Symbols speak louder than numbers
analysis and design of nonlinear control systems with the symbolic computation system MAPLE

van Essen, H.A.
Award date:
1992

Link to publication
Symbols speak louder than Numbers

Analysis and Design of Nonlinear Control Systems
with the Symbolic Computation System MAPLE

WF 92.061
Harm van Essen
Symbols speak louder than numbers

Analysis and Design of Nonlinear Control Systems
with the Symbolic Computation System MAPLE

Harm van Essen


Eindhoven University of Technology (TUE)
Department of Mechanical Engineering
Division of Mechanical Engineering Fundamentals (WFW)

Coach: Ir. A.G. de Jager (TUE)
Chair: Prof. dr. ir. J.J. Kok (TUE)

Report No. WFW 92.061
Summary

This thesis makes a contribution to the development of systematic design methodologies for nonlinear control systems. During the last decade important progress has been made in the development of such systematic design methodologies for nonlinear control systems. This contribution exists of the implementation of some recently developed analytical analysis and design strategies in a symbolic computation environment. A specific choice for the symbolic computation system MAPLE has been made.

For highly nonlinear systems, linear controllers -obtained from a nonlinear system linearized in a working point- will often not be adequate. Controller design strategies based directly on the nonlinear system model, can be expected to provide significantly improved performance. Some of the most promising controller techniques are based on the exact linearization theory. Besides this theory, some more design problems can be formulated, like stabilization, tracking and disturbance attenuation. Regarding to these control objectives, the notion of the zero dynamics is a very important property of nonlinear (control) systems.

In the literature several methods are proposed for the exact (analytical) solutions of these analysis and design problems. An overview of the methods is presented in the book *Nonlinear Control Systems, an Introduction* ([I]). The theories in this book form the basis of this thesis. For small scale test problems these methods require already a fair amount of computation. For industrial scale problems these methods can only be applied when the computations are assisted by Symbolic Computation (SC) programs.

In this thesis an overview of theoretical results concerning the analysis of nonlinear control systems is presented. This thesis is restricted to an extensive and consistent treatment of the zero dynamics and the exact linearization theories. The relative degree, the normal form, the zero dynamics, and exact linearization are treated extensively. The emphasis is on the acquisition of algorithms to compute essential information for the analysis. Further, the symbolic computation system MAPLE is discussed and the implementation of the algorithms in MAPLE procedures is treated. Also examples, descriptions, flowcharts and listings of all procedures are included as a kind of manual.

In this thesis several procedures are developed, which offer powerful computational tools. The procedures together, form a consistent analysis of exact linearization and zero dynamics. The results established so far with these procedures, seem to be very promising, though their value should be proved in practice, when elaborate tests with realistic systems are performed. Following from the successful implementation of the abstract algorithms in the Symbolic system MAPLE, this program seems to be a suitable environment for the symbolic application and development of analytic control strategies. Moreover, symbolic computation is found to be a powerful research tool, which importance for science and educational purposes is stressed.
# Contents

Summary  

Contents  

Notations  

1. State space description  
2. Differential operations  
3. Change of coordinates in the state space  
4. Distributions

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Introduction</th>
<th>Analysis and Design</th>
<th>Exact Linearization</th>
<th>Asymptotic Stabilization and other design problems</th>
<th>Zero Dynamics</th>
<th>Symbolic Computation</th>
<th>Goals of this thesis</th>
<th>Link with other research</th>
<th>View of contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>MAPLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Nonlinear Control Systems</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Introduction</th>
<th>I The relative degree</th>
<th>II The normal form</th>
<th>III The zero dynamics</th>
<th>IV Exact linearization of the input-state equations</th>
<th>V Exact linearization of the input-output response</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>14</td>
<td>18</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>MAPLE Implementations</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Classes in which the procedures can be ordered</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Discussion of the implementations</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Discussion of the use of MAPLE</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>Conclusions and Recommendations</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conclusions</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Recommendations</td>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>References</th>
<th>37</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix A</th>
<th>Description of the procedures</th>
<th>A.1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix B</th>
<th>Examples</th>
<th>B.1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix C</th>
<th>Flowcharts</th>
<th>C.1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Appendix D</th>
<th>Listings of the procedures</th>
<th>D.1</th>
</tr>
</thead>
</table>
Notation and Terminology

1. State space description

Throughout this thesis a multivariable nonlinear control system with \( m \) inputs \( u_1, \ldots, u_m \) and \( p \) outputs \( y_1, \ldots, y_p \) is considered. In state space form this system is given by the following set of equations:

\[
\begin{align*}
\dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x) u_i \\
y_i &= h_i(x) \quad \forall 1 \leq i \leq p
\end{align*}
\]

Where the mappings \( f, g_1, \ldots, g_m \) and the functions \( h_1, \ldots, h_p \) are smooth in their arguments, i.e. that all entries are real-valued functions of \( x_1, \ldots, x_n \) with continuous partial derivatives of any order.

However, when convenient, these equations will be written in a more condensed form:

\[
\begin{align*}
\dot{x} &= f(x) + g(x) u \\
y &= h(x)
\end{align*}
\]

Having set:

\[
\begin{align*}
u &= \text{col}(u_1 \ldots u_m) \\
y &= \text{col}(y_1 \ldots y_p) \\
g(x) &= [g_1(x) \ldots g_m(x)] \\
h(x) &= \text{col}(h_1(x) \ldots h_p(x))
\end{align*}
\]

Where \( g(x) \) and \( h(x) \) are respectively the \( n \times m \) input matrix and the \( p \)-dimensional output vector.

SISO systems fit easily in this general description. It is the case where \( m=1 \) and \( p=1 \). Also linear systems are included in which case \( f(x) \) and \( h(x) \) are linear (vector) functions of \( x \), and \( g_i \) are (vector) functions of real numbers.

As a state space for this model, a subspace \( U \) of \( \mathbb{R}^n \) is considered rather than \( \mathbb{R}^n \) itself. This limitation may correspond either to a constraint established by the equations themselves (when not all solutions or trajectories are allowed) or to a constraint imposed by the input (in order to avoid some kind of singularities at some points in the state space). Of course in many cases it is allowed to set \( U = \mathbb{R}^n \).

The mappings \( f, g_1, \ldots, g_m \) assign to each point \( x \) of \( U \) a vector of \( \mathbb{R}^n \), namely \( f(x), g_1(x), \ldots, g_m(x) \). For this reason they are referred to as smooth vector fields defined on \( U \). In many
instances it is convenient to define also smooth covector fields defined on a subset $U$ of $\mathbb{R}^n$. It is quite natural to identify such covector fields as row vectors of smooth functions of $x$.

