Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Evaluation of Differential Algebraic Elimination Methods for Deriving Consistency Relations from an Engine Model

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan i Linköping
av

Rikard Falkeborn

LITH-ISY-EX--06/3899--SE

Linköping 2006

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

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Evaluation of Differential Algebraic Elimination Methods for Deriving Consistency Relations from an Engine Model

Examensarbete utfört i Fordonssystem
vid Tekniska högskolan i Linköping
av

Rikard Falkeborn

LITH-ISY-EX--06/3899--SE

Handledare: **Dr Mattias Nyberg**
Scania CV AB
Dr Mattias Krysander
ISY, Linköpings universitet

Examinator: **Dr Jan Åslund**
ISY, Linköpings universitet

Linköping, 11 December, 2006

Avdelning, Institution
Division, Department

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

Datum
Date

2006-12-11

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English
☐ _____

Rapporttyp
Report category

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

ISBN
—

ISRN
LITH-ISY-EX--06/3899--SE

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

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

Titel
Title

Utvärdering av differential-algebraiska elimineringsmetoder för att beräkna konsistensrelationer från en dieselmotor

Evaluation of Differential Algebraic Elimination Methods for Deriving Consistency Relations from an Engine Model

Författare
Author

Rikard Falkeborn

Sammanfattning
Abstract

New emission legislations introduced in the European Union and the U.S. have made truck manufacturers face stricter requirements for low emissions and on-board diagnostic systems. The on-board diagnostic system typically consists of several tests that are run when the truck is driving. One way to construct such tests is to use so called consistency relations. A consistency relation is a relation with known variables that in the fault free case always holds. Calculation of a consistency relation typically involves eliminating unknown variables from a set of equations.

To eliminate variables from a differential polynomial system, methods from differential algebra can be used. In this thesis, the purely algebraic Gröbner basis algorithm and the differential Rosenfeld-Gröbner algorithm implemented in the MAPLE package `diffalg` have been compared and evaluated. The conclusion drawn is that there are no significant differences between the methods. However, since using Gröbner basis requires differentiations to be made in advance, the recommendation is to use the Rosenfeld-Gröbner algorithm.

Further, attempts to calculate consistency relations using the Rosenfeld-Gröbner algorithm have been made to a real application, a model of a Scania diesel engine. These attempts did not yield any successful results. It was only possible to calculate one consistency relation. This can be explained by the high complexity of the model.

Nyckelord
Keywords model based diagnosis, Gröbner basis, diffalg, consistency relation, MSO set

Abstract

New emission legislations introduced in the European Union and the U.S. have made truck manufacturers face stricter requirements for low emissions and on-board diagnostic systems. The on-board diagnostic system typically consists of several tests that are run when the truck is driving. One way to construct such tests is to use so called consistency relations. A consistency relation is a relation with known variables that in the fault free case always holds. Calculation of a consistency relation typically involves eliminating unknown variables from a set of equations.

To eliminate variables from a differential polynomial system, methods from differential algebra can be used. In this thesis, the purely algebraic Gröbner basis algorithm and the differential Rosenfeld-Gröbner algorithm implemented in the MAPLE package `diffalg` have been compared and evaluated. The conclusion drawn is that there are no significant differences between the methods. However, since using Gröbner basis requires differentiations to be made in advance, the recommendation is to use the Rosenfeld-Gröbner algorithm.

Further, attempts to calculate consistency relations using the Rosenfeld-Gröbner algorithm have been made to a real application, a model of a Scania diesel engine. These attempts did not yield any successful results. It was only possible to calculate one consistency relation. This can be explained by the high complexity of the model.

Acknowledgments

I would like to express my gratitude to a number of people who have made this thesis possible.

First, my excellent supervisors, Mattias Nyberg and Mattias Krysander, and my examiner, Jan Åslund, for help and interesting discussions about the thesis. Jens Molin and Joakim Hansen whom I shared a room with at Scania, for interesting discussions about everything from MATLAB to football. I also want to thank the other master thesis workers, Carl Svärd, Henrik Wassén, Klas Håkansson and Mikael Johansson for many nice coffee breaks.

Finally I would like to thank Tina Lundkvist for always being there for me when I need it.

Rikard Falkeborn
Södertälje, December 2006

Contents

Chapter 1

Introduction

This master's thesis was performed at Scania CV AB in Södertälje. Scania is a worldwide manufacturer of heavy duty trucks, buses and engines for marine and industrial use. The work was carried out at the diagnosis group responsible for the on-board diagnosis (OBD).

1.1 Background

New emission legislations introduced in the European Union and the U.S. have made truck manufacturers face stricter requirements for low emissions and *on-board diagnostic* systems. Among other things, the newly introduced legislations in the European Union states that faults on the truck affecting the emissions over a certain level must be detected and the driver must be alerted. In the future, these legislations will become even stricter and therefore there is a need to improve the OBD system further.

Faults can for example be faulty sensors measuring the wrong value or leakage in hoses that will give the engine unexpected properties and thus increase the emissions. Other reasons to incorporate fault diagnosis is to make the driver aware of faults that can damage the engine so the truck can be taken to a repair shop in time. Another benefit is if the OBD system can isolate the fault and then inform the mechanic about what component that is faulty when the truck arrives to the repair shop. This way, the need for manual trouble-shooting is reduced to a minimum and thus saving both time and money for the customer.

The diagnosis system typically consists of several tests that are run while driving on the computer controlling the engine. A test can for example be to check whether the cooling water is boiling or not or to compare a measured signal to an estimation of the same signal.

1.2 Existing Work

Designing a diagnosis system manually is a difficult work that requires a lot of time and engineering skills. Therefore, several master's thesis have been performed at Scania with the purpose of constructing a MATLAB-toolbox for automatic generation of the diagnosis system.

The first step was taken in [7], where a SIMULINK-model was transformed to analytical equations, and from these equations, small parts of the engine that could hypothetically be used as tests were picked out. The parts of the engine that were picked out were *overdetermined* and are good candidates to use for tests in a diagnosis system. This is described more in Chapter 2. Later, in [16] the sets found in [7] were examined and in the cases were the tests were possible to execute, they were also implemented. This provided possible tests that could be used in a diagnosis system. The next step was taken in [6] where the tests were evaluated more thoroughly and the "best" tests, i.e. the tests that were most sensitive to faults in the engine model, were picked out to be a part of the diagnosis system.

1.3 Target Group

The target group is engineers at Scania CV AB and undergraduate science students with an interest in diagnosis.

1.4 Objectives

The work previously done in [7, 16, 6] has provided a foundation towards a completely automated design of the diagnosis system for the engine. However, some problems still remains, for example about the stability of the system. This thesis aims to investigate whether methods from differential algebra can be used to improve the design of the diagnosis system.

1.5 Thesis Outline

This section describes the outline of the thesis.

Chapter 1 gives an introduction to the thesis.

Chapter 2 presents a background to model based diagnosis and structural analysis.

Chapter 3 starts with a introduction to Gröbner basis and differential algebra. It also explains the need for a polynomial engine model and describes how such a model could be made.

Chapter 4 contains a comparison between Gröbner basis and the Rosenfeld-Gröbner algorithm implemented in the MAPLE package `diffalg`.

Chapter 5 presents the results from the differential-algebraic approach to diagnosis for the engine model.

Chapter 6 presents some numerical approaches and shows why these are not suitable for a diesel engine.

Chapter 7 concludes the thesis and presents some ideas for future work.

Chapter 2

Model Based Diagnosis

This chapter briefly describes the concept of model based diagnosis. For more details, see for example [20].

2.1 Model Based Diagnosis

As mentioned in Section 1.1, due to legislation demands, the OBD system is required to find faults on the diesel engine. One way of finding these faults is *model based diagnosis*. The basic idea of model based diagnosis is to use measurements and a model of the engine. This knowledge is then used to try to decide whether a fault has occurred or not.

There are several ways to detect faults, one is to calculate *consistency relations*[1]. A consistency relation is a relation between known variables that, in the fault free case, always holds.

Example 2.1

Consider the system

$$\begin{aligned} \dot{x} &= -x + u \\ y &= x \end{aligned} \tag{2.1}$$

with one state variable x, one known actuator signal u, and one known sensor y. Using Equation (2.1), the unknown variable x can be eliminated and the consistency relation $\dot{y} + y - u = 0$ can be calculated.

When designing a diagnosis system, it is important to know what consistency relations that might not hold for a certain fault and what consistency relations that will be unaffected from the fault. A consistency relation that, when a specific fault occurs, does not hold is said to be *sensitive* to this fault. When a static fault can be detected, the consistency relation is said to be *strongly sensitive*. Correspondingly, a consistency relation that can only detect dynamic faults is said to be *weakly sensitive*.

[1] Other terms often used in literature are parity relations or analytical redundancy relations.

Example 2.2

Let y be a known sensor signal, u a known actuator signal. Let f_u and f_y be two faults modeled as

$$\begin{aligned} \dot{x} &= u + f_u \\ y &= x + f_y \end{aligned} \tag{2.2}$$

From this, by eliminating the unknown state variable x, the relation

$$\dot{y} - \dot{f}_y - u - f_u = 0 \tag{2.3}$$

can be calculated. This is not a consistency relation since it contains the unknown signals f_u and f_y. The corresponding consistency relation is $\dot{y} - u = 0$ and is strongly sensitive to the fault f_u and weakly sensitive to the fault f_y since the consistency relation will not hold when the fault f_y has a non-zero derivative while if f_y is a constant fault, the consistency relation will still hold.

2.1.1 Realization of Consistency Relations

Consistency relations normally consist of measured signals, but also time derivatives of these that are normally not known. Estimating the derivative of a noisy signal can be difficult, and if it is a high order derivative, almost impossible [20]. Because of this, consistency relations usually has to be realized on state-space form. In the linear case, this problem is already solved by adding dynamics to the consistency relation, see for example [20, ch. 5] for details.

In the nonlinear case, the problem is not that easy, but attempts have been made in for example [10] to extend the procedure to a class of polynomial functions.

There exists methods for estimating the derivative, such as lowpass-filtering and using difference quotients or curvefitting using polynomials and then analytically differentiate these. However, since realization of consistency relations has not been considered in this thesis, this will not be discussed further.

2.2 Structural Analysis

With a large model, such as an engine model with 150 equations, it is difficult to know what combinations of equations to use when starting to calculate consistency relations. For this purpose, *structural analysis* has been shown to be useful at Scania. Structural analysis only considers the structure of what variables are part of which equations, the analytical relations are completely left out. This gives a much easier system to analyze. The information on which variables that are included in which equations is often represented in a *biadjacency matrix*, where an "x" in position (i, j) represents that variable j is included in equation i. This is best illustrated with an example.

Example 2.3

Consider the model

$$\begin{aligned} e_1: &\quad x_1^3 = \cos x_2 \\ e_2: &\quad \sqrt{x_2} = 5e^{x_2} + u_1 \\ e_3: &\quad y_1 = x_1 \\ e_4: &\quad y_2 = x_2 \end{aligned}$$

with four equations, two unknown variables x_1 and x_2, one known actuator signal u_1 and two sensor signals y_1 and y_2. The structural representation of this system can be represented as the following biadjacency matrix.

Equation	Unknown		Known		
	x_1	x_2	u_1	y_1	y_2
e_1	x	x			
e_2		x	x		
e_3	x			x	
e_4		x			x

This representation can now be used to analyze the system structurally.

The purpose of using structural analysis in diagnosis is that with this representation, it is much easier to pick out overdetermined equation sets than if the complete analytical relations were considered. These sets can, hypothetically, lead to consistency relations.

Definition 2.1 (Structurally overdetermined set, [17]) *A set of equations E is said to be* structurally overdetermined *with respect to the set of variables X iff*

$$|E| > |var_{X,E}(|) \tag{2.4}$$

With words, Definition 2.1 means that a system is structurally overdetermined if there are more equations than unknown variables.

Structurally overdetermined sets are basically sets where, hypothetically, all unknown variables can be eliminated. However, when calculating a consistency relation, only sets of equations with one more equation than unknown are necessary. Therefore, using the definition of structurally overdetermined, *minimally structurally overdetermined* (MSO) sets are defined.

Definition 2.2 (Minimally structurally overdetermined set, [18]) *A structurally overdetermined set is a* minimally structurally overdetermined *(MSO) set if none of its proper subsets are structurally overdetermined.*

MSO sets are basically sets with one more equation than unknown where every unknown variable can be eliminated. If an equation is removed from the set, this is no longer possible. The difference between an MSO set and a set of equations with one more equation that unknown is illustrated in the following example.

Example 2.4

The set of equations

$$\begin{aligned} e_1 : \quad & x_1 = x_2 + u \\ e_2 : \quad & x_2 = \cos x_2 + 2u \\ e_3 : \quad & y = \sin x_2 \end{aligned}$$

where x_1 and x_2 are unknown variables, u is a known actuator and y is a known sensor signal is structurally overdetermined according to Definition 2.1. Also, it has one more equation than unknown. However, it is not an MSO set since the proper subset of equations $\{e_2, e_3\}$ is structurally overdetermined. The set $\{e_2, e_3\}$ is however an MSO set according to Definition 2.2 since neither e_2 nor e_3 are structurally overdetermined.

MSO sets contain exactly enough information to produce a consistency relation, but it is far from certain that it is possible to eliminate the unknown variables. In this thesis, the MATLAB implementation described in [7] has been used to find all possible MSO sets. The implementation takes a SIMULINK-model as input and transforms it into a structural model. The implementation then finds all possible MSO sets and returns these. The algorithm used is based on graph-theoretical methods and can found in [17].

