- **Mjuk funktion**

En funktion som är oändligt deriverbar.

- **Vektorfält**

Ett vektorfält $f(x) \in \mathbb{R}^n$ associerar en vektor (illustreras med en pil) i varje punkt av dess definitionsområde. I figur B.1 kan ramen ses som definitionsområdet och pilarna som vektorer.



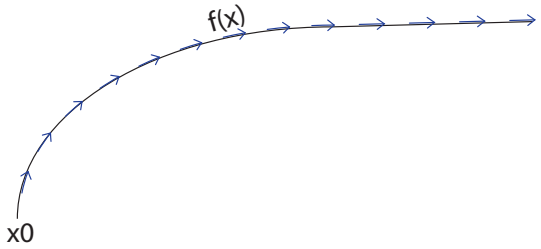
Figur B.1: *Ett vektorfält $f(x) \in \mathbb{R}^n$ associerar en vektor i varje punkt av dess definitionsområde.*

Exempel:

$$f(x) = \begin{pmatrix} 2x_3 \\ -1 \\ 0 \end{pmatrix}$$

- **Flödet**

Flödet betecknas med $\Phi_t^f(x)$, en mjuk funktion av t och x, med egenskapen att $x(t) = \Phi_t^f(x^o)$ löser den partiella differentialekvationen $\dot{x} = f(x)$ med initialvillkoret $x(0) = x^0$. Geometriskt följer flödet en bana och vektorfältet $f(x)$ (pilar i figur B.2) är ständigt tangent till flödets riktning. Flödets bana börjar i punkten $x(0) = x_0$, se figur B.2.



Figur B.2: *Flödet är en bana som ständigt tangerar vektorfältet* $f(x)$ *och börjar i punkten* $x(0) = x_0$.

- **Distribution [2]**

En distribution är ett vektorrum som spänns upp av ett antal vektorer$f_1(x), \ldots, f_d(x)$. Vektorerna som spänner upp vektorrummet är kolonnvektorerna i matrisen som representerar distributionen.

Beteckning:
$\Delta(x) = span\{f_1(x), \cdots, f_d(x)\}$.

- **Codistribution [2]**

Samma sak som distribution fast vektorerna som spänner upp vektorrymden är radvektorer istället för kolonnvektorer.

- **Nollrum [3]**

Nollrummet för matrisen A är mängden av lösningar till den homogena ekvationen $AX = 0$.

Beteckning:
$Ker\{\}$.

- **Annihilatorn [2]**

Annihilatorn till en distribution är en codistribution och fås genom att ta nollrummet till distributionen. Omvänt gäller att annihilatorn till en codistribution är en distribution och fås genom att ta nollrummet till codistributionen. Annihilatorn betecknas med $^{\perp}$.

- **Icke-singulär distribution [2]**

En distribution anses vara icke-singulär om dimensionen av distributionen är konstant över hela defenitionsmängden, dvs. $dim(distributionen) = d$.

- **Singulär distribution [2]**

En singulär distribution har en varierande dimension över definitionsmängden.

- **Reguljär punkt [2]**

En punkt anses vara reguljär om punkten samt en omgivning runt denna är icke-singulär.

- **Diffeomorfism [2]**

En lokal respektive global diffeomorfism uppfyller villkoren att ha en transformation där både Φ och Φ^{-1} är mjuka mappningar och har kontinuerliga partiella derivator samt att $\Phi(\Phi^{-1}(x)) = x$ gäller lokalt respektive globalt.

- **Involutiv [2]**

En distribution är involutiv om Lie-produkten (se avsnitt B2) mellan två vektorfält, vilka som helst, som tillhör distributionen också tillhör distributionen.

Beteckning:
$\tau_1 \in \Delta, \tau_2 \in \Delta \Longrightarrow [\tau_1, \tau_2] \in \Delta$.