2. Differential operations

In this thesis several types of differential operations involving vector fields are used. These operations are frequently used in the analysis of nonlinear control systems and are described by:

- The **differential** or **gradient** of a real-valued function $\lambda$ (defined on an open subset $U$ of $\mathbb{R}^n$) is a covector field denoted $d\lambda$ and defined like:

\[
\frac{d\lambda}{dx} = \left[ \frac{\partial \lambda}{\partial x_1}, \frac{\partial \lambda}{\partial x_2}, \ldots, \frac{\partial \lambda}{\partial x_n} \right]
\]  

(N4)

- The **Jacobian** of a vector field $f$ is a matrix defined as:

\[
\frac{df}{dx} = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \ldots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1} & \ldots & \frac{\partial f_n}{\partial x_n}
\end{bmatrix}
\]  

(N5)

- The **directional derivative** of $\lambda$ along $f$ or **Lie derivative**, often written as: $L_f \lambda(x)$. It involves a real-valued function $\lambda$ and a vector field $f$, both defined on an open subset $U$ of $\mathbb{R}^n$. From these a new smooth real-valued function is defined, whose value at each $x$ of $U$ is equal to the inner product:

\[
L_f \lambda(x) = \langle \frac{d\lambda}{dx}, f(x) \rangle = \frac{\partial \lambda}{\partial x_1} f_1(x) + \cdots + \frac{\partial \lambda}{\partial x_n} f_n(x)
\]  

(N6)

repeated use of this operation is defined by the following recursion:

\[
L_f^{k+1} \lambda(x) = \frac{\partial (L_f^{k} \lambda(x))}{\partial x} f(x)
\]

(N7)

If $\gamma$ denotes a $s \times 1$ vector of real-valued functions, then also the directional derivative along a vector field $f$ is a $s \times 1$ vector of real-valued functions:

\[
L_f \gamma(x) = \begin{bmatrix}
L_f \gamma_1(x) \\
\vdots \\
L_f \gamma_s(x)
\end{bmatrix}
\]  

(N8)

For the sake of notational compactness, sometimes the following notation will be used. Let $\gamma$ be a $s \times 1$ vector of smooth functions and $\{g_1, \ldots, g_m\}$ a set of vector fields. Now let $L_{g\gamma}$ denote the $s \times m$ matrix
The Lie product or Lie bracket of \( f \) and \( g \), this type of differential operation involves two vector fields \( f \) and \( g \), both defined on an open subset \( U \) of \( \mathbb{R}^n \). From these a new smooth vector field is constructed, noted \([f,g]\), whose value at each \( x \) of \( U \) is defined by:

\[
[f,g](x) = \left. \frac{\partial g}{\partial x} \right|_x - \left. \frac{\partial f}{\partial x} \right|_x \tag{N10}
\]

Note that

\[
\left. \frac{\partial f}{\partial x} \right|_x, \left. \frac{\partial g}{\partial x} \right|_x \tag{N11}
\]

denote the \( n \times n \) Jacobian matrices of the mappings \( f \) and \( g \).

Repeated use of this operation on a vector field \( g \) together with the same vector field \( f \) is possible. In order to avoid a monstrous notation like \([f, [f, ...,[f,g]]]\) a recursive operation is defined:

\[
\begin{align*}
\text{ad}_f^g(x) &= [f, \text{ad}_f^{g-1} g](x) \\
\text{ad}_f^g(x) &= g(x)
\end{align*}
\tag{N12}
\]

3. Change of coordinates in the state space

A nonlinear change of coordinates can be described in the form:

\[
z = \Phi(x) \quad \text{with:} \quad \Phi(x) = \begin{bmatrix}
\Phi_1(x) \\
\Phi_2(x) \\
\vdots \\
\Phi_n(x)
\end{bmatrix} \tag{N13}
\]

If

(i) \( \Phi(x) \) is invertible, i.e. there exists a function \( \Phi^{-1}(z) \) such that \( \Phi^{-1}(\Phi(x)) = x \quad \forall x \in \mathbb{R}^n \)

(ii) \( \Phi(x) \) and \( \Phi^{-1}(z) \) are both smooth mappings, i.e. have continuous partial derivatives of any order,

then a transformation of this type is called a global diffeomorphism on \( \mathbb{R}^n \). However, in most cases these properties are defined and checked only in the neighbourhood of a given point. A transformation of this type is then called a local diffeomorphism.

It is very easy to test whether a smooth function \( \Phi(x) \), defined on some subset \( U \) of \( \mathbb{R}^n \) is a local diffeomorphism or not: If the Jacobian matrix of \( \Phi \) is nonsingular at a point \( x = x^0 \), then \( \Phi(x) \) is a local diffeomorphism, defined on a suitable open subset \( U^0 \) of \( U \), containing \( x^0 \).

The effect of a change of coordinates on the description of a nonlinear system is analyzed in the following way:
Notation and Terminology

\[ z(t) = \Phi(x(t)) \tag{N15} \]

Expressing

\[ x(t) = \Phi^{-1}(z(t)) \tag{N16} \]

the equations in the new coordinates are obtained:

\[ \dot{z}(t) = \tilde{f}(z(t)) + \tilde{g}(z(t))u(t) \tag{N17} \]

4. Distributions

Suppose a number \( d \) of \( n \) dimensional smooth vector fields \( f_1, \ldots, f_d \) are given, all defined on the same open set \( U \). At all points \( x \) in \( U \), the vectors \( f_1(x), \ldots, f_d(x) \) span a vector space (a subspace of \( \mathbb{R}^n \)). This vector space can be denoted by

\[ \Delta(x) = \text{span}\{f_1(x), \ldots, f_d(x)\} \tag{N18} \]

In doing this, a vector space is assigned to each point \( x \) of the set \( U \). Moreover, because the vector fields are smooth, also the assignment can be regarded as a smooth one. The assignment of the subspace spanned by the values at \( x \) of some smooth vector fields defined on \( U \), is called a smooth distribution. For the assignment as a whole, the distribution is denoted

\[ \Delta = \text{span}\{f_1, \ldots, f_d\} \tag{N19} \]

where (N18) denotes the value of the distribution at a point \( x \).

We say that a distribution \( \Delta_1 \) contains a distribution \( \Delta_2 \), and is written \( \Delta_1 \supset \Delta_2 \), if \( \Delta_1(x) \supset \Delta_2(x) \) for all \( x \). Also, a vector field \( f \) belongs to a distribution \( \Delta \), and is written \( f \in \Delta \), if \( f(x) \in \Delta(x) \) for all \( x \).

In the subsequent analysis the notion of smooth distribution is of fundamental importance. Moreover, a lot of important concepts associated with the notion of distribution can be formulated, mostly being an extension from elementary concepts concerning vector spaces. The following ones will be frequently used in the next chapters.

**dimension**

The dimension of a distribution at a point \( x \) is the dimension of the subspace \( \Delta(x) \).

**singularity and regular points**

A distribution \( \Delta \), defined on the open set \( U \) is nonsingular if there exists an integer \( d \) such that

\[ \dim(\Delta(x)) = d \quad \forall \ x \in U \tag{N20} \]

A point \( x^0 \) in \( U \) is called a regular point of the distribution \( \Delta \) if \( \Delta \) is nonsingular on a neighbourhood \( U^0 \) of \( x^0 \) in \( U \).
If \( x^0 \) is a regular point of the smooth distribution \( \Delta \) and \( \dim(\Delta(x^0)) = d \) then (II)(lemma 1.3.2)) there exist a set of \( d \) smooth vector fields \( \{f_1, \ldots, f_d\} \) defined on \( U \) and a neighbourhood \( U^0 \) of \( x^0 \) with the properties:

(i) the vectors \( f_1(x), \ldots, f_d(x) \) are linearly independent,
(ii) \( \Delta(x) = \operatorname{span}\{f_1(x), \ldots, f_d(x)\} \),

at each \( x \) in \( U^0 \).

**involutivity**

A distribution \( \Delta \) is said to be involutive, if the Lie bracket \([\tau_1, \tau_2]\) of any pair of vector fields \( \tau_1 \) and \( \tau_2 \) belonging to \( \Delta \) is a vector field which belongs to \( \Delta \), i.e. if

\[
\tau_1 \in \Delta, \tau_2 \in \Delta \Rightarrow [\tau_1, \tau_2] \in \Delta
\]

(N21)

Sometimes, starting from a distribution \( \Delta \) which is not involutive it is useful to construct an appropriate involutive distribution. The smallest involutive distribution containing \( \Delta \) is called the involutive closure of \( \Delta \) and is denoted by \( \operatorname{inv}(\Delta) \). Note that every 1-dimensional distribution is involutive because \([\tau_1, \tau_1] = 0 \in \Delta \).