2.2.1 Derivatives in Models

In structural analysis, derivatives of signals can be handled in two ways. Either by treating $\dot{x}$ and x as the same variable or by treating them as two different variables. If the variables are treated as two different variables, the structural model is called a differentiated-separated structural-model (DSSM) [17] and if the variables are treated as the same variable the structural model is called a differentiated-lumped structural-model (DLSM). In the previous work done at Scania [7], only DLSM models were used. The difference between the two models are illustrated in the following example.

Example 2.5

The model

$$\begin{aligned} e_1 : \quad & \dot{x}_1 = -x_1 + u \\ e_2 : \quad & y = x_1 \end{aligned}$$

is as a DSSM represented as

Equation	Unknown		Known	
	$\dot{x}_1$	x_1	u	y
e_1	x	x	x	
e_2		x		x

while as a DSLM it is represented as

Equation	Unknown	Known	
	x_1	u	y
e_1	x	x	
e_2	x		x

Here it can be noted that the model structurally represented as a DSLM is an MSO set and represented as a DSSM set it is not. To get an MSO set with DSSM representation, e_2 has to be differentiated. This would give the structural representation

Equation	Unknown		Known		
	$\dot{x}_1$	x_1	u	y	$\dot{y}$
e_1	x	x	x		
e_2		x		x	
$\dot{e}_2$	x				x

In Example 2.5, when the model was represented as a DSSM, some equations had to be differentiated if an MSO set should be found. This is interesting and will be discussed in Chapter 4.

Chapter 3

Gröbner Bases and Polynomials

This chapter describes some theory that can be used to derive consistency relations.

3.1 Differential Gröbner Bases

Calculating a consistency relation typically includes eliminating unknown variables from a set of equations. In the elimination process, methods from differential algebra can be used. Given a set of non-differential polynomial equations, a *Gröbner basis* can be calculated to eliminate the unknown variables. Non-differential means that x and $\dot{x}$ are treated as completely different variables. Similar to Gaussian elimination for linear systems, when calculating a Gröbner basis, variables are eliminated and the result is a set of equations with a sort of "triangular" structure, i.e. the first equation contains all variables, the second equation might contain an equal number of equations as the first or one less and so on. The result is a polynomial basis that, in the usual case, contains more equations than the original basis but that spans the same space as the original equations, i.e. they have the same solution set.

Theoretically such reduction always terminate. However, in practice, for example the limited memory of a computer makes it sometimes impossible to calculate a Gröbner basis for some sets of equations. For an introduction to Gröbner bases and how to compute these see [4, 5].

Example 3.1

Given a linear MSO set

$$\begin{aligned} x &= u \\ y &= x \end{aligned} \tag{3.1}$$

that should hold in the fault free case, and using Gaussian elimination to eliminate the unknown variable x, we get

$$\begin{array}{rcrcrcl} x & - & u & & & = & 0 \\ & - & u & + & y & = & 0 \end{array} \tag{3.2}$$

The MSO set could be seen as a linear subspace in $\mathcal{R}^3$ that, when the model is correct, we will never leave. The consistency relation $y - u = 0$ could just be seen as a base vector instead of $y - x = 0$. The difference is that this base vector is known, and checking if the consistency relation holds could be seen as a test if the model still is in the same subspace as in the fault free case.

Now, compare instead the nonlinear model

$$\begin{aligned} x^2 + y^2 + u^2 &= 1 \\ x + y &= 1 \end{aligned} \tag{3.3}$$

where the first equation describes the unit sphere and the second describes a plane. The space spanned by the two equations is a circle in $\mathcal{R}^3$. By calculating a Gröbner basis, the result will be

$$\begin{aligned} x + y &= 1 \\ 2y^2 - 2y + u^2 &= 0 \end{aligned} \tag{3.4}$$

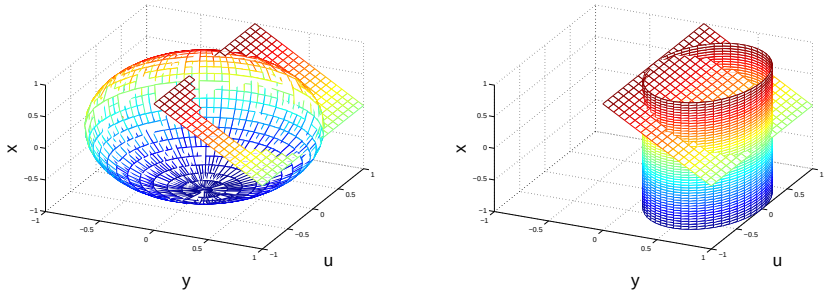
The second equation can be rewritten as

$$\begin{aligned} 2y^2 - 2y + u^2 = 0 \iff \\ \left(\frac{y - \frac{1}{2}}{\frac{1}{2}}\right)^2 + \left(\frac{u}{\frac{1}{\sqrt{2}}}\right)^2 = 1 \end{aligned} \tag{3.5}$$

which, in $\mathcal{R}^3$, is an elliptic cylinder. The space spanned by Equation (3.3) and Equation (3.4) can be seen in Figure 3.1. The original circle is expressed as the intersection of a plane and an elliptic cylinder. Checking if the consistency relation still holds could be seen as checking if we still are on the known elliptic cylinder.

There are limitations with Gröbner bases, for example they only handle polynomials and can be very computer intense. Another problem is that they do not handle derivatives and therefore some equations need to be differentiated by hand. Using methods from differential algebra, the problem with differentiated variables can be handled by the `diffalg` [14] package in MAPLE. The `diffalg` package can handle polynomial systems, with algebraic as well as partially differentiated variables.

The `diffalg` package mainly uses the *Rosenfeld-Gröbner* algorithm to calculate a triangular differential basis. The algorithm consists of two steps, first the differential step which "reduces differential problems to purely algebraic ones" which is the "Rosenfeld"-part of the algorithm [2]. The second step is the "purely



(a) The spaces spanned by Equation (3.3).

(b) The spaces spanned by Equation (3.4).

Figure 3.1: The manifold from Equation (3.3) represented as the intersection of a plane and a sphere and as a plane and an elliptic cylinder.

algebraic step" which originally performed Gröbner bases calculations to calculate a triangular base for the system, hence the name Rosenfeld-Gröbner. It should here be pointed out that the algorithm does not first differentiate the whole system and then perform the algebraic calculations but rather iterates between the two steps. Later versions and implementations of the algorithm uses special properties of the system so that the use of Gröbner bases is no longer necessary. These are faster and more computationally efficient [2]. The Rosenfeld-Gröbner algorithm was first presented in [3] but a short introduction can be found in [23].

3.1.1 Ranking

A *ranking* is a way to decide in what order variables should be eliminated with `diffalg`, the corresponding term for Gröbner bases is called *variable ordering*. Several rankings exist, but the most interesting for diagnosis are *pure lexicographic ordering* and *elimination ordering*. Pure lexicographic ordering means that a list is taken and `diffalg` or the Gröbner basis command tries to eliminate the variables in the list in the order they appear. For example if the list $[x_1, x_2, x_3]$ is given and pure lexicographic ordering is specified, MAPLE first tries to eliminate x_1, then x_2, and last x_3. Elimination order instead lets the user specify two sets where each variable in the first set should be eliminated before any variable in the second set, i.e. if $([v_1, \ldots, v_n], [w_1, \ldots, w_p])$ is given and elimination ordering is specified, this tells MAPLE that it should eliminate all the variables v_i before any variable w_j, if possible. This makes it good to use to calculate consistency relations since the ranking can be specified as (X, Z) where X is the set of all unknown variables and Z is the set of all known variables in the equations of the MSO set.

In this thesis, the ranking that x_1 should be eliminated before x_2 is denoted $x_1 \prec x_2$. There also exist other rankings where it is possibly to specify for example that $x_1^2 \prec x_2^2 \prec x_1 \prec x_2$ or, for `diffalg`, that any derivative of a variable should be ranked higher than a non-differentiated variable.

An example on how to compute a consistency relation using differential Gröb-

ner bases in MAPLE can be found in the following example.

Example 3.2

We have the system

$$\begin{aligned} e_1 : \quad & \dot{x}_1(t) = x_1(t) + 2x_2(t) \\ e_2 : \quad & \dot{x}_2(t) = x_2^2(t) + u(t) \\ e_3 : \quad & y(t) = x_1(t) \end{aligned}$$

To compute a consistency relation with `diffalg` in MAPLE the following can be entered

```
> with(diffalg):
> e1:= diff(x1(t),t)-x1(t)-2*x2(t):
> e2:= diff(x2(t),t)-x2(t)^2-u(t):
> e3:= y(t)-x1(t):
> E := [e1,e2,e3]:
> R := differential_ring(ranking=[x1,x2,y,u],
>              derivations=[t],notation=diff):
> G := Rosenfeld_Groebner(E, R):
> C := equations(G):
```

The first line loads the `diffalg` package so the commands used becomes available. The next 4 lines defines the equations. The `diffalg` package as well as the Gröbner basis command takes the ingoing equations and assumes these are equal to 0. Next, `R` is defined using the command `differential_ring`.

The command `differential_ring` takes three arguments. The first specifies the ranking. In this example, `ranking=[x1,x2,y,u]` specifies pure lexicographic ordering, i.e. the variables are ranked as $x_1 \prec x_2 \prec y \prec u$. The next argument, `derivations=[t]` specifies that `t` is the only variable that we will differentiate with respect to. The last argument to `differential_ring` specifies that differentiations are notated as `diff(x(t),t)`.

The second last line solves the problem and calculates a triangular basis from the equations in `E` and the specifications in `R`. The last line extracts the equations and from `C` we can now extract the consistency relation $2\ddot{y}(t) - 2\dot{y}(t) - \dot{y}(t)^2 + 2\dot{y}(t)y(t) - y(t)^2 - 4u(t) = 0$. This consistency relation can now be used to detect faults in the system.

As mentioned earlier, eliminating all unknown variables using `diffalg` or using the Gröbner basis algorithm can be a very computer intense task. The runtime of the Gröbner basis algorithm for the worst case scenario is $\mathcal{O}\left(2^{2^n}\right)$ where n is the number of the variables [1]. The complexity of the Rosenfeld-Gröbner algorithm is not yet known [12], but since new unknowns are continually introduced in differential algorithms, the complexity is even worse than for the Gröbner basis algorithm [23, p. 13].

It should here be noted that, even if the complexity of the Rosenfeld-Gröbner algorithm was known, these algorithms solve different problems and the complexity can not be compared, at least not directly. This is because the Gröbner basis algorithm does not handle derivatives and some equations needs to be differentiated leading to a greater number of equations. To calculate the consistency relation in Example 3.2 with `diffalg`, it was sufficient to use the set of equations $\{e_1, e_2, e_3\}$. However, if the Gröbner basis algorithm should be used, the set of equations $\{e_1, \dot{e}_1, e_2, e_3, \dot{e}_3, \ddot{e}_3\}$ must be used, i.e. six equations compared to the three equations that `diffalg` needed. This is further investigated in Chapter 4.

3.2 Polynomial Representation of Nonlinearities

Since `diffalg` and Gröbner bases only handle polynomial nonlinearities, other nonlinearities have to be replaced by polynomials. The nonlinearities in the engine model were handled differently depending on the sort of nonlinearity.

3.2.1 Fractions

Fractions were handled by finding the least common denominator of the equations and multiplying with that to change the equation. E.g. the equation

$$x_1 + \frac{x_2}{x_1} = x_3 \tag{3.6}$$

would be rewritten as

$$x_1^2 + x_2 = x_1 x_3 \tag{3.7}$$

3.2.2 Lookup Tables

In the model of a Scania engine, many lookup tables are used. A lookup table can be seen as a function that takes one or more input signals x and numerically maps these to an output y, i.e. $y = f(x)$ where f is defined by numerical values. When approximating the lookup table with a polynomial, the first step would be to choose what terms that should be included. This will be discussed in Section 3.3. If for example the lookup table has only one input and N terms should be included, the estimated output, $\hat{y}$, can then be written as $\hat{y} = a_0 + a_1 x + \ldots + a_N x^N = \sum_{j=0}^{N} a_j x^j$. Let x_i and y_i denote values of x and y that $y_i = f(x_i)$. Then the goal is to minimize the sum $\sum_{i=0}^{n} (y_i - \hat{y}_i)^2$, i.e. minimize the 2–norm of the estimated error. To do so, create the matrices A, z and b where

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^N \end{pmatrix}, \quad z = \begin{pmatrix} a_0 \\ \vdots \\ a_N \end{pmatrix}, \quad b = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{3.8}$$

Now the overdetermined *linear* equation system $Az = b$ can be formed and solved in a least square sense[1] as $z = (A^TA)^{-1}A^Tb$. Lookup tables with more than one input can be treated in the same way by changing the matrix A so it consists of polynomials in all variables as well as mixed terms.