Exempel:

$$U = \{x \in \mathbb{R}^3 : x_1^2 + x_3^2 \neq 0\}$$

$$\Delta = span\{f, g\}$$

$$f(x) = \begin{pmatrix} 2x_3 \\ -1 \\ 0 \end{pmatrix} \quad g(x) = \begin{pmatrix} -x_1 \\ -2x_2 \\ x_3 \end{pmatrix} \quad [f,g](x) = \begin{pmatrix} -4x_3 \\ 2 \\ 0 \end{pmatrix}$$

$$(f, g, [f, g])(x) = \begin{pmatrix} 2x_3 & -x_1 & -4x_3 \\ -1 & -2x_2 & 2 \\ 0 & x_3 & 0 \end{pmatrix}$$

$(f, g, [f, g])(x)$ har rangen 2 ty $[f, g] = -2f$. Därför är distributionen Δ

involutiv.

- **Involutiva höljet [2]**

För en distribution som inte är involutiv existerar det ett involutivt hölje vilket innebär den minsta större distributionen som är involutiv. Ett sätt att skapa involutiva höljet på är att ta Lie-produkten mellan alla vektorer i distributionen och lägga till de vektorer som inte tillhör distributionen.

Exempel:

$$U = \{x \in \mathbb{R}^3\}$$

$$\Delta = span\{f, g\}$$

$$f(x) = \begin{pmatrix} 2x_2 \\ 1 \\ 0 \end{pmatrix} \quad g(x) = \begin{pmatrix} 1 \\ 0 \\ x_2 \end{pmatrix} \quad [f,g](x) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$(f, g, [f, g])(x) = \begin{pmatrix} 2x_2 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & x_2 & 1 \end{pmatrix} = \mathbb{R}^3$$

$(f, g, [f, g])(x)$ har rangen 3 och därför är distributionen Δ icke-involutiv. $(f, g, [f, g])(x)$ är involutiva höljet till Δ.

- **Invarianta distributioner [2]**

En distribution anses vara invariant under ett vektorfält f om Lie-produkten mellan f och varje vektorfält i distributionen blir ett vektorfält i distributionen.

Exempel:
Om f är invariant under Δ så gäller att $[f, \Delta] \in \Delta$.

B.2 Räkneoperationer

- **Jakobianen [2]**

$$\frac{\partial \Phi}{\partial x} = \begin{pmatrix} \frac{\partial \Phi_1}{\partial x_1} & \cdots & \frac{\partial \Phi_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Phi_n}{\partial x_1} & \cdots & \frac{\partial \Phi_n}{\partial x_n} \end{pmatrix}$$

Exempel:

$$\Phi(x_1, x_2) = \begin{pmatrix} x_1 + x_2 \\ sin\ x_2 \end{pmatrix}$$

$$\frac{\partial \Phi}{\partial x} = \begin{pmatrix} 1 & 1 \\ 0 & cos\ x_2 \end{pmatrix}$$

- **Lie-produkt [2]**

$$[f,g](x) = \frac{\partial g}{\partial x} f(x) - \frac{\partial f}{\partial x} g(x)$$

Exempel:

$$f(x) = \begin{pmatrix} 2x_3 \\ -1 \\ 0 \end{pmatrix} \quad g(x) = \begin{pmatrix} -x_1 \\ -2x_2 \\ x_3 \end{pmatrix}$$

$$[f,g](x) = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2x_3 \\ -1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} -x_1 \\ -2x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -4x_3 \\ 2 \\ 0 \end{pmatrix}$$

- **Differential / Gradient [2]**

$$d\lambda = (\frac{\partial \lambda}{\partial x_1} \frac{\partial \lambda}{\partial x_2} \cdots \frac{\partial \lambda}{\partial x_n})$$

Exempel:

$$\lambda = x_1\ x_2\ Sin(x_3)$$

$$d\lambda = (x_2\ Sin(x_3),\ x_1\ Sin(x_3),\ x_1\ x_2\ Cos(x_3))$$

- **Lie-derivata [2]**

$$L_f\lambda(x) = \sum_{i=1}^{n} \frac{\partial \lambda}{\partial x_i} f_i(x)$$

Exempel:

$$\lambda = x_1\, x_2\, Sin(x_3)$$
$$f = \begin{pmatrix} x_1 \\ x_1 x_2 \\ 2 \end{pmatrix}$$

$$L_f\lambda(x) = x_1\, x_2\, Sin(x_3) + x_1^2\, x_2\, Sin(x_3) + 2\, x_1\, x_2\, Cos(x_3)$$