**codistributions**

Completely analogous to the equivalence between vector fields and covector fields, it is also convenient to consider codistributions. A smooth codistribution is defined as the assignment of the subspace spanned by the values at \( x \) of some smooth covector fields defined on \( U \). Suppose \( d \) \( n \) dimensional smooth covector fields \( w_1, \ldots, w_d \) are given, all defined on the same open set \( U \). For these covectors, a codistribution is denoted

\[
\Omega = \operatorname{span}\{w_1, \ldots, w_d\}
\]

**annihilator**

An important concept in the relation between distribution and codistribution is that of the annihilator. The annihilator of a distribution \( \Delta \) is the set of all covectors which annihilate all vectors in \( \Delta(x) \). So the annihilator of a distribution is a codistribution. The annihilator of a distribution \( \Delta \) is noted \( \Delta^\perp \). Analogous, the annihilator of a codistribution is a distribution, spanned by the set of all vectors which annihilate all covectors of \( \Omega \). The annihilator of \( \Omega \) is denoted \( \Omega^\perp \). The sum of the dimensions of \( \Delta \) and \( \Delta^\perp \) (and also \( \Omega \) and \( \Omega^\perp \)) is exactly equal to \( n \), the dimension of the state space.

**interpretation of a matrix as a distribution v.v.**

It is possible to interpret a matrix \( F \) with \( n \) rows, which entries are smooth functions of \( x \) as follows:

- the columns of \( F \) can be considered as smooth \( n \)-dimensional vector fields,
- the matrix can be regarded as a smooth distribution, spanned by its columns,
- the value of this distribution at each point \( x \) is equal to the image of the matrix \( F(x) \) at this point

\[
\Delta(x) = \operatorname{Im}(F(x))
\]

- the dimension of this distribution at a point \( x^0 \) is equal to the rank of \( F(x^0) \),
the annihilator of the distribution $\Delta$ is a codistribution spanned by a set of row vectors $w$ satisfying
\[ wF(x) = 0 \]
(N24)
So the annihilator $\Delta^\perp$ is the kernel of the transposed matrix $F(x)$:
\[ \Delta^\perp = \text{kernel}(F(x)^T) \]
(N25)

Naturally, considering a matrix $W(x)$ with $n$ columns whose entries are smooth functions of $x$
- the rows of $W$ can be regarded as smooth $n$-dimensional covector fields,
- the matrix can be considered as a smooth codistribution, spanned by its rows.

etc.

**complete integrability**

A nonsingular $d$ dimensional distribution $\Delta$, defined on $U$, is said to be completely integrable if, for each point $x^0$ of $U$, there exist $n-d$ real-valued functions $\lambda_i$ defined on a neighbourhood $U^0$ of $x^0$, such that:
\[ \text{span}\{d\lambda_1, \ldots, d\lambda_{n-d}\} = \Delta^\perp \quad \text{on} \quad U^0 \]
(N26)

The *Frobenius theorem* says that a nonsingular distribution is completely integrable if and only if it is involutive.
Chapter 1 Introduction

Analysis and Design

Analysis for nonlinear systems means mostly simulation, i.e. numerically solving the nonlinear differential equations describing the system. Design of control systems is often done for the linearized system. The linearized model can be obtained in a variety of ways, including Jacobian linearization via Taylor series expansion of a nonlinear model at a nominal working point. Standard methods for analysis and design of control systems are based on linear algebra packages like LAPACK, or interactive programs like MATLAB and MATRIX-X. So, although most systems are inherently nonlinear, the vast majority of controller design techniques for control are based on linear system models. These linear control strategies, such as LQG and adaptive control usually provide adequate performance if the system is sufficiently linear in the region of the working point. However, if the system is essential nonlinear or deviates significantly from the working point at which the nonlinear model was linearized, this strategy may not be adequate: To ensure stability for a wider range of operating conditions, linear controllers must be significantly detuned and the performance is often severely degraded. In cases a high performance - e.g. fast and accurate robot control, critical process control etc. - is required, linear controllers - obtained from a nonlinear system linearized in a working point - will not be adequate.

Exact Linearization

For highly nonlinear systems, controller design strategies based directly on nonlinear system models can be expected to provide significantly improved performance. During the last decade important progress has been made in the development of such systematic design methodologies for nonlinear control systems. An overall view is presented by Isidori in his book Nonlinear Control Systems, an Introduction ([I]). The theories in this book form the basis of this thesis.

Some of the most promising controller techniques are based on exact linearization theory. Unlike Jacobian linearization, which linearizes the system only at the nominal working point, these methods employ nonlinear transformations and feedbacks that provide exact linearization of the model in the neighbourhood of the nominal working point. The nonlinear model can be linearized - under some rather strict conditions - in either an input-state or in an input-output point of view. Once a linear model is available, linear controller design techniques can be employed in order to satisfy additional control objectives.
Asymptotic Stabilization and other design problems

Besides the exact linearization theory, some more design problems for nonlinear control systems can be formulated. Asymptotic stabilization via smooth feedback is one of the most interesting. In fact, feedback stabilization is important as a preliminary step in achieving results for remaining control problems as (asymptotic) tracking and disturbance attenuation. A very important property of nonlinear (control) systems regarding to these design problems, is the notion of the zero dynamics. The concept of zero dynamics is ought to be the nonlinear analog of the notion of (transmission) zeros of a transfer function (matrix) of a linear system. From this analogy several important results regarding stability of nonlinear control systems can be derived. In the next section such a result will be presented, but for the sake of good order, first the zero dynamics are discussed.

Zero Dynamics

The concept of zero dynamics can be approached in two different manners, both satisfying the analogy with the (transmission) zeros of a corresponding linear system. First, the zero dynamics can be regarded as the internal dynamics imposed on the system when the constraint that the output is identically zero is enforced by an appropriate choice of input and initial state conditions. Indeed, the original definition and terminology of the zero dynamics was that, in the case of an invertible square minimal linear system, these dynamics identify a unique linear autonomous dynamical system whose natural frequencies are exactly the transmission zeros of the system transfer function matrix. The other way to approach the notion of zero dynamics is to suppose a nonlinear system rendered maximally unobservable via feedback. Here, the zero dynamics can be considered as the internal dynamics of the unobservable component of the closed-loop system. In fact these dynamics are the nonlinear analog of the "numerator dynamics" of a linear system. Again, in case of a (minimal and invertible) linear system, the corresponding natural frequencies are the transmission zeros of the system.

The two approaches to the important concept of zero dynamics summarized above, are not equivalent for any nonlinear system. There are, however, two noticeable cases in which this equivalence occurs (I2): The case of an arbitrary linear system (regarding to the definition) and also the case of those nonlinear systems for which a relative degree can be defined. The class of systems for which a relative degree can be defined is exactly equal to the class of systems which can be rendered noninteractive by means of a regular static state feedback (I1)prop.4.3.2. From this follows that also the notion of the relative degree will be an important property of nonlinear systems in the present analysis.