3.2.3 Square Roots

Square roots can be eliminated in four ways

- By doing the same procedure as in Section 3.2.2 and "guessing" a degree of the polynomial needed.
- By doing a Taylor expansion around a working point and guessing the needed number of terms.
- By moving everything but the square root to one side of the equations and squaring both sides.
- By introducing a help variable, e.g. transforming

$$\sqrt{x_1}x_2 = x_3 \tag{3.9}$$

to the equation system

$$\begin{aligned} s_1x_2 &= x_3 \\ s_1^2 &= x_1 \end{aligned} \tag{3.10}$$

The first two methods are only approximations of the true functions and were therefore not used. The third approach was used if the square root already was the only term on one side of the equation, since that would keep the number of equations lower than the fourth. If not, the fourth approach was used. Note that using a help variable or squaring the equations to get rid of square roots does not give an equivalent model since in Equation (3.9), $\sqrt{x_1}$ is required to be non-negative while in Equation (3.10) no such restriction exist on s_1.

3.2.4 Other Analytical Functions

Analytical functions can be treated as either of the first two methods in Section 3.2.3. In this thesis the first of the two methods were used since that gives an equally good approximation over the whole working area while a Taylor series is a very good approximation close to the working point but becomes worse the further away from the working point the signals are.

[1] It's not necessary to calculate the inverse of A^TA, it is faster and more numerically stable to solve the equation system with Gaussian elimination. In MATLAB this can be done by typing `x = A\b`.

3.3 Order of Polynomials

The order of the polynomials chosen, greatly impacts the fit of the model. A higher order always gives a better fit, but can lead to overly complex models. To avoid this, the model complexity can be weighed in together with the model fit. One of the most used methods is the *Akaike information criterion* [11]

$$\text{AIC} = \min_{d,\theta} \left(1 + \frac{2d}{N}\right) \sum_{t=1}^{N} \epsilon(t,\theta)^2 \tag{3.11}$$

where N is the number of samples, θ are the estimated parameters, d is dimension of θ and ϵ is the prediction error.

With only one input, it's generally easy to choose what terms to add. If the fit is to low, add the next higher order term and try again. If there is two or three inputs, it is not that easy. One way to do this is to use a *Genetic algorithm* approach such as the one used in [21]. Genetic algorithms is an optimization technique that is inspired by Darwin's theory of evolution and the *Survival of the fittest*. The algorithm starts with the initialization of a population, in literature often called *chromosomes*, which are possible solutions to the optimization problem. The chromosomes are often, but not always, coded as binary strings where each bit represents a certain property of the chromosome. A "1" means the chromosome has that specific property and a "0" means it does not. If the algorithm is used for system identification, a natural coding would be that each bit of the chromosome represents a term in the expression. For example the short binary string "1010" could mean that this chromosome represents a model that has a constant and a quadratic term but no linear or cubic term. After the chromosomes have been initialized, they are evaluated with respect to some fitness function. This fitness function can take several different things in account, for system identification this fitness function can for example take model complexity and model fit. If chromosome i is denoted c_i, the fitness function could be chosen as

$$f(c_i) = f_{complexity}(c_i) + k_{fit} f_{fit}(c_i) \tag{3.12}$$

where k_{fit} is a parameter that can be used to weigh model complexity against the fit of the model. As an example, f_{fit} could be the prediction error of the model and $f_{complexity}$ could be defined as

$$f_{complexity}(c_i) = k_{degree} n(c_i) \tag{3.13}$$

where k_{degree} is a constant and n is the total degree of the terms. Other ways of defining the complexity could of course be used, such as giving mixed terms higher penalty than terms with only one variable in.

Next is the reproductive step, where the chromosomes from the old generation reproduce. Typically, a better fitness gives a bigger the chance to reproduce. Here it is possible to implement several different enhancements to the reproduction part of the algorithm, such as *mutation* and *elitism* that are described in Algorithm 3.1. The new generation is then evaluated and then the algorithm starts over again.

The algorithm runs until some condition is fulfilled, for example a certain limit on the fitness or after a predefined time. It is important to point out that this algorithm does not necessarily give the optimal solution, but rather a "close" to optimal solution. The algorithm is described in Algorithm 3.1. One drawback using genetic algorithms is that the runtime is nondeterministic, i.e. there is no way to know in advance how long time the algorithm needs to run until a satisfactory result is achieved. However, since this is can be done offline this is not a problem in this case.

Algorithm 3.1 Genetic algorithm

Start Initialize populations of N chromosomes c_i.
while Termination condition is false **do**
  for all i **do**
    Evaluate fitness $f(c_i)$ for chromosomes i
  end for
  Reproduce Create a new generation by using the following steps

1. *Selection*: Select the individuals that will be allowed to reproduce. This is done according to some scheme, for example better fitness gives better chance to reproduce.
2. *Crossover*: Combine the selected chromosomes to form the new generation.
3. *Mutation*: With some probability, mutate the new offspring.
4. *Elitism*: Add the best individuals from the old generation to the new generation.
5. *Replace*: From now on, use the new generation in the algorithm.

end while

To illustrate the algorithm, a small example is seen in Example 3.3.

Example 3.3

As an example, one of the lookup tables used in the engine model was approximated by using Algorithm 3.1. The lookup table has two inputs, from now on called x and y, and one output, called z. It was decided that the total degree of each term would not be allowed to be higher than 5. This resulted in a chromosome with 21 bits. Below is how the chromosome was decoded with bit 1-7 in the first row, bit 8-14 in the second row and bit 15-21 in the last row.

1	x	x^2	x^3	x^4	x^5	y
y^2	y^3	y^4	y^5	xy	xy^2	xy^3
xy^4	x^2y	x^2y^2	x^2y^3	x^3y	x^3y^2	x^4y

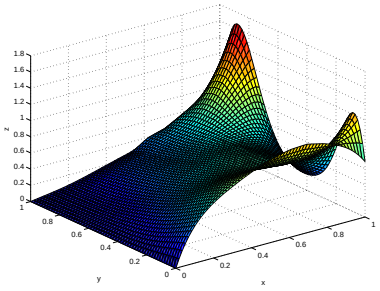
This means that a chromosome with the first three bits as ones and the 18 remaining as zeros would mean that the lookup table should be approximated as $a_0 + a_1x + a_2x^2$.

As fitness function, Equation (3.12) was used with Equation (3.13) as the complexity function. Algorithm 3.1 was used to find what terms that should be used. It is implemented in the MATLAB toolbox GENETIC ALGORITHM AND DIRECT SEARCH TOOLBOX.

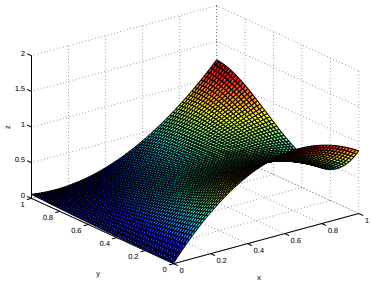
The algorithm converges in under 20 seconds, resulting in a polynomial with 7 terms. The resulting polynomial was

$$a_0 + a_1x^2a_2y + a_3y^2 + a_4xy + a_5x^2y^2 + a_6x^3y^2 \tag{3.14}$$

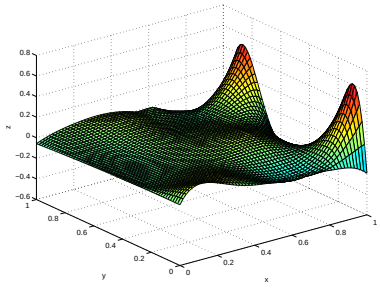
In Figure 3.2 the original lookup table, the approximated lookup table as well as the absolute error can be seen. One can see that the error is quite large at the two peaks in the corners. One way to reduce the height of these peaks would be to increase k_{fit} but that would result in a more complex expression.



(a) The original lookup table.



(b) The lookup table approximated with a polynomial.



(c) The absolute error of the approximated lookup table.

Figure 3.2: The lookup table used in Example 3.3.

3.4 Interesting Sections

When representing a function as a polynomial it is important to think of what part of the function it needs to be valid for. Since a consistency relation could be seen

as a test if the model is valid or not (see Section 2.1), the polynomial model needs to be a good approximation to the original model for *all possible values the signals might take* since, if the signals take values that have not been accounted for in the modeling process, the OBD system would give false alarms since the model would (probably) not be valid for these values. This implies that to make a polynomial model of an engine that is accurate over a large working area, the complexity of the polynomials will be high and that a lot of data is needed. However, this problem has not been addressed and is considered out of scope for this thesis.

3.5 An Example of Finding Consistency Relations Using `diffalg`

This section demonstrates, in an example, the possibilities of the method with structural analysis and differential Gröbner bases. The example is taken from [8] and is two coupled water tanks, with two sensors measuring the water level in the tank and two sensors measuring the outflow of the tanks. The process also has an actuator which controls a pump that fills the upper tank. The equations describing the process are

$$\begin{aligned} \dot{h}_1 &= d_1 u - d_2\sqrt{h_1} \\ \dot{h}_2 &= d_3\sqrt{h_1} - d_4\sqrt{h_2} \\ y_1 &= h_1 \\ y_2 &= h_2 \\ y_3 &= d_5\sqrt{h_1} \\ y_4 &= d_6\sqrt{h_2} \end{aligned} \tag{3.15}$$

where d_i are model parameters, u is the control signal, y_i are the measurement signals and h_i are the height in the tanks. To get a polynomial model of the system, two help variables need to be introduced to avoid the square roots, see Section 3.2.3. The new model with polynomials is

$$\begin{aligned} e_1 &: & \dot{h}_1 &= d_1 u - d_2 s_1 \\ e_2 &: & \dot{h}_2 &= d_3 s_1 - d_4 s_2 \\ e_3 &: & s_1^2 &= h_1 \\ e_4 &: & s_2^2 &= h_2 \\ e_5 &: & y_1 &= h_1 \\ e_6 &: & y_2 &= h_2 \\ e_7 &: & y_3 &= d_5 s_1 \\ e_8 &: & y_4 &= d_6 s_2 \end{aligned} \tag{3.16}$$

Note that the model in Equation (3.16) is not equivalent with the model in Equation (3.15) since in Equation (3.15), it is required that $\sqrt{h_i}$ is non-negative but

in Equation (3.16) both positive and negative values on s_i satisfies the equations. The corresponding biadjacency matrix is seen in Table 3.1.

Equation	Unknown				Known				
	h_1	h_2	s_1	s_2	u	y_1	y_2	y_3	y_4
e_1	x		x		x				
e_2		x	x	x					
e_3	x		x						
e_4		x		x					
e_5	x					x			
e_6		x					x		
e_7			x					x	
e_8				x					x

Table 3.1: The biadjacency matrix for Equation (3.16).

From this structural model it is possible to extract 17 MSO sets, all of which are possible to eliminate the unknown variables from. Using `diffalg`, the consistency relations were calculated in under 30 seconds. The consistency relations are

$$d_3^4 y_1^2 - 2d_3^2 d_4^2 y_1 y_2 + d_4^4 y_2^2 - 2d_3^2 y_1 \dot{y}_2^2 - 2d_4^2 y_2 \dot{y}_2^2 + \dot{y}_2^4 = 0 \quad (3.17a)$$

$$d_5^2 y_1 - y_3^2 = 0 \quad (3.17b)$$

$$-d_3^2 d_6^4 y_1 + d_4^2 d_6^2 y_4^2 + 4d_4 d_6 y_4^2 \dot{y}_4 + 4y_4^2 \dot{y}_4^2 = 0 \quad (3.17c)$$

$$d_1^2 u^2 - 2d_1 u \dot{y}_1 - d_2^2 y_1 + \dot{y}_1^2 = 0 \quad (3.17d)$$

$$-d_4^2 d_5^2 y_2 + d_3^2 y_3^2 - 2d_3 d_5 y_3 \dot{y}_2 + d_5^2 \dot{y}_2^2 = 0 \quad (3.17e)$$

$$d_6^2 y_2 - y_4^2 = 0 \quad (3.17f)$$

$$d_1^2 d_3^4 u^2 y_2 - 2d_1 d_2 d_3^3 u y_2 \dot{y}_2 - 2d_1 d_3^2 d_4^2 u y_2 \dot{y}_2 - d_2^2 d_3^2 d_4^2 y_2^2 - 4d_1 d_3^2 u y_2 \dot{y}_2 \ddot{y}_2 +$$
$$d_2^2 d_3^2 y_2 \dot{y}_2^2 - 4d_2 d_3 d_4^2 y_2^2 \ddot{y}_2 + d_4^4 y_2 \dot{y}_2^2 + 4d_2 d_3 y_2 \dot{y}_2^2 \ddot{y}_2 - 4d_4^2 y_2^2 \ddot{y}_2^2 - d_4^2 \dot{y}_2^4 + 4y_2 \dot{y}_2^2 \ddot{y}_2^2 = 0 \quad (3.17g)$$

$$-d_3 d_6^2 y_3 + d_4 d_5 d_6 y_4 + 2d_5 y_4 \dot{y}_4 = 0 \quad (3.17h)$$

$$-d_1 d_5^2 u + d_2 d_5 y_3 + 2y_3 \dot{y}_3 = 0 \quad (3.17i)$$

$$d_1 d_3^2 d_6^4 u - d_2 d_3 d_4 d_6^3 y_4 - 2d_2 d_3 d_6^2 y_4 \dot{y}_4 - 2d_4^2 d_6^2 y_4 \dot{y}_4 - 4d_4 d_6 y_4^2 \ddot{y}_4 - 8d_4 d_6 y_4 \dot{y}_4^2 - 8y_4^2 \dot{y}_4 \ddot{y}_4 - 8y_4 \dot{y}_4^3 = 0 \quad (3.17j)$$

$$-d_3^2 d_6^2 y_1 + d_4^2 y_4^2 + 2d_4 d_6 y_4 \dot{y}_2 + d_6^2 \dot{y}_2^2 = 0 \quad (3.17k)$$

$$d_1^2 d_3^2 u^2 - 2d_1 d_2 d_3 u \dot{y}_2 - 2d_1 d_3^2 u \dot{y}_1 - d_2^2 d_4^2 y_2 + d_2^2 \dot{y}_2^2 + 2d_2 d_3 \dot{y}_1 \dot{y}_2 + d_3^2 \dot{y}_1^2 = 0 \quad (3.17l)$$

$$-d_1 d_5 u + d_2 y_3 + d_5 \dot{y}_1 = 0 \quad (3.17n)$$

$$-d_1 d_3 d_6^2 u + d_2 d_4 d_6 y_4 + d_3 d_6^2 \dot{y}_1 + 2d_2 y_4 \dot{y}_4 = 0 \quad (3.17o)$$

$$-d_3 d_6 y_3 + d_4 d_5 y_4 + d_5 d_6 \dot{y}_2 = 0 \quad (3.17p)$$

$$-d_1 d_3^2 d_6^2 u + d_2 d_3 d_4 d_6 y_4 + d_2 d_3 d_6^2 \dot{y}_2 + 2d_4^2 y_4 \dot{y}_4 + 2d_4 d_6 y_4 \ddot{y}_2 + 2d_4 d_6 \dot{y}_2 \dot{y}_4 + 2d_6^2 \dot{y}_2 \ddot{y}_2 = 0 \quad (3.17q)$$

$$-d_1 d_3 d_6 u + d_2 d_4 y_4 + d_2 d_6 \dot{y}_2 + d_3 d_6 \dot{y}_1 = 0 \quad (3.17r)$$

Of the 17 consistency relations that were calculated, only two consisted of non-differentiated signals. To evaluate the consistency relations, derivatives were estimated by curve-fitting a polynomial of degree three in a neighborhood around the point where the derivative was estimated. After that the polynomial was analytically differentiated, i.e.