Bilaga C

Implementerad kod

Den här bilagan innehåller den kod som utvecklats under examensarbetet. Den består av de funktioner som finns med i figur 6.1, samt underfunktioner till dessa. Förutom Mathematicas egna funktioner använder sig programmet av funktioner skapade på Fordonssystem vid Linköpings universitet. Vid intresse av dessa funktioner kontakta avdelningen.

FaultDetectionTransformation

Anna Önnegern

Version:2004−08−10

Funktioner

■ Skapa transformationerna

■ **Konstruera Δ; ΔSolve[p, h, g, xvar][[1]], ΔSolve[p,h,g,xvar][[2]]**

```
ΔSolve[p_, h_, g_, xvar_] :=
  Module[{i, Δ, ΔsingularQ, Δtemp, intersect},
   Δ = {p[[1]]};
   For[i = 2, i ≤ Length[p], i++,
    DM = DistMemberQ[p[[i]], Δ];
    If[DM === True, , Δ = Join[Δ, {p[[i]]}]]];

   {Δ, ΔsingularQ} = CreateDistributionΔ[g, h, Δ, x0, xvar];
   Return[{Δ, ΔsingularQ}]];
```

■ **Kontrollera att inte l ingår i Δ FaultDetectableQ[Δ, l]**

```
FaultDetectableQ[Δ_, l_] := Module[{i, DM, temp},
  temp = False;

  For[i = 1, i ≤ Length[l], i++,
   DM = DistMemberQ[l[[1]], Δ];
   If[DM === True, temp = True,]];
  Return[temp]]
```

- **Fyll ut Δ;** **CreateExtension[Δ,x0,xvar]**

```
CreateExtension[Δ_, x0_, xvar_] := Module[{i, F, IM},
  F = Δ;
  IM = IdentityMatrix[Dimensions[F][[2]]];

  Do[
   For[i = 1, i ≤ Length[IM], i++,
    If[MatrixRank[Join[F, {IM[[i]]}] /. Thread[xvar → x0]] ==
      Length[F], , F = Join[F, {IM[[i]]}]];
    If[Dimensions[F][[2]] - Dimensions[F][[1]] == 0,
     Break[]]];
  Return[
   F]]
```

- **Konstruera Φ1;** **Φ1Solve[Δ, F,x0,xvar]**

```
Φ1Solve[Δ_, F_, x0_ , xvar_] := Module[{DΔ, Φ1},
  DΔ = Dimensions[Δ][[1]];

  Φ1 = FrobSolve[F, x0, xvar, DΔ];
  Return[Φ1]]
```

- **Konstruera Ψ1;** **Ψ1Solve[Δ, h, x0, xvar, yvar]**

```
Ψ1Solve[Δ_, h_, x0_, xvar_, yvar_] :=
 Module[{intersectF, D, F, Ψ1temp, Ψ1},
  intersectF = IntersectF[h, xvar, Δ];
  D = Dimensions[NullSpace[intersectF]][[1]];
  F = CreateExtension[intersectF, x0, xvar];

  Ψ1temp = FrobSolve[F, x0, xvar, D];
  Ψ1 = CreateΨ1[h, Ψ1temp, yvar, xvar];
  Return[Ψ1];]
```