Referring to the important results regarding stability of nonlinear systems, which can be derived from the zero dynamics, it is stressed that the existence of an output map whose corresponding zero dynamics is asymptotically stable, is a sufficient and necessary condition for the existence of smooth locally stabilizing state feedback laws (I2). This can intuitively be understood when the zero dynamics are regarded as the unobservable component of the system. If this unobservable part is not stable, stabilizing feedbacks cannot affect the instabilities at all and will not exist. In fact a "time-bomb" is present in the system.

Symbolic Computation

In the literature several methods are proposed for the exact (analytical) solutions of analysis and design problems formulated in the previous (I1). For small scale test problems these methods require already a fair amount of computation. For industrial scale problems these methods can
only be applied when the computations are assisted by Symbolic Computation (SC) programs. This is of particular (scientific) interest because numerical solutions often gives only an approximation of the properties of the system (along a trajectory or in a working point).

At the moment, there is little (industrial) experience with these new controller design strategies due to this computational problems and the fact that they were just recently developed. However, results established sofar in mechanical ([1],[2],[7],[8]) and process control ([3]), seems to be promising, though many problems are yet to be solved. More research is required, in which more realistic models should be used and experimental studies should be conducted. It is expected that the introduction of symbolic computation will be of great consequence to this future research.

Goals of this thesis

This thesis makes a contribution to the development of systematic design methodologies for nonlinear control systems. This contribution exists of the implementation of some recently developed analytical analysis and design strategies in a symbolic computation environment.

In doing this, some specific choices have been made. First, the symbolic computation system MAPLE is used throughout this research. Second, this thesis is restricted to an extensive and consistent treatment of the zero dynamics and the exact linearization theories. This means in particular that stabilization and connected control objectives like tracking and attenuation will not be discussed. However, some general properties of nonlinear systems -necessary in the analyses in this thesis- like the relative degree, the "normal form", certain transformations and several computational tools, will be useful in a broader context of analysis.

According to these choices, also the goal of this thesis is twofold:

- To investigate the possibilities of a modern symbolic computation program like MAPLE for research and educational purposes, and, more specific, to investigate the application of this program to the development and design of (nonlinear) control systems.

- To offer computational tools which provide possibilities for analysis and design of nonlinear control systems and will be useful in further research to employment and development of these nonlinear control strategies.

Link with other research

Only for a few years, general purpose Symbolic Computation systems are powerful enough to be able to implement complex analytic algorithms. Application of symbolic computation in development of (nonlinear) control systems, was and is often restricted to ad-hoc solutions of problems obtained with the aid of symbolic computation.

To my knowledge, only few research is addressed in order to develop general packages for analysis and design of arbitrary nonlinear systems. Birk and Zeitz ([10]), are developing a package with extensive possibilities in the Macsyma environment. Akhrif and Blankenship ([11]), are known to be engaged in application of symbolic computation in nonlinear control systems.
View of contents

In a previous part of this thesis, a survey of notation and terminology that will be used in the other chapters is presented. These notes include state space description, differential operators, coordinate transformations and distributions in differential geometry. In Chapter 2 the symbolic computation program MAPLE is discussed. Further, in Chapter 3 theoretical results concerning analysis and design strategies of nonlinear control systems are presented. The relative degree, normal form, zero dynamics, the input-state, and input-output variant of exact linearization are discussed. Whenever convenient, formulas and algorithms which are useful in symbolic computation are treated. In Chapter 4 the implementations and the use of MAPLE are discussed. In this chapter a view of the written procedures, relations between theory and implementations and several problems in implementing and using MAPLE for this kind of analyses is discussed. Finally in Chapter 5 conclusions and recommendations for further research are presented.

In four appendices information is collected concerning the software which belongs to this thesis. These software are MAPLE procedures in which analysis and design strategies, discussed in Chapter 3, are implemented. In fact, these procedures are the final results of this thesis. The procedures are collected in two packages: the zerodyn package, which contains important analysis and design tools and the tools package, in which a number of useful computational tools, used frequently by the procedures in zerodyn, are stored. In Appendix A, a description of purpose, calling sequence and results of all procedures is summarized. This description should contain enough information to use the procedures. In Appendix B some worked examples are presented. Some of these examples support the use of the procedures, where others give some insight in the theoretical algorithms presented in the Chapters 3 and 4. In Appendix C flowcharts of some important procedures are collected. These flowcharts do not form elaborate specifications of the procedures by themselves, they are rather supporting a better understanding in the structure of the procedures. Finally, in Appendix D, the integral listings of all procedures are presented.
Chapter 2 MAPLE

MAPLE is a system for symbolic mathematical computation. It has been developed since 1981 by the Symbolic Computation Group at the University of Waterloo, Canada. Recently also groups in Zurich (ETH) and Paris are active for the development of MAPLE. At the moment version V (1991) is available for several processor systems, among which UNIX workstations and the PC.

There exist a few other Symbolic Computation packages. Already in the early sixties the first computer programs which could systematically deal with unassigned variables in general purpose mathematical problems like solving equations and differentiation were developed. These programs, Reduce and Macsyma are worth mentioning, where not only expensive and computational demanding but especially the possibilities where rather limited. These programs are still available and as good as possible modified to present demands but new developments in mathematics and computer hardware have given ground to a new generation of Symbolic Computation systems. At the moment two general purpose systems are market leaders, the one is MAPLE, and the other one is MATHEMATICA, which is indissoluble joined with the name of designer Stephen Wolfram. These two programs offer roughly the same possibilities (and constraints!) and differ only in details like syntax and internal structure. In general MAPLE is hold to be significantly faster in symbolic computation, having slightly more possibilities and support of mathematical routines and having neater semantics, while MATHEMATICA has much better graphical support on screen and plotter (PostScript, animations etc.) and is famous for it's documentation. Both packages are still improved and expanded. Each new release is much stronger than the previous version and less different from the "competitor".

At the moment the possibilities and the power of symbolic computation systems arrived at a level where several scientists show serious interests in the use of such programs for their research. Also this thesis is an investigation for using symbolic computation in analysis and design of control systems.

In my research leading to this thesis I worked with MAPLE on a SUN-Sparc workstation. For this reason I will restrict the further discussion to MAPLE. In the remainder of this chapter some elementary information on MAPLE is presented.

The internal structure of MAPLE is rather simple. The program itself consists of a relative small kernel which contains all standard mathematical operations like fractions, polynomials, factorizations, (exact) solutions of (differential) equations, differentiation, integration, limits, series, manipulation of expressions etc.. Beside this kernel MAPLE has a large library of standard mathematical and manipulation functions (Legendre, Bessel) which are only loaded when they are explicitly needed. This is why MAPLE is relatively fast. However, the strength of MAPLE are the several packages that are supplied. These packages contain a lot of functions and procedures, written in the MAPLE programming language, that support fields of special interest like Statistics, Simplex Linear Programming, Gröbner bases etc. Two of these packages
are especially worth mentioning. In the first place the Linear Algebra package which contains about 100 procedures concerning linear algebra that permit us to manipulate matrices and vectors and to determine among other things the rank, determinant, inverse, eigenvalues, and several decompositions. The functions of the Linear Algebra package are frequently used in the implementations developed in this thesis. In the second place the Student package attracts attention. This package is specially developed for calculus courses at university. It provides functions and procedures for a step by step analysis of differentiation, integration and series etc.. This package forms a solid basis for future calculus education that will be completely changed when strong Symbolic Computation systems like MAPLE will become common property among students. The special functions in the Student Package are not used in the procedures of this thesis.