1. For each sample, t_n, the derivative should be estimated in, pick out samples around t_n and denote that batch of data $t = \{t_k\}, k = n - m, \ldots, n + m$
2. In a least square sense, see Section 3.2.2, estimate the coefficients to the polynomial $y(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

3. Estimate the derivatives in t_n, i.e. save $\dot{y}(t_n) = a_1 + 2a_2t_n + 3a_3t_n^2$ and $\ddot{y}(t_n) = 2a_2 + 6a_3t_n$

Estimating the derivatives like this is a computer-intense task that can probably not be used in an online application. However, for offline estimation it works fine.

The only faults considered in this example are additive faults on the sensors y_i and the actuator signal u. It is obvious that the consistency relations sensitive to a fault in sensor y_i will be the consistency relations that include the signal y_i. The same goes for faults in the actuator signal u. It is also easy to verify that the consistency relations with only $\dot{y}_i$ or $\ddot{y}_i$ and no y_i will only be weakly sensitive to faults in y_i. The sensitivity to faults of the consistency relations are summarized in Table 3.2.

Consistency relation	F_u	F_{y_1}	F_{y_2}	F_{y_3}	F_{y_4}
c_1		1	1		
c_2		1		1	
c_3		1			1
c_4	1	1			
c_5			1	1	
c_6			1		1
c_7	1		1		
c_8				1	1
c_9	1			1	
c_{10}	1				1
c_{11}		1	x		1
c_{12}	1	x	1		
c_{13}	1	x		1	
c_{14}	1	x			1
c_{15}			x	1	1
c_{16}	1		x		1
c_{17}	1	x	x		1

Table 3.2: Sensitivity to faults of consistency relations. A "1" means the consistency relation is strongly sensitive to the fault and an "x" means the consistency relation is weakly sensitive to the fault.

The system was simulated and a simple proportional controller was used to control the system. The reference signal with the control signal as well as the sensor signals with no measurement noise can be seen in Figure 3.3. To illustrate the correctness of the consistency relations, the system was simulated with no faults and no measurement noise at all. The result can be seen in Figure 3.4. It can be noted that the consistency relations that are farest away from zero are consistency relation 7, 10 and 16, that are shown in Equations (3.17g), (3.17j) and (3.17q) are the only ones that contains second order derivatives. This suggests that the estimation of second order derivatives are not good.

To validate that the consistency relations would be possible to detect faults, it was evaluated how well they responded when faults were added in the simulations.

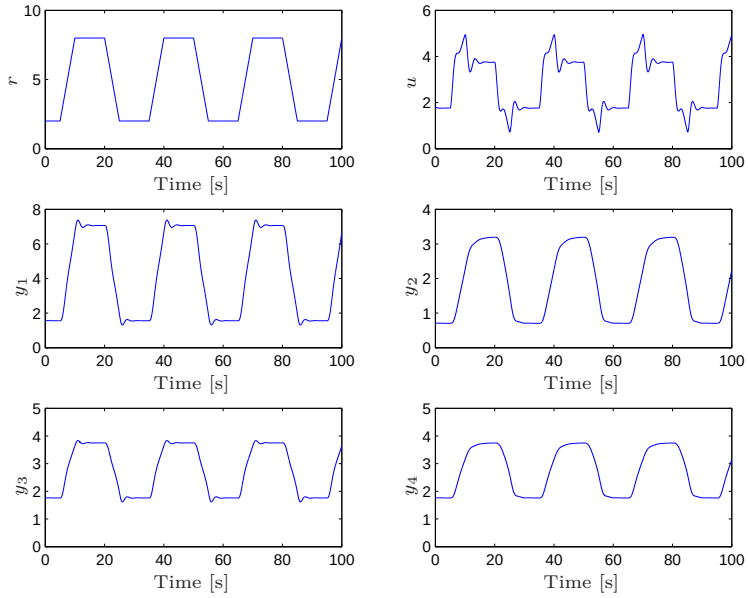


Figure 3.3: Reference, actuator and measurement signals in fault free case with no noise added.

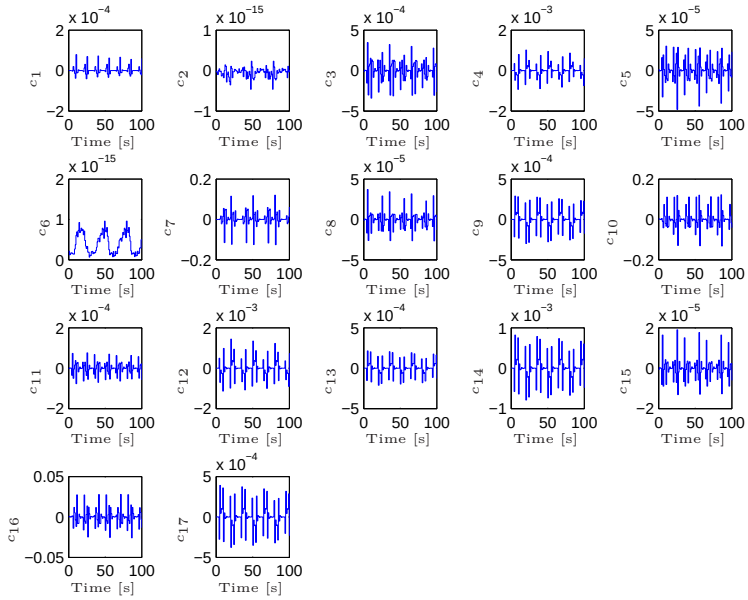
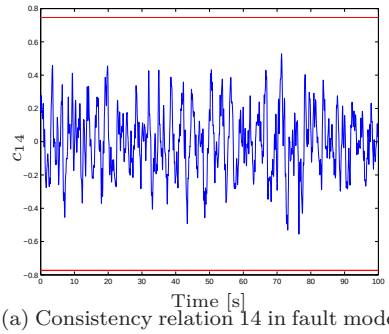


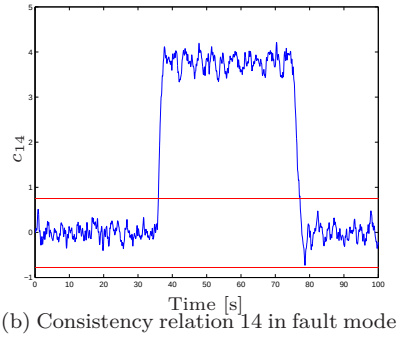
Figure 3.4: The consistency relations in the fault free case with no measurement noise added.

To make the evaluation more realistic, some measurement noise was added as well. To reduce the influence from the added noise, the measurement signals were low-pass filtered. Faults were added as a ramp with slope 1 from 0 at time $t = 35$ s up to the maximum value, and then removed as a ramp at time $t = 75$ s.

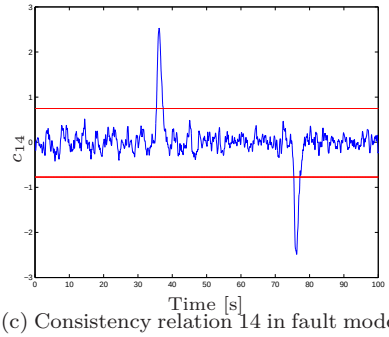
All consistency relations responded as they should according to Table 3.2. However, some consistency relations responded stronger than others. For the sake of brevity, only consistency relation 14 is shown simulated with faults in Figure 3.5. From Table 3.2 it can be seen that the consistency relation should be strongly sensitive to F_u and F_{y_4} and be weakly sensitive to F_{y_1}. In Figure 3.5, this can be verified.



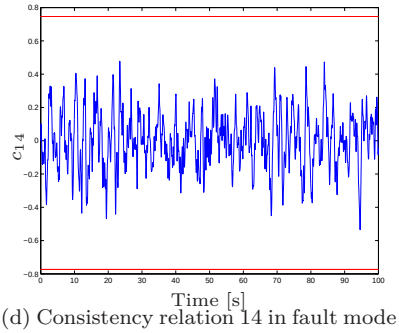
(a) Consistency relation 14 in fault mode NF.



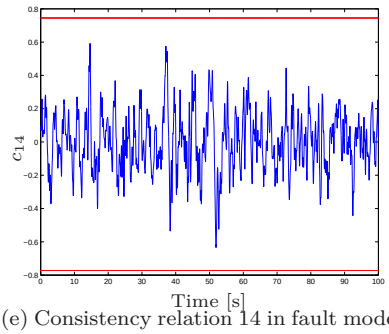
(b) Consistency relation 14 in fault mode F_u.



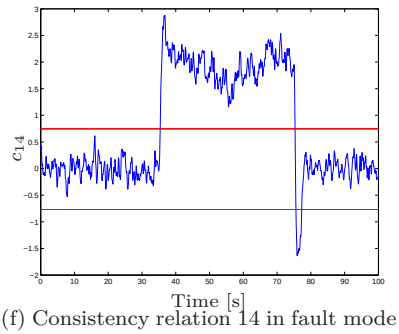
(c) Consistency relation 14 in fault mode F_{y_1}.



(d) Consistency relation 14 in fault mode F_{y_2}.



(e) Consistency relation 14 in fault mode F_{y_3}.



(f) Consistency relation 14 in fault mode F_{y_4}.

Figure 3.5: Consistency relation 14 with faults.

Chapter 4

A Comparison between `diffalg` and Gröbner Bases

As described in Chapter 3, both `diffalg` as well as Gröbner bases can be used to calculate consistency relations for polynomial systems. In this chapter, an attempt to investigate which method that most suitable to use for diagnosis purpose, i.e. is it better to directly use `diffalg` for the differential system, or is it better to "manually" differentiate the system and then use Gröbner basis? I.e. which method is capable of solving the most cases?

In the introduction in [19], it is stated that differential algorithms is to prefer over purely algebraic algorithms such as the Gröbner basis algorithm. However, [19] regards more general systems with partial derivatives in many variables instead of, as in this thesis, only time derivatives. Therefore, an investigation is of interest.

4.1 Differentiating for Gröbner Bases

To calculate a consistency relation using Gröbner basis, usually some equations needs to be differentiated before the calculations can start. The question is which equations need to be differentiated and how many times? In Example 2.3 only one equation needed to be differentiated while in Example 3.2 it was necessary to differentiate one equation once and another equation twice for a total of three differentiations. In [18, p. 204] it is shown that there is an upper bound for how many times each equation needs to be differentiated. Before this bound is given, some variables needs to be introduced.

Starting with a set of equations E, let X denote the set of variables in E where x and $\dot{x}$ are considered to be the same variable structurally. Then let $\bar{X}$ denote the set of variables in E where x and $\dot{x}$ are considered to be different variables. Let $\alpha(x)$ be the highest order derivative of x in E and let ξ be defined as in Equation (4.1).

$$\xi = \sum_{x \in X} (1 + \alpha(x)) \tag{4.1}$$

The number ξ can be seen as an upper limit of the number of variables in $\bar{X}$ included in E. The following example demonstrates X, $\bar{X}$ and ξ for a small set of equations.

Example 4.1

From the set of equations E

$$E = \begin{cases} \dot{x}_1 &= -x_1 + x_2 \\ \dot{x}_2 &= -x_2 + u \\ y &= x_2 \end{cases} \tag{4.2}$$

we get $X = \{x_1, x_2\}$ and $\bar{X} = \{x_1, x_2, \dot{x}_1, \dot{x}_2\}$. We also get $\alpha(x_1) = 1$ and $\alpha(x_2) = 1$ since the highest order derivatives of both x_1 and x_2 is one. Using Equation 4.1 we get $\xi = 2 + 2 = 4$.