- **Konstruera Ψ,H2; ΨandH2Solve[Ψ1,yvar][[1]], ΨandH2Solve[Ψ1,yvar][[2]]**

```
ΨandH2Solve[Ψ1_, yvar_] :=
  Module[{i, IM, DΨ1, Dh, DH2, H2, j, Ψ, Ψ1temp, MR2, MR, DΨ},
   IM = IdentityMatrix[Length[yvar]];
   DΨ1 = Length[Ψ1];
   Dh = Length[h]; (*ska ju transformera h,
    måste därför ha samma dimension*)
   DH2 = Dh - DΨ1;

   H2 = Array[0 &, {DH2, Dh}];  (*Skapar en tom matris H2*)
   j = 1;
   Ψ = Ψ1;

   Do[
    For[i = 1, i <= Length[yvar], i++,
     Ψ1temp = Join[Ψ, {yvar[[i]]}];
     MR2 = MatrixRank[Diff[Ψ1temp, yvar]];
     MR = MatrixRank[Diff[Ψ, yvar]];
      If[MR2 == MR, , H2[[j]] = IM[[i]]; Ψ = Ψ1temp; j = j + 1 ];
     DΨ = Lenght[Ψ];
     If[Dh == DΨ, Break[]]]];
    Return[{Ψ, H2}]];
```

- **Konstruera Φ; ΦSolve[Φ1, H2, h, xvar]**

```
ΦSolve[Φ1_, H2_, h_, xvar_] := Module[
  {i, IM, Φtemp, DΦtemp, Dxvar, DΦ3, Φtemp2, MR2, MR, DΦ},
  IM = IdentityMatrix[Length[xvar]];
  Φtemp = Join[Φ1, H2.h];
  DΦtemp = Length[Φtemp];
  Dxvar = Length[xvar];
  DΦ3 = Dxvar - DΦtemp;
  (*eftersom tillstånden ska transformeras*)

  For[i = 1, i ≤ Length[IM], i++,
   Do[
    Φtemp2 = Join[Φtemp, {xvar[[i]]}];
    MR2 = MatrixRank[Diff[Φtemp2, xvar]];
    MR = MatrixRank[Diff[Φtemp, xvar]];
     If[MR2 == MR, , Φtemp = Φtemp2]];
   DΦ = Length[Φtemp];
   If[DΦ = Dxvar, Break[]]];
  Return[Φtemp]]
```

- **Räkna fram Φ och Ψ. ComputeTransformations[xvar, yvar, g,p,l,h,x0]**

```
ComputeTransformations[xvar_,
   yvar_, g_, p_, l_, h_, x0_] := Module[
   {Δ, ΔsingularQ, FaultDetectedQ, F, Φ1, Ψ1, Ψ, H2, Φ},
   If[p == Array[0 &, Dimensions[p]], Return[{Δ, ΔsingularQ,
      FaultDetectedQ, F, Φ1, Ψ1, Ψ, H2, Φ}],];
   {Δ, ΔsingularQ} = ΔSolve[p, h, g, xvar];
   FaultDetectedQ = FaultDetectableQ[Δ, l];
       If[FaultDetectedQ === True, Return[{Δ, ΔsingularQ,
      FaultDetectedQ, F, Φ1, Ψ1, Ψ, H2, Φ}],];
   F = CreateExtension[Δ, x0, xvar];
   Φ1 = Φ1Solve[Δ, F, x0, xvar];
   Ψ1 = Ψ1Solve[Δ, h, x0, xvar, yvar];
   {Ψ, H2} = ΨandH2Solve[Ψ1, yvar];
   Φ = ΦSolve[Φ1, H2, h, xvar];
   Return[
    {Δ, ΔsingularQ, FaultDetectedQ, F, Φ1, Ψ1, Ψ, H2, Φ}]];
```