In essence, MAPLE is an interactive program which is designed for advanced calculus and algebra. But as mentioned in the previous, MAPLE provides possibilities for programming. For this purpose an internal programming language is available. This language allows us to write our own functions and procedures, wherein all MAPLE commands can be used. The MAPLE programming language is a full higher programming language. All conventional programming structures, like repetitions, conditional executions and data structures are available. The user defined procedures can be stored in files with .m as extension. Therefore they are referred to as M-files. All the procedures concerning the analysis of nonlinear control systems defined in this thesis are written in the MAPLE language.

Finally, MAPLE can also be used for numerical calculations, i.e. floating point computations, with infinite accuracy thanks to the internal structure of computations and for (2D,3D) plotting of functions.

More information on the use and properties of MAPLE can be found in [4], [5] and [6].
Chapter 3 Nonlinear Control Systems

Introduction

In this chapter some theoretical results concerning the analysis of nonlinear control systems are presented. First, a state space description of the nonlinear system that is considered throughout this chapter is presented. Afterwards, in subsequent sections, the following notions will be treated: the relative degree, the normal form, the zero dynamics, exact linearization of the input-state equations, and finally exact linearization of the input-output behaviour of nonlinear systems.

Of these notions a definition or a description is given, further information on the meaning and possibilities for employment of these notions for analysis of nonlinear systems is discussed, and if convenient algorithms are presented to compute essential information for the analysis or design of nonlinear control systems.

The emphasis lies on a special class of systems, namely systems having a relative degree. However, for some fields of interest, i.e., the zero dynamics and the exact linearization of input-output response, an extension to a broader class of systems is provided.

The theories discussed in this chapter will be used frequently in the following chapter, in which will be described how most of the presented analysis is implemented in the symbolic computation system MAPLE. Often a reference is made to literature for further information and proofs of propositions, in order to obtain a clear presentation.

The multivariable nonlinear system that is considered throughout this chapter is equal to the one in Notation and Terminology

\[ \dot{x} = f(x) + \sum_{i=1}^{m} g_i(x) u_i \] (N1)

\[ y_i = h_i(x) \quad \forall 1 \leq i \leq p \]

However, the analysis in this chapter and the implementation in the software that is developed in this thesis, is mostly restricted to systems having the same number of input and output channels, in which case \( m = p \).

For some basic analysis, like the relative degree, transformation to the normal form and the zero dynamics, it is possible to achieve the same results for systems having a different number of inputs and outputs, with the constraint that the number of inputs should be larger than or equal to the number of outputs \( (m \geq p) \). Occasionally, at the appropriate places, remarks are provided to specify how results can be adapted in order to include these systems.

It should be clear that SISO systems fit easily in (N1), it is the case where \( m = p = 1 \).
**I The relative degree**

An important property of a system is the notion of the relative degree. A SISO system is said to have a relative degree \( r \) at a point \( x^0 \) of the state space if

\[
\begin{align*}
(i) & \quad L_x^k h(x) = 0 \quad \forall \ x \ \text{near} \ x^0 \quad \forall \ k < r - 1 \\
(ii) & \quad L_x^{r-1} h(x^0) \neq 0
\end{align*}
\]

An important feature of the relative degree is that \( r \) is exactly the number of times that the output \( y(t) \) should be differentiated at an arbitrary moment \( t = \tau \) in order to have the value of the input \( u(\tau) \) explicitly appearing. Therefore, the relative degree of a (SISO) linear system is exactly equal to the difference between the degree of the denominator polynomial and the degree of the numerator polynomial of the corresponding transfer function.

There may be points in the state space, where the relative degree cannot be defined. This occurs when the first function of the sequence

\[
L_x^1 h(x), \ L_x^2 h(x), \ldots, \ L_x^r h(x), \ldots
\]

which is not identically zero for all \( x \) in the neighbourhood of \( x^0 \), is exactly zero at the point \( x = x^0 \). The relative degree can also not be defined when

\[
L_x^k h(x) = 0 \quad \forall \ x \ \text{near} \ x^0 \quad \forall \ k > 0
\]

In this case the output is not depending on the input at all, but only on the initial state.

This definition can be generalized for MIMO systems. A MIMO system has a vector relative degree \( \{r_1, \ldots, r_m\} \) at a point \( x^0 \) of the state space if

\[
\begin{align*}
(i) & \quad L_x^j h_i(x) = 0 \quad \forall \ x \ \text{near} \ x^0 \quad \forall \ k < r_i - 1 \\
(ii) & \quad \text{the} \ m \times m \ \text{matrix} \ A \ \text{is nonsingular at} \ x = x^0
\end{align*}
\]

\[
A = \begin{bmatrix}
L_x^{r_i-1} h_1(x) & \cdots & L_x^{r_i-1} h_i(x) \\
L_x^{r_i-1} h_2(x) & \cdots & L_x^{r_i-1} h_2(x) \\
\vdots & \ddots & \vdots \\
L_x^{r_i-1} h_m(x) & \cdots & L_x^{r_i-1} h_m(x)
\end{bmatrix}
\]

Note that each integer \( r_i \) is associated with the \( i \)-th output \( h_i \) of the system. When the sum of all the integers \( r_i \) is equal to the dimension \( n \) of the state space, the system is said to have a full order relative degree. Here, \( r_i \) is exactly the number of times that the \( i \)-th output \( y_i(t) \) should be differentiated at \( t = \tau \) in order to have at least one of the components of the input vector \( u(\tau) \) explicitly appearing.

In close analogy with the SISO case, also in the MIMO case the vector relative degree cannot be defined:

- If, for the first value of \( k \) (0\( \leq k \leq n \)) for which at least one element of the row

\[
\begin{bmatrix}
L_x^{r_i} h_1(x) & \cdots & L_x^{r_i} h_i(x)
\end{bmatrix}
\]

is not identically zero for all \( x \) in the neighbourhood of \( x^0 \) and is exactly zero at \( x = x^0 \) AND no other element of this row is not identically zero for all \( x \) in the neighbourhood of \( x^0 \) as well, and NOT exactly zero at \( x = x^0 \).

- If the matrix \( A \) is singular. Formally, the relative degree of each output channel may be defined (at least one element of each row in \( A \) should be nonzero), but will not be valid in most of the subsequent analysis.
If for some value $1 \leq i \leq m$

$$L_x L_{x_k}^j h(x) = 0 \quad \forall \ x \text{ near } x^0 \quad \forall \ k > 0 \quad \forall \ 1 \leq j \leq m$$

then the $i$-th output of the system is not at all affected by the input and the relative degree is not well defined.

As indicated in the introduction of this chapter, it is possible to extend the results established so far to a system having a number of inputs larger than the number of outputs ($m \geq p$). In this case the nonsingularity condition of the matrix $A$ is replaced by the assumption that this matrix has a rank equal to the number of rows, i.e. rank$(A) = p$.

The relative degree plays a crucial role in some of the analysis in this thesis. In [1], Isidori treats the class of nonlinear systems having a relative degree $\{r_1, \ldots, r_m\}$ as a special class of systems and shows among other things that all systems having a relative degree can be rendered noninteractive by static state feedback and that a full order relative degree is a necessary condition for a nonlinear system to be exact feedback linearizable. In the following sections these, and other, aspects of nonlinear control systems will be discussed. In the remainder of this chapter, the notions relative degree (SISO) and vector relative degree (MIMO) will just be mentioned as relative degree.

II The normal form

By means of a change of coordinates in the state space, the system equations can be (locally) transformed to a "normal form". This normal form is of particular interest because it entails a very simple structure in the equations describing the system. The normal form is found useful in understanding how certain control problems can be solved. In the following proposition the mapping which describes the change of coordinates is presented. This proposition is treated in [1](prop.4.1.4) and concerns SISO systems. The simple structure of single-input single-output systems leads to a rather transparent analysis which gives insight in the theory. Later on an extension to MIMO systems will be made.