Since the engine models used at Scania are implemented in SIMULINK on state space form, only first order derivatives are used. This means α is either 1 if the differentiated state variable is a member of the MSO set or 0 if it is not. Further, this implies that ξ is the number of unknowns plus the number of differentiated variables in the MSO set. This means the we can rewrite ξ to

$$\xi = |X| + N_{\mathrm{diff}_X} \tag{4.3}$$

where N_{diff_X} is the number of differentiated signals in X.

A theorem from [18] is now presented.

Theorem 4.1 *Given an MSO set E with respect to X, an upper limit m for the number of differentiations that are needed to obtain an MSO set with respect to $\bar{X}$ is given by*

$$m = 1 + \xi - |E| \tag{4.4}$$

Proof: *A proof is available in [18].* □

The bound m can, using Equation (4.3) and Equation (4.4), be rewritten as

$$\begin{aligned} m &= 1 + \xi - |E| \\ &= 1 + |X| + N_{\mathrm{diff}_X} - |E| \\ &= N_{\mathrm{diff}_X} \end{aligned} \tag{4.5}$$

The last step in Equation (4.5) comes from the fact that E is an MSO set and has therefore one more equation than unknown. This means that it is not necessary

to differentiate each equation more than the number of differentiated signals in E. This is only a theoretical upper limit though, in practice, it is usually not necessary to differentiate that many times.

Further, in [18], an algorithm is presented that, given an MSO set where x and $\dot{x}$ are considered the same variable, results in an MSO set where x and $\dot{x}$ are considered different variables. It is also shown that the equations in the new MSO set are of *minimal order*. With minimal order, it is meant that the number of differentiations of the equation is minimal. It is also shown that it is not possible to lower the number of differentiations for one equation and increase the number of differentiations of other equations, i.e. it is not possible to differentiate differently and get a consistency relation with first order derivatives of two signals instead of one second order derivative of one signal. The algorithm is shown in Algorithm 4.1.

Algorithm 4.1 Get differentiated MSO set

Input set of equations E.
$M := E$
while $M^+ = \emptyset$ **do**
  $E := \frac{dE}{dt}$
  $M := M \cup E$
end while
return M^+

In words, Algorithm 4.1 can be described to take an MSO set of equations E with respect to X. If E is already overdetermined with respect to $\bar{X}$, then there is nothing more to do. Else differentiate all equations once. If this gives an overdetermined part with respect to $\bar{X}$, take that part out and end the algorithm. If not, keep differentiating until an MSO set with respect to $\bar{X}$ is found. Theorem 4.1 guarantees that the algorithm always terminates in a finite number of iterations. It should also be pointed out that the set of equations E used in the algorithm does not have to be polynomials but can be any expressions.

4.2 Evaluating the Methods

To evaluate whether Gröbner basis or `diffalg` should be used for differential polynomials, equation sets that were MSO sets were randomized using the following design parameters:

$N_{\mathbf{diff}}$**:** Number of unknown variables appearing as differentiated variables.

$N_{\mathbf{alge}}$**:** Number of unknown variables appearing only non-differentiated.

$M_{\mathbf{compl}}$**:** The probability of each term to be nonlinear in the MSO sets. $M_{\text{compl}} = 0$ results in a completely linear system whereas $M_{\text{compl}} = 1$ results in only nonlinear terms. The nonlinear terms can be both mixed terms as well as polynomial terms with a maximal degree of 3.

Variables were randomized in $N_{\text{diff}} + N_{\text{alge}}$ equations with the probability of a term to be nonlinear of M_{compl}. In order to increase similarity to the engine models used at Scania, the differentiated variables where added as $\dot{x}_i = f(x)$ where f is a polynomial function. In each equation, the number of terms was also randomized between two and five. To get an MSO set, one extra equation was added as a measurement equation looking like $y_1(t) = x_i(t), i \in \{1, \ldots, N_{\text{diff}} + N_{\text{alge}}\}$. Added to that, in some algebraic equations, actuator signals were added to have some more known signals in the system. This algorithm does not necessary generate an MSO set of desired order since there might be smaller parts of the set that are also structurally overdetermined. However, when the output was generated, the system was checked and if it was not an MSO set of decided order, the algorithm was run again until it really produced an MSO set of decided order. This is not efficient, however, for smaller values of N_{diff} and N_{alge}, the algorithm proved to be sufficient.

4.2.1 Difference in Number of Equations

As stated in Section 4.1, the number of equations needed to calculate a consistency relation depends on whether `diffalg` or Gröbner basis should be used. Since Gröbner basis only handles algebraic expression, equations have to be differentiated. In this section, a small investigation is made to estimate how many more equations that are necessary in practice when using Gröbner basis.

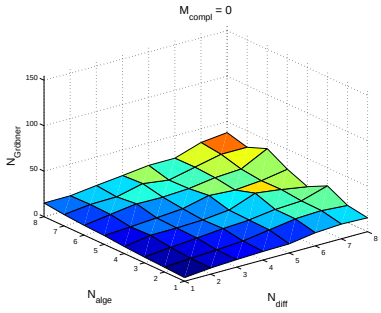
In order to do this, systems where randomized according to the parameters in Section 4.2. To get a quantitative estimation, each set of different parameters where randomized multiple times and a mean was taken to reduce the influence from the random factor. The total number of equations needed for Gröbner basis, $N_{\text{Gröbner}}$, was found using Algorithm 4.1 and is shown in Figure 4.1 together with the maximum bound given by Theorem 4.1.

In the figure, the number of equations needed for Gröbner basis seem to depend on the complexity of the equations involved. The higher the value of M_{compl}, the larger the number of equations needed. Especially in Figure 4.1a and Figure 4.1b there is a significant increase in the number of equations needed when going from the linear case to the nonlinear case. This is quite natural since when differentiating a nonlinear term the number of unknown increases, e.g. when time differentiating x^2 the result is $2x\dot{x}$ and both x and $\dot{x}$ are in the new equation instead of only $\dot{x}$ as would have been the case if the term was linear. It also suggests, not surprisingly, that the more nonlinear an MSO set is, the more difficult it is to solve since it requires more equations.

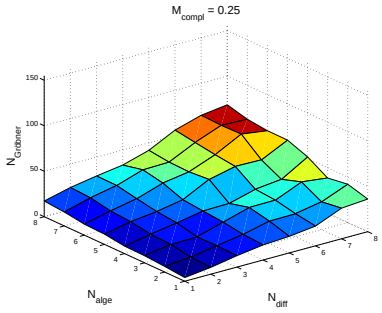
The theoretical upper bound is a worst case scenario and not usually something that the number of equations are close to. This can be seen in Figure 4.1.

4.3 Simulation Results

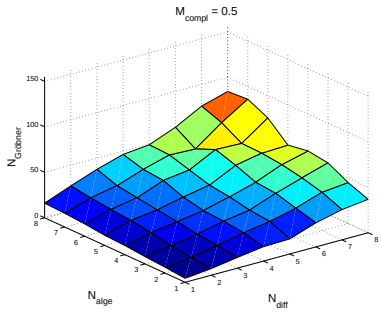
MSO sets were randomized according to Section 4.2 with both N_{diff} and N_{alge} varying from 1 to 5 and M_{compl} varying from 0 to 1. The corresponding MSO sets with x and $\dot{x}$ considered to be separated variables were also calculated using



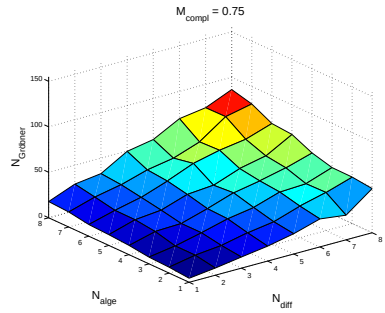
(a) Equations needed for Gröbner basis with $M_{\text{compl}} = 0$.



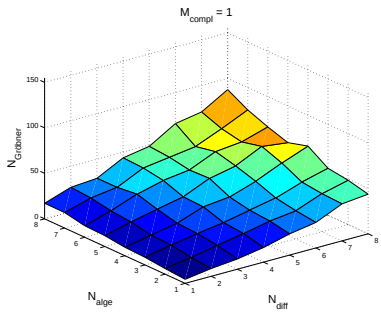
(b) Equations needed for Gröbner basis with $M_{\text{compl}} = 0.25$.



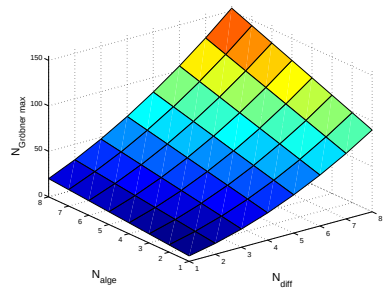
(c) Equations needed for Gröbner basis with $M_{\text{compl}} = 0.5$.



(d) Equations needed for Gröbner basis with $M_{\text{compl}} = 0.75$.



(e) Equations needed for Gröbner basis with $M_{\text{compl}} = 1$.



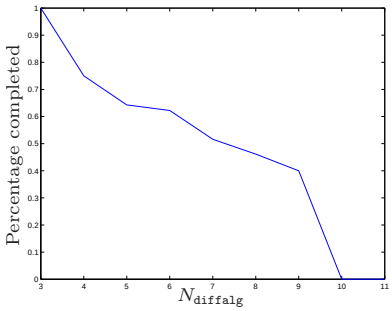
(f) Maximum number of equations according to Theorem 4.1.

Figure 4.1: Number of equations needed for Gröbner basis calculations compared to the number of differential and algebraic equations needed for `diffalg` with different vales on M_{compl}.

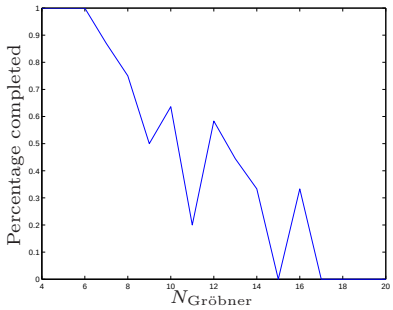
Algorithm 4.1 to be able to use Gröbner basis. After this, consistency relations were calculated using both `diffalg` and the Gröbner basis command and, in the cases where both commands were capable of calculating a consistency relation, the resulting consistency relations were equal. This is as expected since Algorithm 4.1 returns an MSO set of minimal order and the `diffalg` command calculates a triangular basis with minimal order.

From a total of 192 simulations with both Gröbner basis and `diffalg`. In 108 cases, both commands completed the calculation and neither of the two completed in 58 cases. Gröbner basis completed and `diffalg` did not in 9 cases while `diffalg` was the only one to complete in 17 cases.

In Figure 4.2, the percentage of completed `diffalg` calculations and Gröbner basis calculations are shown together with the number of equations needed. The percentage of completed calculations seem do decrease with the number of equations. It should be noted that the variations for Gröbner basis for large number equations can be explained with few simulations with this number of equations and therefore the values fluctuate. It should also be noted that the plot for Gröbner basis has been cut off for values higher than 20. This is because for larger amounts of equations, none of the computations completed.



(a) Percentage of `diffalg` calculations completed and the number of equations in the differential MSO set.



(b) Percentage of Gröbner basis calculations completed and the number of equations in the algebraic MSO set.

Figure 4.2: The number of equations and the percentage of `diffalg` and Gröbner basis calculations that completed.

Figure 4.3 shows the number of equations needed for Gröbner basis calculations and the percentage of those who completed when using `diffalg`. Like in Figure 4.2b, the plot has been cut off for values higher than 20 since none of the computations completed for values of $N_{\text{Gröbner}}$ larger than the values shown in the plot.

The percentage of `diffalg` that completed and the percentage of Gröbner basis that completed compared to the value of M_{compl} is found in Figure 4.4. This shows that the difficulty in eliminating all variables depends on the complexity of the problem.

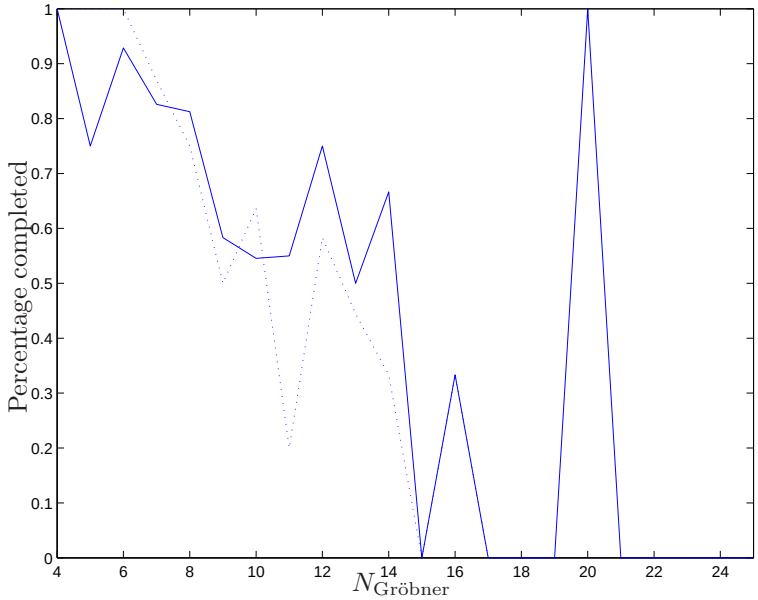
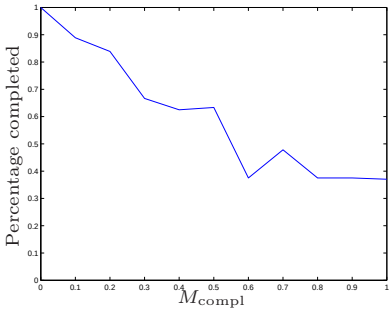
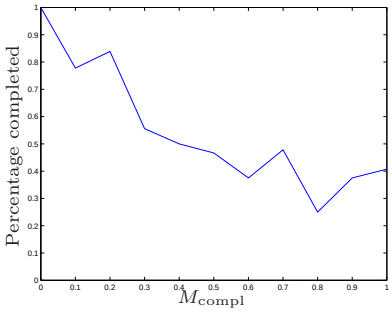


Figure 4.3: Percentage of `diffalg` calculations that completed compared to the number of equations that was needed for Gröbner Basis calculations. For comparisson, the percentage of Gröbner Basis calculations that completed are also plotted as a dotted line.