- **Kontrollera om Φ och Ψ är rimliga**
  CheckTransformations[resultComputeTransformations]

```
CheckTransformations[
   resultComputeTransformations] := Module[
   {temp, Δ, ΔsingularQ, FaultDetectedQ, F, Φ1, Φ1test,
    ZeroMatrix, Ψ1, Ψ1temp, HL, VL, Ψ1test, Ψ, H2, Φ},
   temp = 1;

   {Δ, ΔsingularQ, FaultDetectedQ, F, Φ1, Ψ1, Ψ, H2, Φ} =
    resultComputeTransformations;

   If[Δ == {}, Print["Δ is a enpty matrix."]; temp = 2,];
   If[ΔsingularQ === True,
    Print["Δ is singular."]; temp = 2,];

      If[FaultDetectedQ === True,
    Print["It is impossible to create a distribution
      where l dosn't belong to Δ."]; temp = 2,];

   Φ1test = Δ.Transpose[JacobianMatrix[Φ1, xvar]];
   ZeroMatrix = Array[0 &, Dimensions[Φ1test]];
   If[Φ1test == ZeroMatrix, , Print["dΦ1 * Δ = ", Φ1test,
     ". This should be", ZeroMatrix]; temp = 2];

   Ψ1temp = Ψ1 /. Thread[yvar → h];
   HL = Transpose[Diff[Ψ1temp, xvar]];
   VL = Transpose[Union[Δ, NullSpace[Diff[h, xvar]]]];
   temp2 = True;
   For[i = 1, i ≤ Length[HL],
    i++, temp1 = DistMemberQ[HL[[i]], VL];
    If[temp1 === True, , temp2 = False]];
   If[temp2 === True, , Print[Fel, fel, fel]; temp = 2];

   If[Dimensions[Ψ] == Dimensions[yvar], ,
    Print["Ψ has wrong dimension"]; temp = 2];

   If[Dimensions[Φ] == Dimensions[xvar], ,
    Print["Φ has wrong dimension"]; temp = 2];

   If[temp == 1, Print["Seams correct."],]];
```

- **Transformera systemet**

- **Skapa transformationerna samt transformera systemet FaultDetectionTransformation[xvar, yvar, g,p,l,h,x0]**

```
FaultDetectionTransformation[
   xvar_, yvar_, g_, p_, l_, h_, x0_] :=
  Module[{Δ, ΔsingularQ, FaultDetectedQ, F, Φ1, Ψ1, Ψ,
    H2, Φ, ftemp, gtemp, htemp, output},
   {Δ, ΔsingularQ, FaultDetectedQ, F, Φ1, Ψ1, Ψ, H2, Φ} =
    ComputeTransformations[xvar, yvar, g, p, l, h, x0];
   DΦ1 = Length[Φ1];
   DH2 = Length[H2];
   {ftemp, gtemp, htemp, output} = TransformSystem[
     Ψ, Φ, DΦ1, DH2, xvar, yvar, g, p, l, h, x0];
   Return[{Δ, ΔsingularQ, FaultDetectedQ, F, Φ1,
     Ψ1, Ψ, H2, Φ, ftemp, gtemp, htemp, output}]];
```

- **Transformera systemet TransformSystem[Ψ, Φ, DΦ1,DH2,xvar, yvar, g,p,l,h,x0]**

```
TransformSystem[Ψ_, Φ_, DΦ1_, DH2_, xvar_, yvar_, g_, p_, l_,
   h_, x0_] := Module[{ftemp, gtemp, htemp, output},
   {ftemp, gtemp, htemp} = TransformState[
     Φ, DΦ1, DH2, g, p, l, h, xvar];
   output = TransformOutput[Ψ, yvar, htemp];
   Return[{ftemp, gtemp, htemp, output}]];
```

■ Skapa kvotsystemet

■ Skapa kvotsystemet FDTQuotientSystem[resultFDT]

```
FDTQuotientSystem[result_] :=
 Module[{Ψ1, H2, Φ1, Φ, ggllpp, ff, hh, DΨ1,
   DH2, DΦ1, DΦ3, yyvar, ggllpp1, ff1, hh1},
  Ψ1 = result[[6]];
  H2 = result[[8]];
  Φ1 = result[[5]];
  Φ = result[[9]];
  ggllpp = result[[11]];
  ff = result[[10]];
  hh = result[[13]];

  DΨ1 = Length[Ψ1];
  DH2 = Length[H2];
  DΦ1 = Length[Φ1];
  DΦ3 = Length[Φ] - DH2 - DΦ1;

  yyvar = Createyyvar[DΨ1, DH2];

  (*Plockar ut den del som ska vara med i kvotsystemet*)
  ggllpp1 = QuotientSystemglp[ggllpp, DΦ1];
  ff1 = QuotientSystemf[ff, hh, DΦ1, DΨ1, DH2, yyvar];
  hh1 = QuotientSystemh[hh, DΨ1];

  Return[{ff1, ggllpp1, hh1}]]
```