Suppose a system has a relative degree $r$ at $x^0$. Then $r \leq n$. Now set the first $r$ functions of the mapping $\Phi$ describing the coordinates transformation:

\[
\begin{align*}
\Phi_1(x) &= h(x) \\
\Phi_2(x) &= L_h h(x) \\
&\vdots \\
\Phi_r(x) &= L_{r-1} h(x)
\end{align*}
\]

(3.3)

If $r$ is strictly less than $n$, it is always possible to find $n-r$ more functions such that the mapping

\[
\Phi(x) = \begin{bmatrix} 
\Phi_1(x) \\
\vdots \\
\Phi_r(x)
\end{bmatrix}
\]

(3.4)

has a jacobian matrix which is nonsingular at $x^0$ and therefore qualifies as a local coordinates transformation in a neighbourhood of $x^0$. The value at $x^0$ of these additional
functions can be chosen arbitrarily. Moreover, it is always possible to choose $\Phi_{n_1}(x)$, ..., $\Phi_{n_1}(x)$ in such a way that

$$L_x^i \Phi(x) = 0$$

\[ \forall x \text{ near } x^0 \quad \forall r+1 \leq i \leq n \tag{3,5} \]

This mapping $\Phi$ describes the state transformation to the normal form. For easy understanding of the special structure of the system equations in the normal form the state space description of a SISO system in the new coordinates $z = \Phi(x)$ is derived. The calculations leading to this structure are easy to carry out:

For the first $z_1, ..., z_{r-1}$ new coordinates we obtain equations in the form:

$$\frac{dz_i}{dt} = \frac{\partial \Phi_i}{\partial x} \frac{dx}{dt} = \frac{\partial L_x^{i-1} \Phi}{\partial x} \frac{dx}{dt} = L_x^{i} h(x(t)) = \Phi_{n_1}(x(t)) = \dot{z}_i(t) \quad \forall 1 \leq i \leq r-1 \tag{3,6}$$

Next, for the special equation in the coordinate $z_r$ we obtain:

$$\frac{dz_r}{dt} = L_x^{r} h(x(t)) + L_x^{r-1} h(x(t)) u(t)$$

replacing

$$x(t) = \Phi^{-1}(z(t))$$

and setting:

$$a(z) = L_x^{r-1} h(\Phi^{-1}(z))$$

$$b(z) = L_x^{r} h(\Phi^{-1}(z))$$

this equation can be written as:

$$\frac{dz_r}{dt} = b(z(t)) + a(z(t)) u(t) \tag{3,7}$$

As far as the other new coordinates are concerned, we cannot expect any special structure for the corresponding equations, if nothing else has been specified. However, if the additional mapping functions $\Phi_{n_1}(x)$, ..., $\Phi_{n_1}(x)$ have been chosen in such a way that they are solutions of the partial differential equation $L_x^i \Phi(x) = 0$ (3,5) then the nice properties of these last $n-r$ equations appears:

$$\frac{dz_i}{dt} = \frac{\partial \Phi}{\partial x} (f(x(t)) + g(x(t)) u(t) = L_x^i \Phi(x(t)) + L_x^i \Phi(x(t)) u(t) \tag{3,9}$$

setting:

$$q(z) = L_x^i \Phi(\Phi^{-1}(z)) \quad \forall r+1 \leq i \leq n$$

these equations can be rewritten in the new coordinates as:

$$\frac{dz_i}{dt} = q(z(t)) \quad \forall r+1 \leq i \leq n \tag{3,10}$$

In general it is not easy to solve this partial differential equation, i.e. to construct a set of $n-r$ functions such that $L_x^i \Phi(x) = 0$ and $\Phi$ defines a (local) smooth coordinate transformation. Usually it is much simpler to construct functions $\Phi_{n_1}(x)$, ..., $\Phi_{n_1}(x)$ with the only property that the jacobian matrix of the mapping $\Phi(x)$ is nonsingular at $x^0$ so
that the mapping defines (at least locally) a smooth coordinate transformation. If this is the case the same structure for the first $r$ equations is derived, but it is not possible to obtain a special structure for the last $n-r$ ones (3.10). Therefore they appear in a form with the input $u$ explicitly present:

$$
\dot{z}_i = q_i(z) + p_i(z)u \quad \forall \ r+1 \leq i \leq n
$$

(3.11)

In summary, the state space description of the SISO system in the new coordinates will be as follows:

$$
\begin{align*}
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\vdots \\
\dot{z}_{r-1} &= z_r \\
\dot{z}_r &= b(z) + a(z)u \\
\dot{z}_{r+1} &= q_{r+1}(z) \\
&\vdots \\
\dot{z}_n &= q_n(z) \\
y &= h(x) = z_i
\end{align*}
$$

(3.12)

In a block diagram the special structure of these equations in normal form will become clear. Set $v = b(z) + a(z)u$, then

In order to write the equations in a slightly more condensed manner, now a suitable vector notation is introduced. As follows from the previous, it makes sense to distinguish between the coordinates $z_1, \ldots, z_r$ and the other coordinates $z_{r+1}, \ldots, z_n$. We will do that in the following way:

$$
\zeta = \begin{bmatrix} z_1 \\ \vdots \\ z_r \end{bmatrix}, \quad \eta = \begin{bmatrix} \eta_{r+1} \\ \vdots \\ \eta_n \end{bmatrix}
$$

(3.13)

In these coordinates the normal form can be rewritten as:

$$
\begin{align*}
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\vdots \\
\dot{z}_{r-1} &= z_r \\
\dot{z}_r &= b(\zeta, \eta) + a(\zeta, \eta)u \\
\eta &= q(\zeta, \eta)
\end{align*}
$$

(3.14)
An interesting remark considering the equilibrium point $x^o$ is that, if $x^o$ is chosen such that $f(x^o)=0$ and $h(x^o)=0$, then necessarily the first $r$ new coordinates $z_i$ are 0 at $x^o$. The value at $x^o$ of the last $n-r$ new coordinates can be chosen arbitrarily so particularly being 0 at $x^o$. Therefore, without loss of generality, one can assume that $\zeta=0$ and $\eta=0$ at $x^o$. Thus, if $x^o$ was an equilibrium point for the system in the original coordinates, its corresponding point $(\zeta, \eta) = (0,0)$ is an equilibrium for the system in the new coordinates. From this we deduce

$$b(\zeta, \eta) = 0 \quad q(\zeta, \eta) = 0 \quad \text{at } (\zeta, \eta) = (0,0)$$

The previous analysis concerned the normal form of SISO systems. Now an extension to MIMO systems is made. In a bird’s-eye view the state transformation to the normal form and the special structure of the describing equations will be treated. The results turn out to be straightforward generalisations of the corresponding SISO versions and are summarized in the following proposition: (II(prop.5.1.3))

Suppose a system has a relative degree \( \{r_1, \ldots, r_m\} \) at $x^o$. Then\(^1\)

\[
\begin{align*}
    r_1 + \cdots + r_m &\leq n \\
    \Phi_i'(x) &= h_i(x) \\
    \Phi_i''(x) &= L_i h_i(x) \\
    \cdots \\
    \Phi_i^{(n)}(x) &= L_i^{n-1} h_i(x)
\end{align*}
\]