(a) Percentage of `diffalg` calculations completed and M_{compl}.



(b) Percentage of Gröbner basis calculations completed and M_{compl}.

Figure 4.4: The number of equations and the percentage of `diffalg` and Gröbner basis calculations that completed.

4.4 Conclusions

From the results of the simulation results in Section 4.3, some conclusions can be drawn.

As expected, the more equations an MSO set contains, the more difficult it is to derive a consistency relation. Likewise, the more complex the equations of an MSO set is, the more difficult it is to eliminate the unknown variables. However, it is not possible to draw any clear conclusions about what method is better than the other.

The `diffalg` package is capable of eliminating the unknown variables in more MSO sets than using differentiation and calculating a Gröbner basis, however, the difference is not significant. However, "larger MSO sets", both `diffalg` and Gröbner basis are often incapable of solving the problem. Since Gröbner Basis completes in some cases where `diffalg` does not, an idea could be to try `diffalg` first and if it does not complete, then one could try Gröbner Basis.

Even if it has been decided to use `diffalg` for the calculations of consistency relations, Algorithm 4.1 can still be useful. If the algorithm returns an MSO set where one signal is differentiated several times and it is concluded that this derivative can not be estimated, there is no point in trying to calculate a consistency relation since it will not be realizable. Another point in using Algorithm 4.1 is that if one has several MSO sets from a model, Figure 4.3 suggests that one should start with the MSO set that has the lowest number of equations needed for Gröbner Basis since that calculation is more likely to succeed. Thus, Algorithm 4.1 could be seen as a measure of the complexity of the MSO.

Chapter 5

Results of Differential Algebraic Methods

In this chapter, the results of the approach to calculate consistency relations using `diffalg` for the Scania diesel engine are discussed.

5.1 Polynomials

In Chapter 3 the need for a polynomial engine model was explained. Therefore, all non-polynomial nonlinearities were transformed into polynomials. As a criterion to what model complexity was needed, the fit was evaluated until it was deemed sufficient. The reason for this, and that no other more sophisticated methods were used such as these described in Section 3.3, is that, at the time of the design, it was more of a question whether it was possible to get a "good" representation of the engine model using polynomials. As it later turns out, in Section 5.2, there was no need for a more sophisticated way of designing the model as described in Section 3.3. As an example, the simulated value and the measured value of the pressure in the exhaust manifold, p_{em}, is shown in Figure 5.1. As seen in the figure, the simulated values are a good approximation of the true values.

5.2 The Engine Model

The MSO sets from the engine model was calculated using the algorithm described in [7]. Unfortunately, as shown in the histogram in Figure 5.2, most of the MSO sets contain many equations. Therefore, a straightforward attempt to calculate consistency relations did not give any results[1].

To try the simplest possible case, all polynomial nonlinearities were replaced by $x + 1$ and all constants with 1, but still it was not possible to eliminate all

[1]When attempting to calculate consistency relations using elimination order the memory of the computer ran out and the calculations had to be aborted.

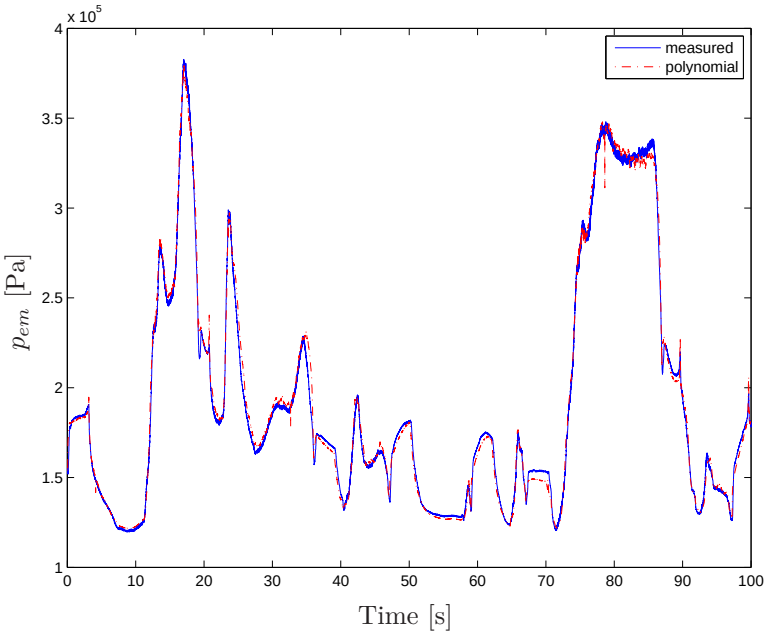


Figure 5.1: The pressure in the exhaust manifold, p_{em}, as measured (solid) values and simulated with polynomials (dash-dotted).

unknown variables. As a result of this, some ways to reduce the computational load was investigated.

5.3 Elimination Order for Gröbner Bases

As mentioned in Chapter 3, computing a differential Gröbner basis is a very computer intense task. One way to try to reduce this is to specify a ”clever” elimination order using structural methods [9]. In [9] an approach investigated was to find variables only present in few equations (preferably in only 2 equations) and extracting these equations and eliminating the unknown from these equations. Finding what variables to eliminate is done by using graph theoretical algorithms. A perhaps more intuitive approach is to find the variables that are the only unknown in one equation and then eliminate these first. The reason for this is that it is assumed that it is easier to eliminate these variables since these variables can be solved for. This can be done by using a *biadjacency matrix* (see Section 2.2) for the set of equations and choosing the variables that appear as a single ”x” in a row of unknowns to eliminate first. After that, that column is set to zeros and the algorithm continues.

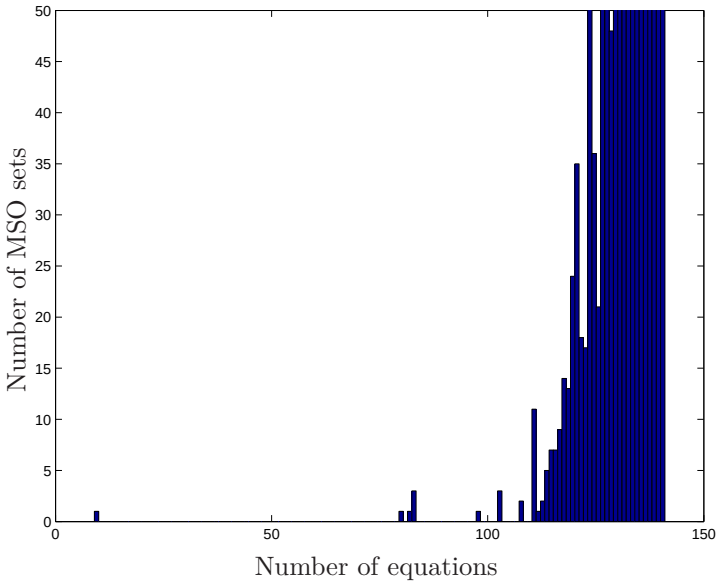


Figure 5.2: Histogram over the number of equations in each MSO set.

Example 5.1

In the MSO set consisting of the linear equations

$$\begin{aligned} e_1 &: & \dot{x}_1 &= x_2 \\ e_2 &: & \dot{x}_2 &= -2x_2 + x_3 + u \\ e_3 &: & \dot{x}_3 &= x_1 + 3u \\ e_4 &: & y &= x_1 \end{aligned}$$

The corresponding biadjacency matrix is then as in Table 5.1a. Here, x_1 would be the first variable to eliminate since it's the only unknown in e_4, after that the biadjacency matrix would be as in Table 5.1b.

Now, both x_2 and x_3 appear as the only unknowns in e_1' and e_3' and the elimination order can be chosen as $x_1 \prec x_2 \prec x_3$ or $x_1 \prec x_3 \prec x_2$.

This method was tried with `diffalg` on the same example as in [9, p. 11], a water tank system with an MSO set with 7 polynomial nonlinear equations and 6 unknowns[2]. Using the algorithm mentioned above gave an elimination order which calculated the same consistency relation as in [9, p. 12] in only a few seconds which was a little faster than letting `diffalg` choose the elimination order using elimination ordering, see Section 3.1.1. This method was tried on the real engine model but did not yield any successful results.

[2]In [9] the consistency relation was calculated using Gröbner bases, and some equations had to be manually differentiated resulting in an MSO set with 11 equations and 10 unknowns.

Equation	Unknown			Known	
	x_1	x_2	x_3	u	y
e_1	x	x			
e_2		x	x	x	
e_3	x		x	x	
e_4	x				x

(a) Incidence matrix for the MSO set in Equation (5.1) .

Equation	Unknown			Known	
	x_1	x_2	x_3	u	y
e_1'		x			x
e_2		x	x	x	
e_3'			x	x	x

(b) Incidence matrix for the MSO set in Equation (5.1) after x_1 is "eliminated" .

Table 5.1: Incidence matrices in Example 5.1.

5.4 "Structure" of Equations

It's not only the number of equations that matters for the computational load. The complexity of the equations also plays an important role. Higher order derivatives, and especially when these appear multiplied together with other terms could make `diffalg` incapable of eliminating the unknown variables in the system. Multiplication of derivatives with other terms in a state space model could for example appear when the derivative and a fraction of two variables are in the same equation, see Section 3.2.1.

Example 5.2

An MSO set with only one unknown is

$$\begin{aligned} \dot{x}(1+x^2)+y &= 0 \\ \ddot{x}^2+x &= 0 \end{aligned} \tag{5.1}$$

These equations were fed into `diffalg` but it was not possible to compute a consistency relation.

As seen in Figure 5.3, one MSO set includes one polynomial nonlinearity, the rest of the MSO sets contain at least 9 polynomials, all but one with a degree of at least 3. Added to that, several other nonlinearities where signals are multiplied and squared exist in the model. These circumstances, together with the large number of equations makes the approach by using `diffalg` nonfeasible for the complete diesel engine.

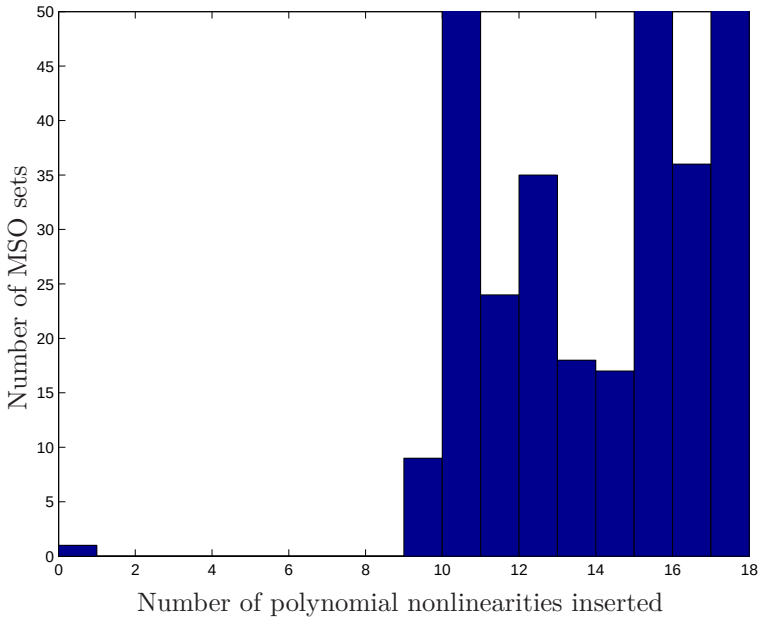


Figure 5.3: Histogram over the number of polynomial nonlinearities in each MSO set.

5.5 One Consistency Relation

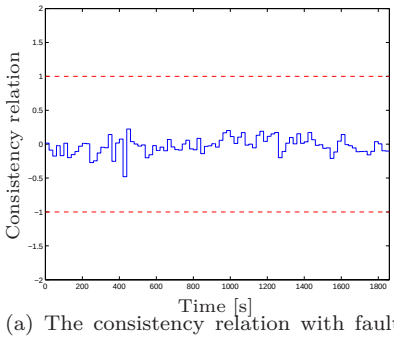
As seen in Figure 5.2, one MSO set did only contain 9 equations and no derivatives of signals. From this MSO set it was possible to eliminate all unknown signals and obtain a consistency relation. The signals included in the consistency relation are listed in Table 5.2. This means the consistency relation should be sensitive to faults in these signals.