■ Skapa kvotsystemet TSQuotientSystem[DΨ1, DH2, DΦ1, DΦ3, resTS]

```
TSQuotientSystem[DΨ1_, DH2_, DΦ1_, DΦ3_, res_] :=
 Module[{ggllpp, ff, hh, yyvar, ggllpp1, ff1, hh1},
  ggllpp = res[[2]];
  ff = res[[1]];
  hh = res[[4]];

  yyvar = Createyyvar[DΨ1, DH2];

  (*Plockar ut den del som ska vara med i kvotsystemet*)
  ggllpp1 = QuotientSystemglp[ggllpp, DΦ1];
  ff1 = QuotientSystemf[ff, hh, DΦ1, DΨ1, DH2, yyvar];
  hh1 = QuotientSystemh[hh, DΨ1];

  Return[{ff1, ggllpp1, hh1}]]
```

- **Skriv ut kvotsystemet FDTPrintQuotientSystem[result, xvar, g, l, p]**

```
FDTPrintQuotientSystem[result_, g_, l_, p_] :=
 Module[{i, Ψ1, H2, Φ1, DΨ1, DΦ1, DH2, yyvar,
   zvar, ggllpp1, ff1, hh1, Dg, Tg, Dl, Tl, Dp, Tp},
  Ψ1 = result[[6]];
  H2 = result[[8]];
  Φ1 = result[[5]];
  DΨ1 = Length[Ψ1];
  DH2 = Length[H2];
  DΦ1 = Length[Φ1];

  {ff1, ggllpp1, hh1} = FDTQuotientSystem[result];

  yyvar = Createyyvar[DΨ1, DH2];
  zvar = Createzvar[DΦ1, DH2, DΦ3];


  Dg = Length[g] - 1;
  Rg = Range[1, Dg];
  Dl = Length[l];
  Rl = Range[1 + Dg, Dg + Dl];
  Dp = Length[p];
  Rp = Range[1 + Dg + Dl, Dg + Dl + Dp];

  For[i = 1, i ≤ DΦ1, i++,
   Print["d", zvar[[i]], " = ", ff1[[1]][[i]], " + ",
    ggllpp1[[i]][[Rg]], u, " + ", ggllpp1[[i]][[Rl]],
    "m", " + ", ggllpp1[[i]][[Rp]], "w"]];
  For[i = 1, i ≤ DΨ1, i++,
   Print[yyvar[[i]], " = ", hh1[[1]][[i]]]]]
```

- **Skriv ut kvotsystemet TSPrintQuotientSystem[DΨ1, DH2, DΦ1, DΦ3, res, g, l, p]**

```
TSPrintQuotientSystem[DΨ1_, DH2_, DΦ1_, DΦ3_, res_,
  g_, l_, p_] := Module[{i, ggllpp, ff, hh, yyvar,
   zvar, ggllpp1, ff1, hh1, Dg, Tg, Dl, Tl, Dp, Tp},
  zvar = Createzvar[DΦ1, DH2, DΦ3];
  yyvar = Createyyvar[DΨ1, DH2];

  {ff1, ggllpp1, hh1} =
   TSQuotientSystem[DΨ1, DH2, DΦ1, DΦ3, res];

  Dg = Length[g] - 1;
  Rg = Range[1, Dg];
  Dl = Length[l];
  Rl = Range[1 + Dg, Dg + Dl];
  Dp = Length[p];
  Rp = Range[1 + Dg + Dl, Dg + Dl + Dp];

  For[i = 1, i ≤ DΦ1, i++,
   Print["d", zvar[[i]], " = ", ff1[[1]][[i]], " + ",
    ggllpp1[[i]][[Rg]], u, " + ", ggllpp1[[i]][[Rl]],
    "m", " + ", ggllpp1[[i]][[Rp]], "w"]];
  For[i = 1, i ≤ DΨ1, i++,
   Print[yyvar[[i]], " = ", hh1[[1]][[i]]]]]
```