If $r = r_1 + \ldots + r_m$ is strictly less than $n$, it is always possible to find $n-r$ more functions such that the mapping $\Phi(x)$, represented by the column

\[
\begin{bmatrix}
    \Phi_1(x) \\
    \vdots \\
    \Phi_n(x)
\end{bmatrix}
\]

has a jacobian matrix which is nonsingular at $x^0$ and therefore qualifies as a local coordinates transformation in a neighbourhood of $x^0$. The value at $x^0$ of these additional functions can be chosen arbitrarily. Moreover, if the distribution

\[
G = \text{span}\{e_{r+1}, \ldots, e_m\}
\]

is involutive near $x^0$, it is always possible to choose $\Phi_{r+1}(x), \ldots, \Phi_m(x)$ in such a way that

\[
L_{e_i} \Phi(x) = 0 \\
\forall x \text{ near } x^0 \quad \forall r+1 \leq i \leq n \quad \forall 1 \leq j \leq m
\]

This mapping $\Phi(x)$ describes the state transformation to the normal form. Also in the MIMO case it makes sense to distinguish between the first $r$ and the last $n-r$ new coordinates. Consistently with the notation in the SISO case set:

---

\(^1\) The necessity of the statement $r_1 + \ldots + r_m \leq n$ is made explicitly clear by Isidori in case of a nonsingular matrix $A$. 

In this notation the normal form of the equations describing a nonlinear system with \( m \) inputs and \( m \) outputs, having a relative degree \( \{ r_1, ..., r_m \} \) at \( x_0 \), can be written as:

\[
\begin{align*}
\dot{\xi}_i &= \Phi_i(x) \\
\dot{\xi}_i &= \Phi_i(x) \\
\vdots &= \vdots \\
\dot{\xi}_i &= \Phi_i(x)
\end{align*}
\]  

\[
\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \Phi_{r_1}(x) \\ \Phi_{r_2}(x) \\ \vdots \\ \Phi_{r_m}(x) \end{bmatrix}
\]

\[
\zeta = (\xi_1, ..., \xi_m)
\]

In this notation the normal form of the equations describing a nonlinear system with \( m \) inputs and \( m \) outputs, having a relative degree \( \{ r_1, ..., r_m \} \) at \( x_0 \), can be written as:

\[
\begin{align*}
\dot{\xi}_i &= \Phi_i(x) \\
\dot{\xi}_i &= \Phi_i(x) \\
\vdots &= \vdots \\
\dot{\xi}_i &= \Phi_i(x)
\end{align*}
\]  

\[
\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \Phi_{r_1}(x) \\ \Phi_{r_2}(x) \\ \vdots \\ \Phi_{r_m}(x) \end{bmatrix}
\]

\[
\zeta = (\xi_1, ..., \xi_m)
\]

However, if the distribution spanned by the vector fields \( g_1(x), ..., g_m(x) \) is not involutive (which is likely to be the most general case) or if we are not able or willing to find solutions for the set of partial differential equations (3.18), the previous notation for the coordinates \( \eta \) (3.21) does not hold and the inputs \( u_i \) appears explicitly in these equations:

\[
\begin{align*}
\dot{\xi}_i &= \Phi_i(x) \\
\dot{\xi}_i &= \Phi_i(x) \\
\vdots &= \vdots \\
\dot{\xi}_i &= \Phi_i(x)
\end{align*}
\]  

\[
\eta = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \Phi_{r_1}(x) \\ \Phi_{r_2}(x) \\ \vdots \\ \Phi_{r_m}(x) \end{bmatrix}
\]

\[
\zeta = (\xi_1, ..., \xi_m)
\]

However, if the distribution spanned by the vector fields \( g_1(x), ..., g_m(x) \) is not involutive (which is likely to be the most general case) or if we are not able or willing to find solutions for the set of partial differential equations (3.18), the previous notation for the coordinates \( \eta \) (3.21) does not hold and the inputs \( u_i \) appears explicitly in these equations:

\[
\dot{\eta}_i = q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j
\]

\[
\eta = q(\xi, \eta)
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]  

In a block diagram, the special structure of the equations (3.20) to (3.23) will become clear. For the sake of clear notation set

\[
\begin{align*}
\dot{\xi}_i &= \Phi_i(x) \\
\dot{\xi}_i &= \Phi_i(x) \\
\vdots &= \vdots \\
\dot{\xi}_i &= \Phi_i(x)
\end{align*}
\]  

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]  

However, if the distribution spanned by the vector fields \( g_1(x), ..., g_m(x) \) is not involutive (which is likely to be the most general case) or if we are not able or willing to find solutions for the set of partial differential equations (3.18), the previous notation for the coordinates \( \eta \) (3.21) does not hold and the inputs \( u_i \) appears explicitly in these equations:

\[
\dot{\eta}_i = q(\xi, \eta) + \sum_{j=1}^{m} p_j(q(\xi, \eta)) u_j
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]  

In a block diagram, the special structure of the equations (3.20) to (3.23) will become clear. For the sake of clear notation set

\[
\begin{align*}
\dot{\xi}_i &= \Phi_i(x) \\
\dot{\xi}_i &= \Phi_i(x) \\
\vdots &= \vdots \\
\dot{\xi}_i &= \Phi_i(x)
\end{align*}
\]  

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]  

However, if the distribution spanned by the vector fields \( g_1(x), ..., g_m(x) \) is not involutive (which is likely to be the most general case) or if we are not able or willing to find solutions for the set of partial differential equations (3.18), the previous notation for the coordinates \( \eta \) (3.21) does not hold and the inputs \( u_i \) appears explicitly in these equations:

\[
\dot{\eta}_i = q(\xi, \eta) + \sum_{j=1}^{m} p_j(q(\xi, \eta)) u_j
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]  

In a block diagram, the special structure of the equations (3.20) to (3.23) will become clear. For the sake of clear notation set

\[
\begin{align*}
\dot{\xi}_i &= \Phi_i(x) \\
\dot{\xi}_i &= \Phi_i(x) \\
\vdots &= \vdots \\
\dot{\xi}_i &= \Phi_i(x)
\end{align*}
\]  

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]  

However, if the distribution spanned by the vector fields \( g_1(x), ..., g_m(x) \) is not involutive (which is likely to be the most general case) or if we are not able or willing to find solutions for the set of partial differential equations (3.18), the previous notation for the coordinates \( \eta \) (3.21) does not hold and the inputs \( u_i \) appears explicitly in these equations:

\[
\dot{\eta}_i = q(\xi, \eta) + \sum_{j=1}^{m} p_j(q(\xi, \eta)) u_j
\]

\[
\begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix}
\]

\[
\begin{align*}
\dot{\eta}_i &= q(\xi, \eta) + \sum_{j=1}^{m} a_j(q(\xi, \eta)) u_j \\
\eta_i &= q(\xi, \eta)
\end{align*}
\]
Chapter 3
Nonlinear Control Systems

Considering the equilibrium point of the system in normal form we can say: If \( x^0 \) is an equilibrium of \( f(x) \), if \( h_1(x^0) = ... = h_m(x^0) = 0 \), and if \( \Phi_{-1}(x^0) = ... = \Phi_{-1}(x^0) = 0 \), the normal form thus found is locally defined in the neighborhood of the point \( (\zeta, \eta) = (0,0) \). However, it is not necessary to choose \( (\zeta^0, \eta^0) = (0,0) \). In that case the normal form is defined around the working point \( (\zeta^0, \eta^0) = (\Phi_{-1}(x^0), \Phi_{-1}(x^0)) \).