Signal name	Description	Unit
n_{trb}	Turbine speed	[r/min]
p_{amb}	Ambient pressure	[Pa]
t_{amb}	Ambient temperature	[K]
w_{cmp}	Air mass flow	[kg/s]

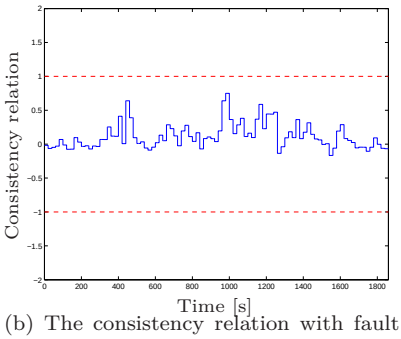
Table 5.2: The signals used in the consistency relation.

To verify that the consistency relation really is consistent with the model, the consistency relation was tested with data from simulations of the model. The consistency relation was also tested with measurement data from a real diesel engine measured during one ETC (European Transient Cycle) to see if the model is a good approximation of the real engine. To reduce the influence of noise, the mean value over 20 seconds was taken. The result is seen in Figure 5.4 where the consistency relations have been normalized so the thresholds could be set to

−1 and 1. The thresholds were set so that the maximal value of the consistency relation with real data were 0.75. The consistency relation with simulated data is scaled in the same way. As seen, the consistency relations are not 0 as they ideally should be. This is mainly because of numerical issues and model errors, both for the data from the real truck since the original model is faulty, but also from the polynomial model since it is approximated from the original SIMULINK-model.



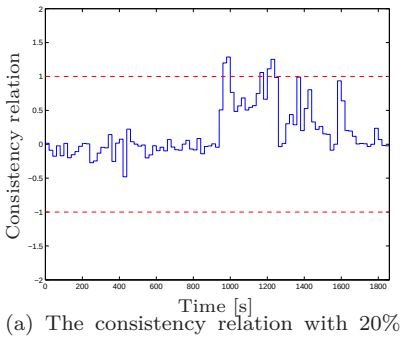
(a) The consistency relation with fault free simulated data.



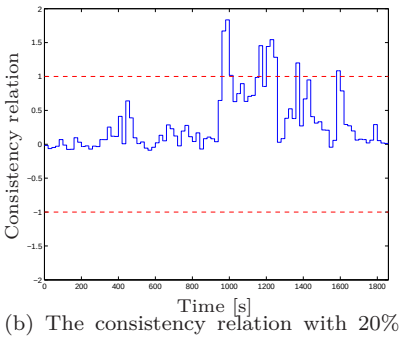
(b) The consistency relation with fault free data from a real engine.

Figure 5.4: The consistency relation with no fault added with simulated and real data. Both consistency relations have been scaled equally much.

The consistency relation should respond to faults in the signals in Table 5.2. This was verified by adding faults and then simulating the consistency relation. In Figure 5.5, a 20% relative error on the pressure sensor was added after half the cycle. As seen in the figure, the consistency relation responds to this fault and goes over the limit and the fault can be detected.



(a) The consistency relation with 20% gain error on p_{amb} with simulated data.



(b) The consistency relation with 20% gain error on p_{amb} with real data.

Figure 5.5: The consistency relation with 20% gain fault on p_{amb} with simulated and real data. Both consistency relations have been scaled equally much.

The consistency relation calculated with simulated data, Figure 5.4a and Figure 5.5a, is much better than the consistency relation calculated with real data, Figure 5.4b and Figure 5.5b. This is expected since the consistency relation with

simulated data has less model errors.

5.6 Complexity of Equations

In order to demonstrate to complexity of the problem to eliminate all variables, some elimination results are presented from one of the MSO sets that were found. Using the simplest possible polynomials, $x + 1$ and all constants set to 1, the MAPLE command `eliminate` can be used on non differential equations. To circumvent that `eliminate` only handle algebraic equations, the differentiated variables were assumed to be known. Like before, nonlinearities such as fractions and multiplication were left unchanged.

Since the equations originate from a SIMULINK-model, most of the equations have the form $x_i = f(x)$, i.e. many variables are easy to eliminate. Starting from 80 equations, eliminating all unknown but 6 results in about half a page of remaining equations. Eliminating one more unknown results in just above one page of remaining equations. The next unknown eliminated results in about two pages of equations. Eliminating one more unknown results in four equations with 3 unknown variables on 46 pages! Elimination of the last terms was not possible.

Chapter 6

Numerical Approaches

In this chapter, some numerical approaches for diagnosis of the diesel engine were investigated.

6.1 Transformation to a Static System

If it is not possible to eliminate all unknown variables, is it possible to do something if almost all unknown variables from an MSO set is eliminated? Consider the case where it is possible to eliminate all variables but one and its time derivative. Then the MSO set would consist of two equations and the unknown variable and its time derivative. Can this be used to detect faults, and if so, is it possible to perform the computations online?

Assume that there would be a static relation with no differentiated variables at all. Then solving for the unknown would be a matter of finding the roots of a polynomial in one variable, which can be a difficult task but not an impossible one, and, if multiple roots are found, see if any of the roots in equation one gives a "small" value when inserted in equation two. Now, what if there's a dynamic part in the equations. Lets say we have two equations, one unknown variable and it's time derivative. Can that be solved? Finding the solutions numerically to a system of two equations with two variables is a more difficult task, especially if it has to be performed online. On top of that, if we find the solutions to the unknown, say x and $\dot{x}$. How do we decide if these are consistent? The next example will demonstrate some problems with this.

Example 6.1

A small time-discrete system is described by

$$x(t+1) = -x(t)^2 + u(t) \tag{6.1a}$$

$$y(t) = x(t) + v(t) \tag{6.1b}$$

where $v(t)$ is measurement noise with standard deviation σ, $u(t)$ is a known actuator signal, $y(t)$ is a known sensor signal and $x(t)$ is an unknown state variable.

This can be seen as a static system with two unknown variables $x(t)$ and $x(t+1)$.

Now, we can solve for the unknown $x(t)$ and $x(t+1)$. Can we then, by using the model, see if these values are consistent? In this case this results in a residual $r(t) = y(t+1) + y(t)^2 - u(t)$ but in a more complex case it may not be possible to calculate this analytically and the roots would have to be found numerically. This system was implemented and v was simulated as white noise with $\sigma = 1$. Because of the noise, the measurement signal was first lowpass filtered. Additionally, the mean value over 20 seconds was taken to smooth out the noise. The system was driven with $u(t)$ as pulses with amplitude 1, and after ≈ 500 seconds was a 10% gain error on the measurement signal $y(t)$ added. This is shown in Figure 6.1. The same result was achieved without low pass filtering, but with a noisier residual.

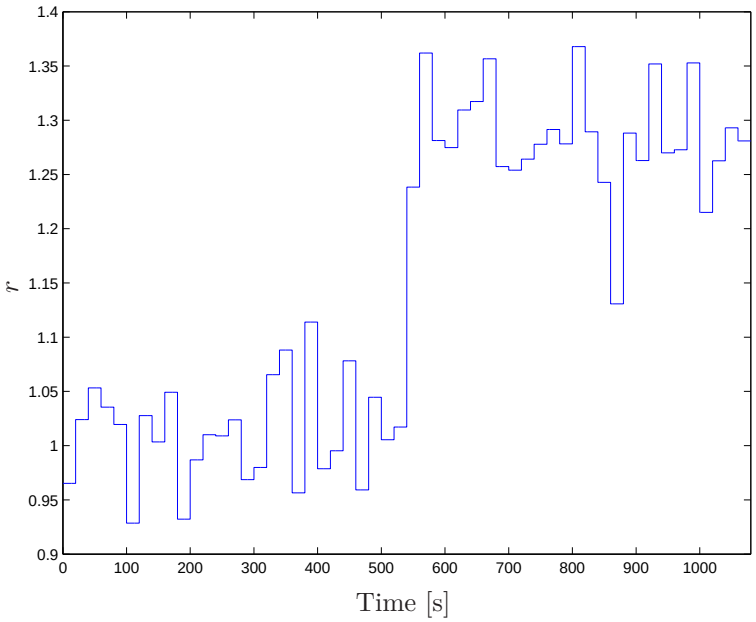


Figure 6.1: Plot of the residual in Example 6.1.

As seen in Figure 6.1 the residual changes level when the fault is introduced. In this case the residual was not equal to 0 though. Some short calculations reveal

why this is not the case.

$$\begin{aligned} E(r) &= E(y(t+1) + y(t)^2 - u(t)) \\ &= E(x(t+1) + v(t+1) + (x(t) + v(t))^2 - u(t)) \\ &= E(\underbrace{x(t+1) + x(t)^2 - u(t)}_{x(t+1)-x(t+1)=0} + v(t+1) + v(t)^2 + 2x(t)v(t)) \\ &= \underbrace{E(v(t+1))}_{=0} + E(v(t)^2 + 2x(t)v(t)) \\ &= \underbrace{E(v(t)^2)}_{\sigma^2} + 2x(t)\underbrace{E(v(t))}_{=0} = \sigma^2 \end{aligned} \tag{6.2}$$

With $\sigma = 1$ the expected value of r equals 1 in the fault free case as seen in Figure 6.1.

In Example 6.1, a time-discrete system was considered for simplicity. If a time-continuous system would have been considered instead, one would have to approximate the time-derivative with a difference quotient and the resulting problem would have looked similar to Equation 6.1a anyway. In the example, the expected value of the residual was σ^2, which is constant. However, consider the same system but with Equation (6.1a) as $x(t+1) = -u(t)x(t)^2$ instead. The same calculations as in Equation (6.2) would give $-u(t)\sigma^2$ as expected value of the residual. With a small system, this is not a problem since the expected value is known, but for a larger system this analysis might not be possible to do analytically. Also, to find roots of a polynomial numerically is something that has to be done with nondeterministic methods that are not feasible for a real-time system. Therefore, this approach was not considered anymore.

6.2 Simulating the System

Another approach is to eliminate as many variables as possible and then simulate the equations we have left, and then compare two simulated values of the same variable. In the general case, we end up with a nonlinear DAE. That is, a general differential-algebraic equation where we have to solve the equation[1] $F(\dot{z}(t), z(t), u(t)) = 0$. The reason that this general form and not semiexplicit form has to be considered is that the rewriting of equations as in Section 3.2.1 and Section 3.2.3 might result in a system that is not on semiexplicit form. Solving this system is normally a difficult task that can't be done analytically and we have to rely on numerical methods. To simulate the system, one have to have a starting value for $z(t_0)$ and a starting value for $\dot{z}(t_0)$. These starting values are normally not known and have to be guessed. Generally, there are several possible solutions to the equation and, if the starting values are guessed wrong, the simulation might

[1] If there are no algebraic equations in F it is called an implicit ODE.

give an undesired solution, even if it is a mathematically correct one. Additionally, to be able to get a correct solution some constraints has to be posed on F. First of all, all of the states has to be stable. The reason for this is because if F is not stable, there is no way for the solution to converge if the starting values are guessed wrong. Also, even if the starting values are guessed correctly, there will always be numerical issues that will make the solution to diverge from the correct one if the states does not converge to a solution. If F is an implicit ODE, the jacobian $\frac{\partial F(\dot{z},z,u)}{\partial \dot{z}}$ is required to be nonsingular for $(\dot{z}(t_0), z(t_0), u(t_0))$ in a neighborhood of the current point $(\dot{z}(t), z(t), u(t))$. This is because of the *implicit function theorem* which states that if the jacobian of a function with respect to a variable is nonsingular for some point $(\dot{z}_0, z_0, u_0)$, then there exist a non–ambiguous solution to the function F in a neighborhood of $(\dot{z}_0, z_0, u_0)$.

The simplest and most intuitive way of simulating an implicit differential equation is to replace the derivative with a difference quotient [11]. A first order approximation of F can be

$$F\left(\frac{z_n - z_{n-1}}{h}, z_n, u(t_n)\right) = 0, \qquad h = t_n - t_{n-1} \tag{6.3}$$

where z_n is the approximation of $z(t_n)$ that the method gives. This is a first order method with constant stepsize. To achieve satisfactory performance, multistep methods must be used, i.e. the derivative has to be approximated with a higher order quotient.

6.2.1 Variable Stepsize

To improve the simulation speed the stepsize can be variable, i.e. when the variables change quickly, smaller steps are taken and vice versa. Normally, it is desirable to take as large steps as possible since that decreases the simulation time, but larger steps increases the error introduced in each step. A simple algorithm will be presented that illustrates the variable stepsize method, and to not complicate things the illustration will be done for a simple one-step Euler for an explicit ODE. Now, suppose the equation we want to simulate is $\dot{y} = f(t, y)$ with the initial condition $y(t_0) = y_0$ is known. The whole idea is to do the step from t_n to $t_n + h$ twice with two different algorithms and then compare the answers, and from that estimate the error. Forward Euler can be seen as the first terms in a Taylor series

$$y(t_n + h) = y(t_n) + \dot{y}(t_n)h + \frac{\ddot{y}(t_n)h^2}{2} + \mathcal{O}(h^3) \tag{6.4}$$

First order Euler includes the first two terms in Equation (6.4) and the introduced error is, if the $\mathcal{O}(h^3)$ term is skipped, proportional to h^2. If the term $\ddot{y}(t_n)$ was known, knowing the error introduced would be easy. The problem is that $\ddot{y}(t_n)$ is not known. Therefore, it has to be estimated. Assume we take one step from t_n to $t_n + h$. The simulated value, A_1 is then

$$A_1 = y_n + hf(t_n, y_n) \tag{6.5}$$

We also know, from Equation (6.4) that

$$A_1 = y_{\text{true}}(t_n + h) + Kh^2 + \mathcal{O}(h^3) \tag{6.6}$$

Now, since K is unknown, it has to be estimated. If f is simulated from t_n to $t_n + h$ in two steps, first to $t_n + \frac{h}{2}$ and then to $t_n + h$, then we would end up with another simulated value on y, let's call that A_2. Now, in the first step the introduced error is $K(\frac{h}{2})^2 + \mathcal{O}(h^3)$ and in the second step $K(\frac{h}{2})^2 + \mathcal{O}(h^3)$ with the same K. Now we have two estimated values on $y(t + h)$, with a total error $A_1 - A_2 = Kh^2 + \mathcal{O}(h^3) - \frac{1}{2}Kh^2 + \mathcal{O}(h^3) = \frac{1}{2}Kh^2 + \mathcal{O}(h^3)$. Since $\mathcal{O}(h^3)$ can be assumed to be small we have a new equation that we can solve for the only unknown K. The relative error can then be decided as $\frac{|A_1 - A_2|}{h} \approx \frac{1}{2}|K|h^2$, and if the absolute error is below some acceptable predefined limit ϵ the algorithm continues for the new time $t_n + h$, possible with a larger h. If the error is too large, the algorithm tries again from time t_n with a new and smaller h.