Underfunktioner

- **ΔSolve**

- **CreateDistributionΔ[g,h,Δ,x0,xvar]** **Stanna iterationen när Δ_k+1 = Δ_k**

```
CreateDistributionΔ[g_, h_, Δ_, x0_, xvar_] :=
 Module[{n, i, k, Δtemp, Lp, DM, ΔsingularQ},
  Δtemp = Δ;

  For[n = 1, n ≤ Length[Δ], n = n + 1, (*itterationen*)
   Do[
    Δtemptest = Δtemp;
    intersect = Intersect[h, Δtemp, xvar];

    For[i = 1, i ≤ Length[g], i++, (*summationen*)
      For[k = 1,
       k ≤ Length[intersect], k++, (*Lieprodukten*)
       Lp = LieProduct[g[[i]], intersect[[k]], xvar];
       DM = DistMemberQ[Lp, Δtemp];
       If[DM == False, Δtemp =
         Union[InvolutiveClosure[Δtemp, xvar], {Lp}],]]]
     If[Δtemptest == Δtemp, Break[] ]]];

  If[MatrixRank[Δtemp /. Thread[xvar → x0]] == Length[Δtemp],
   ΔsingularQ = False , ΔsingularQ = True];
  Return [{Δtemp, ΔsingularQ}]]
```

- **Intersect[h, Δ, xvar]**

```
Intersect[h_, Δ_, xvar_] := (
   NullSpace[Union[Diff[h, xvar],
     NullSpace[InvolutiveClosure[Δ, xvar]]]]);
```

- **InvolutiveClosure[F, xvar]**

```
InvolutiveClosure[F_, xvar_] := Module[{i, j, f, Lp, DM},
  f = F;

  For[j = 1, j < Length[f], j++,
   For[i = 2, i <= Length[f], i++,
    Lp = LieProduct[f[[i]], f[[j]], xvar];
    DM = DistMemberQ[Lp, f];
    If[DM == True, , f = Join[f, {Lp}]]]];
  Return[f]]
```

- **Ψ1Solve**

- **IntersectF[h,xvar,Δ]**

```
IntersectF[h_, xvar_, Δ_] :=
  (Union[Δ, NullSpace[Diff[h, xvar]]]);
```

- **CreateΨ1[h, Ψ1temp, yvar, xvar]**

```
CreateΨ1[h_, Ψ1temp_, yvar_, xvar_] :=
 Module[{Dh, MR, DDh, inv, Ψ1},
  var = Intersection[Variables[Ψ1temp], xvar];
  Dh = Diff[h, var];
  MR = MatrixRank[Dh];
  DDh = Dimensions[Dh][[2]];
  If[MR == DDh, ,
   Print["This program can not calculate Ψ1. The
      method that is implemented in this program
      does not cover all possible solutions. Either
      there does not exist any solutions for Ψ1
      or it can be found by outher methods.  "]];
  inv = Solve[{yvar == h}, var][[1]];
  Ψ1 = Ψ1temp /. inv;
  Return[Ψ1]]
```

- **Transformera kvotsystemet**

- **Variabeltransformation av tillstånden**
  TransformState[Φ,DΦ1,DH2,g,p,l,h,xvar]

```
TransformState[Φ_, DΦ1_, DH2_, g_, p_, l_, h_, xvar_] :=
  Module[{DΦ3, zvar, Φtemp, f, gtemp, newsys},
   DΦ3 = Length[Φ] - DH2 - DΦ1;
   zvar = Createzvar[DΦ1, DH2, DΦ3];
   Φtemp = Thread[zvar → Φ];
   f = g[[1]];
   gtemp = Join[Delete[g, 1], l, p];
   newsys = ChangeCoordinates[f, gtemp, h, Φtemp, xvar];
   Return[newsys]];
```

- **Variabeltransformation av utsignalen**
  TransformOutput[Ψ,yvar,htemp]

```
TransformOutput[Ψ_, yvar_, htemp_] :=
  Module[{hmap, h2},
   hmap = Thread[yvar → htemp];
   h2 = Ψ /. hmap;
   Return[h2]];
```

- **Createzvar[DΦ1,DH2,DΦ3]**

```
Createzvar[DΦ1_, DH2_, DΦ3_] :=
  Module[{i, z, z1, z2, z3, zvar},

   z1 = {0};
   If[DΦ1 == 1, z1 = {"z1"}, z = CreateVariables[DΦ1, 10, "z"];
    For[i = 1, i <= DΦ1, i++, z1 = Join[z1, {z[[1]][[i]]}]];
    z1 = Delete[z1, 1]];

   z2 = {0};
   If[DH2 == 1, z2 = {"z2"}, z = CreateVariables[DH2, 20, "z"];
    For[i = 1, i <= DH2, i++, z2 = Join[z2, {z[[1]][[i]]}]];
    z2 = Delete[z2, 1]];

   z3 = {0};
   If[DΦ3 == 1, z3 = {"z3"}, z = CreateVariables[DΦ3, 30, "z"];
    For[i = 1, i <= DΦ3, i++, z3 = Join[z3, {z[[1]][[i]]}]];
    z3 = Delete[z3, 1]];

   zvar = Join[z1, z2, z3];
   zvar = ToExpression[zvar];
   Return[zvar]];
```

- **CreateVariables[D, a,"z"]**

```
CreateVariables[D_, a_, z_] := Module[{i, zTS, T, TS, zz},
   zTS = Table[i + a, {i, D}] ;
   (*skapar en tabell för att kunna skriva över värdena*)
   T = Table[i + a, {i, D}];
   For[i = 1, i <= D, i++, TS = ToString[T[[i]]];
    zTS[[i]] = z <> TS; zz = zTS // MatrixForm];
   Return[zz]];
```

■ Skapa kvotsystemet

- **Kvotsystem glp** `QuotientSystemglp[ggllpp, DΦ1]`

```
QuotientSystemglp[glp_, DΦ1_] := Module[{ggllpp},
   ggllpp = Transpose[glp][[Range[1, DΦ1]]];
   Return[ggllpp]];
```

- **Kvotsystem f**
  `QuotientSystemf[ff, hh, DΦ1, DΨ1, DH2, yyvar]`

```
QuotientSystemf[ff_, hh_, DΦ1_, DΨ1_, DH2_, yyvar_] :=
  Module[{Rhh, r, f},
   Rhh = Range[DΨ1 + 1, DΨ1 + DH2];
   r = Thread[hh[[Rhh]] → yyvar[[Rhh]]];
   f = {ff[[Range[1, DΦ1]]] /. r};
   Return[f]];
```

- **Kvotsystem h** `QuotientSystemh[hh, DΦ1]`

```
QuotientSystemh[hh_, DΦ1_] := Module[{ h},
   h = {hh[[Range[1, DΦ1]]]};
   Return[h]];
```

■ Createyyvar[DΨ1,DH2]

```
Createyyvar[DDΨ1_, DDH2_] :=
 Module[{DΨ1, DH2, yy1temp, yy2temp, yyvartemp, yyvar},
  DΨ1 = DDΨ1;
  DH2 = DDH2;

  If[DΨ1 == 1, yy1temp = {"yy1"},
   yy1temp = {0}; yy = CreateVariables[DΨ1, 10, "yy"];
   For[i = 1, i <= DΨ1, i++,
    yy1temp = Join[yy1temp, {yy[[1]][[i]]}]];
   yy1temp = Delete[yy1temp, 1]];

  If[DH2 == 1, yy2temp = {"yy2"}, yy2temp = {0};
   yy = CreateVariables[DH2, 20, "yy"]; For[i = 1, i <= DH2,
    i++, yy2temp = Join[yy2temp, {yy[[1]][[i]]}]];
   yy2temp = Delete[yy2temp, 1]];

  yyvartemp = Join[yy1temp, yy2temp];
  yyvar = ToExpression[yyvartemp];
  Return[yyvar]]
```

Copyright

Svenska

English