The above algorithm is very simple, and just included in the thesis to illustrate the principle of variable stepsize methods. In a real problem, one would use more complicated algorithms to simulate an ODE.

6.2.2 Variable Order

There also exist methods who not only have variable stepsize but also variable order on the approximation of the derivatives. In MATLAB one such method is implemented in `ode15i` that can solve implicit ODE's and DAE's of index 1 [22]. Index is a way of classifying DAE's and can be defined as the number of differentiations you need to perform to write the DAE on state space form. For DAE's of index 1 the above approach can be performed with no problems, for DAE's with index higher than 1, this could lead to problems [11]. For DAE's with higher order, other methods exist such as trying to reduce the index by differentiation.

6.2.3 A Small Example

As an example the small system

$$\dot{x} = -\sqrt{x} + u \tag{6.7a}$$

$$y = (-\sqrt{x} + u)x + x + u = \dot{x}x + x + u \tag{6.7b}$$

can, with polynomial representation be written as

$$u^2 - 2\dot{x}u + \dot{x}^2 - x = 0 \tag{6.8a}$$

$$y - \dot{x}x - x - u = 0 \tag{6.8b}$$

Both these equations are implicit ODE's with almost always nonsingular jacobians. The jacobians with respect to $\dot{x}$ are $-2u + 2\dot{x}$ and $-x$ which means that as long as $\dot{x} \neq u$ and $x \neq 0$ there should be no problems simulating the systems. As a test, both equations were simulated and the simulated variable $\hat{x}$ from both

equations were compared and taken as a residual. In Figure 6.2 the residual is seen with faults added as in Table 6.1. As seen, after the effect of the initial conditions, $r \approx 0$ in the fault free case and $r \neq 0$ when faults are added.

Time	Fault	Size of fault
$0 - 200$	No fault	-
$200 - 400$	Actuator gain fault	10%
$400 - 600$	No fault	-
$600 - 800$	Sensor gain fault	15%
$800 - 1200$	No fault	-
$1200 - 1400$	Actuator bias fault	0.3
$1400 - 1600$	No fault	-
$1600 - 1800$	Sensor bias fault	0.3
$1800 - 2000$	No fault	-

Table 6.1: Fault modes.

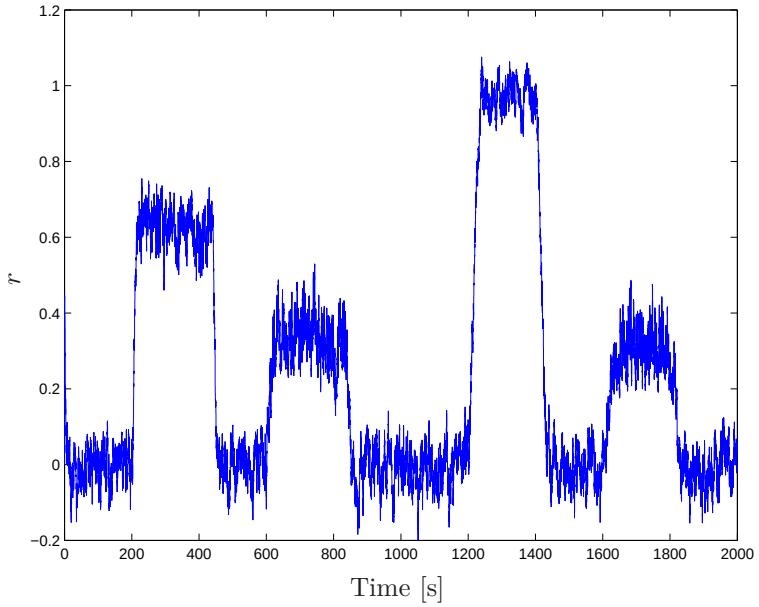


Figure 6.2: Plot of the residual.

Simulating equation systems with variable order and variable stepsize is non-deterministic in time and therefore not suitable for a real-time system. Some other problems occurred when trying to simulate the whole system, see the next section.

6.3 The Engine Model and Numerical Methods

The method suggested in Section 6.2 was tried on an engine model from Scania. For the second smallest MSO set, using polynomials of degree 2, it was possible to eliminate all but 11 unknown variables and hence have 12 equations left. However, due to numerical issues it was not possible to achieve any useful results. As an example, one term in one of the equations contained the ambient pressure p_{amb}, raised to the power of 4 multiplied with the speed of the turbine, n_{trb} raised to the power of 12. That means that the order of those two terms alone is $(10^5)^4(10^4)^{12} = 10^{68}$, which is a very large number. At that magnitude, the numerical precision in MATLAB with double precision is $\approx 10^{52}$, which makes a simulation impossible. Attempts to scale down the equations were made but were not successful. As a comparison, with the case of all constants set to "1" and all polynomial nonlinearities set to $x + 1$, it was possible to eliminate all but 3 unknown variables. The same phenomena as for the "real" model appears here. For example, one of the terms includes the expression $t_{amb}^2 p_{im}^2 p_{amb}^2 n_{trb}^2$ where t_{amb} is the ambient temperature, which is measured in Kelvin, and is of size 10^2, p_{im} is the pressure in the inlet manifold of size 10^5, p_{amb} is the ambient pressure of size 10^5 and n_{trb} is the speed of the turbo of size 10^4. All together this term has the size $(10^2)^2(10^5)^2(10^5)^2(10^4)^2 = 10^{32}$. At this size, double precision gives a precision of $\approx 10^{16}$ which makes a simulation impossible.

It should also be mentioned that since simulating an implicit ODE probably requires variable stepsize this means the computational time is nondeterministic. This is not acceptable in a real-time system like the ECU since deadlines must be held. This makes this method more of a theoretical approach rather than a realistic method for OBD.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, it has been investigated whether some methods from differential algebra can be used in the design of a diagnosis system for a diesel engine at Scania. To be able to use the `diffalg` package, a polynomial model is needed where other nonlinearities such as lookup tables and square roots have to be approximated with polynomials. The results in Section 5.1 showed that it is possible to transform a model with nonpolynomial nonlinearities to a polynomial model successfully.

The evaluation in Chapter 4 suggests that when one faces the task of calculating consistency relations for differential polynomial systems, there is no significant difference between Gröbner basis and `diffalg`. However, for a complex system, both methods may be incapable of solving the problem.

The calculate consistency relations from a diesel engine, the `diffalg` package was used. The results in Section 3.5 showed that for small systems, this method works with good results. However, for larger, more complex systems such as a diesel engine the method was only capable to calculate one consistency relation.

In addition to the differential algebraic methods studied, a few numerical approaches were attempted. These methods did not yield any useful results. However, this is an interesting area which could be investigated further.

7.2 Future Work

The use of `diffalg` did not yield any useful results because of the model complexity. To lower the computational complexity, the idea in [15], where cascaded connected subsystems were found and each subsystem was treated separately could be investigated.

In [13], the existing OBD-system used at Scania and the automatic generated diagnosis system developed in [7, 16, 6] are evaluated. In [13] it is concluded that

some of the signals in the automatic generated consistency relations do not affect the consistency relations when a fault has occurred in these signals. This implies that some parts of the model of the Scania engine might be simplified without the loss of too much information. With a simplified model, the methods used in this thesis might be usable.

Another interesting investigation could be to use more complex numerical approaches, for example *particle filter* to detect faults.

Bibliography

[1] M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[2] F. Boulier and F. Lemaire. Computing canonical representatives of regular differential ideals. *International Symposium on Symbolic and Algebraic Computation. Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation*, pages 38 – 47, 2000.

[3] François Boulier. *Étude et implantation de quelques algorithmes en algèbre différentielle*. PhD thesis, Université de Lille I, 1994.

[4] B. Buchberger. Groebner Bases: A Short Introduction for Systems Theorists. In R. Moreno-Diaz, B. Buchberger, and J.L. Freire, editors, *Proceedings of EUROCAST 2001 (8th International Conference on Computer Aided Systems Theory - Formal Methods and Tools for Computer Science)*, Lecture Notes in Computer Science 2178. Las Palmas de Gran Canaria, Copyright: Springer - Verlag Berlin, 19-23 February 2001.

[5] B. Buchberger. Groebner Bases and Systems Theory. *Multidimensional Systems and Signal Processing, Special Issue on "Applications of Groebner Bases in Multidimensional Systems and Signal Processing"*, 12(3/4):223–253, October 2001. Z. Lin, L. Xu (eds.).

[6] Henrik Einarsson and Gustav Arrhenius. Automatic design of diagnosis systems using consistency based residuals. Master's thesis, Uppsala University, Mars 2005.

[7] Lars Eriksson. Structural algorithms for diagnostic system design using simulink models. Master's thesis LiTH-ISY-EX-3601-2004, Department of Electrical Engineering, Linköpings universitet, January 2004.

[8] Erik Frisk. *Residual Generation for Fault Diagnosis*. PhD thesis, Linköpings Universitet, November 2001.

[9] Erik Frisk. Efficient elimination orders for the elimination problem in diagnosis. Technical Report LiTH-R-2532, Department of Electrical Engineering, Linköpings Universitet, SE-581 83 Linköping, Sweden, 2003.

[10] Erik Frisk and Jan Åslund. Lowering orders of derivatives in non-linear residual generation using realization theory. *Automatica*, 41(10):1799–1807, 2005.

[11] Torkel Glad and Lennart Ljung. *Modellbygge och simulering*. Studentlitteratur, 2004.

[12] Oleg Golubitsky, Marina Kondratieva, Marc Moreno Maza, and Alexey Ovchinnikov. Bounds for algorithms in differential algebra. *Journal of Symbolic Computation*, Submitted to.

[13] Joakim Hansen and Jens Molin. Design and evaluation of an automatic designed diagnosis system. Master's thesis, Linköpings universitet, December 2006.

[14] Evelyne Hubert. The `diffalg` package. `http://www-sop.inria.fr/cafe/Evelyne.Hubert/diffalg/`. This is an electronic document. Date retrieved: December 11, 2006.

[15] C.S. Kallesøe, V. Cocquempot, and R. Izadi-Zamanabadi. Model based fault detection in a centrifugal pump application. *IEEE Transactions on Control Systems Technology*, 14(2):204 – 15, 2006.

[16] Kristian Krigsman and John Nilsson. Code generation for efficient real-time execution of diagnostic residual generators. Master's thesis, Royal Institute of Technology, December 2004.

[17] Mattias Krysander. *Design and Analysis of Diagnostic Systems Utilizing Structural Methods*. Licenciate thesis, Linköpings Universitet, 2003. LiU-TEK-LIC-2003:37, Thesis No. 1038.

[18] Mattias Krysander. *Design and Analysis of Diagnosis Systems Using Structural Methods*. PhD thesis, Linköpings universitet, June 2006.

[19] Elizabeth Mansfield. *Differential Gröbner Bases*. PhD thesis, University of Sydney, 1991.

[20] Mattias Nyberg and Erik Frisk. *Model Based Diagnosis of Technical Processes*. 2006.

[21] K. Rodriguez-Vazquez and P. Fleming. A genetic programming/narmax approach to nonlinear system identification. In Ali Zalzala, editor, *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, University of Strathclyde, Glasgow, UK, 1-4 September 1997. Institution of Electrical Engineers.

[22] L.F. Shampine. Solving $0 = F(t, y(t), y'(t))$ in MATLAB. *Journal of Numerical Mathematics*, 10(4):291 – 310, 2002.

[23] Sofi Stenström. Differential Gröbner bases. Master's thesis ISRN LTU-EX–02/073–SE, Department of Mathematics, Luleå Universitet, February 2002.

Appendix A

Notation and Abbreviations

Notation

$\prec$	Preceeds
$\forall$	For all
$\dot{x}$	x differentiated with respect to time
X	The set of unknown variables with x and $\dot{x}$ considered to be the same variable
$\bar{X}$	The set of unknown variables with x and $\dot{x}$ considered to be different variables
Z	Set of known variables
E	Set of equations
$\|X\|$	Number of items in the set X
$\text{var}_X(E)$	The set of variables X that are part of the equations E
M^+	Overdetermined part of M
σ	Standard deviation

Abbreviations

DAE	Differential-algebraic equation
DSSL	Differentiated-separated structural-model
DSSM	Differentiated-lumped structural-model
ECU	Engine control unit
ETC	European transient cycle
MSO	Minimally structurally overdetermined
OBD	On-board diagnostics
ODE	Ordinary differential equation