Note that the coefficients \( a_{ij}(\zeta, \eta) \) in (3.20) are exactly the entries of the matrix \( A_{(3,2)} \) with \( x \) replaced by \( \Phi_{-1}(\zeta, \eta) \). Likewise, the coefficients \( b_{ij}(\zeta, \eta) \) can be considered as entries of the \( mx1 \) column \( b \), which will be frequently used in the following sections:

\[
\begin{bmatrix}
L_{ij} h_i(x) \\
\vdots \\
L_{mn} h_m(x)
\end{bmatrix}
\quad (3.24)
\]

III The zero dynamics

In this section the concept of zero dynamics will be treated. Already in Chapter 1 it is claimed that this concept plays an important role in analyzing nonlinear systems and designing control systems. Also is stated that two approaches to this concept can be distinguished. Therefore, the discussion in this section is presented in two steps: First, the Problem of Zeroing the Output is considered, i.e. to find all pairs of initial conditions \( x^0 \) and inputs \( u(t) \) consistent with the constraint that the output function \( y(t) \) is identically zero for all times, after which the corresponding internal dynamics of the system -which are called the zero dynamics- are analyzed. Second, the Zero Dynamics Algorithm is presented. This algorithm constructs a feedback which renders the closed-loop system maximally unobservable. The internal dynamics of the unobservable component are called the zero dynamics. The latter approach is not restricted to systems having a relative degree.

The analysis is started with solving the problem of zeroing the output and determine the corresponding initial states and input:

If \( y(t) = 0 \) for all \( t \), then

\[
h_i(x(t)) = L_{ij} h_i(x(t)) = ... = L_{in}^{-1} h_i(x(t)) = 0 \quad \forall 1 \leq i \leq m
\]

(3.25)

i.e. \( \zeta(t) = 0 \) for all \( t \). So, when the output of the system is forced to be identically zero, its state is constrained to evolve in such a way that also \( \zeta(t) \) is identically zero.

Imposing the derivative of order \( r_i \) of \( y(t) \) being zero, for all \( 1 \leq i \leq m \), constrains the inputs \( u_1(t), ..., u_m(t) \) to be solutions of the system of equations

\[
y_i^{(r_i)}(t) = b_1(0, \eta(t)) + \sum_{j=1}^{m} a_{ij}(0, \eta(t)) u_j(t) = 0 \quad \forall 1 \leq i \leq m
\]

(3.26)
or:

\[
b(0, \eta(t)) + A(0, \eta(t)) u(t) = 0
\]

The matrix \( A_{(3,2)} \) is nonsingular at \( x = x^0 \) by definition. Thus the matrix \( A(\zeta, \eta) \) is nonsingular at \( (\zeta^0, \eta^0) = (\Phi_{-1}(x^0), \Phi_{-1}(x^0)) = (0, \eta^0) \) and the above equation can be solved for \( u(t) \) if \( \eta(t) \) is close to \( \eta^0 \).
From these considerations it is deduced that if the output $y(t)$ has to be zero for all $t$ then necessarily:
- the initial state of the system must be set to a value such that $\xi^0 = 0$, whereas $\eta^0$ can be chosen arbitrarily.
- the inputs $u(t)$ must be set, according to the value of $\eta^0$, to
  \[ u(t) = -A(0,\eta(t))^1b(0,\eta(t)) \]  
  \[ (3.27) \]
  with $\eta(t)$ the solution of the differential
  \[ \eta(t) = q_0(0,\eta(t)) \]  
  \[ (3.28) \]
  where $q_0$ is defined as
  \[ q_0(\xi,\eta) = q(\xi,\eta) - p(\xi,\eta)[A(\xi,\eta)]^1b(\xi,\eta) \]  
  or (if (3.21))
  \[ q_0(\xi,\eta) = q(\xi,\eta) \]  
  \[ (3.29) \]
  and initial condition $\eta(0) = \eta^0$

- for each set of initial data $\xi=0$ and $\eta=\eta^0$ the input thus defined is the unique input capable to keep the output identically zero for all times.

The dynamics (3.28) correspond to the dynamics describing the "internal" behaviour of the system when input and initial conditions have been chosen in such a way that the output remains identically zero. These dynamics are called the zero dynamics of the system. Now the relevance of the equations in normal form in relation with the zero dynamics has become clear. It has been show that the zero dynamics are completely described by the normal form equations (3.21) and (3.23) in the coordinates $\eta$.

Indeed all properties of the zero dynamics so far were covered using the normal form because of the remarkable insight and computational ease that this notation offers. Though, it is possible to arrive at similar results starting from system equations in different forms. We have already seen that one still can identify the zero dynamics in the case where one has not been able to find exactly the normal form because of difficulties in constructing the additional functions of the mapping $\Phi$ with the property that $L_\Phi(x) = 0$ (3.18). A complicated extra term arises, but the structure of the equations turns out to be the same. The problem of zeroing the output can also be analyzed directly on the original form of the equations but in general the expression for the zero dynamics may become more complicated.

A structural constraint in the theory covered so far, is that only systems for which a relative degree can be defined can be treated. However, also in view of a broader class of systems - systems which have a relative degree are fully incorporated- it is possible to consider the problem of setting the output to zero by means of a proper choice of initial state and input. For this new class of systems, the assumption of a relative degree is replaced by some milder conditions, namely the constancy of the dimensions of certain distributions and/or the rank of certain mappings. Note that this is a completely different way of analyzing systems and depends on a mathematical background regarding differential geometry. In the last two chapters of [1], Isidori pays a lot of attention to several analysis and design methodologies concerning nonlinear (control) systems belonging to this broader class of systems. These extensions seem to be another step in developing general methodologies for design of nonlinear control systems. In this thesis just a fair attempt is made to treat the zero dynamics. However, this is not done by means of a theoretical, mathematical presentation, but rather by discussing the so called Zero Dynamics Algorithm.

The Zero Dynamics Algorithm is an iterative procedure, presented in [1] by Isidori, which constructs a locally maximal output zeroing submanifold. An output zeroing submanifold $M$ is a
submanifold of the state space with the property that, for some choice of the feedback control \( u(x) \), the trajectories of the closed loop system which start in \( M \) will stay in \( M \) for all times (at least in a neighbourhood of \( t=0 \)) and the corresponding output is identically zero. A submanifold is called locally maximal if for some neighbourhood \( U^0 \) of \( x^0 \) any other submanifold \( M' \) satisfies \( M \cap U^0 \supset M' \cap U^0 \). A physical interpretation of this algorithm is that a feedback is designed which renders the closed-loop system maximally unobservable. The internal dynamics of the unobservable part are considered to be the zero dynamics.

**Zero Dynamics Algorithm**

Consider the general nonlinear system \((N1, N2)\) and a working point \( x^0 \) in the state space. Suppose \( f(x^0) = h(x^0) = 0 \). This means that if \( x(0) = x^0 \) and \( u(t) = 0 \) for all \( t > 0 \) then also the output \( y(t) = 0 \) for all times.

**step 0:**
The locally maximal submanifold \( M \) is initialized as \( M_0 \), the set of points where the mapping \( h(x) \) is zero.

\[
M_0 = \{ h(x) = 0 \}
\]  

(3.30)

Suppose the rank \( s_0 \) of the differential \( dh \) of \( h \) is constant in a neighbourhood \( U_0 \) of \( x^0 \). The first \( s_0 \) functions of \( h(x) \) which correspond to the first \( s_0 \) independent rows of \( dh \) are selected in the matrix \( H_0 \) by multiplication with a permutation matrix \( S_p \). Without loss of generality this can be denoted by

\[
H_0(x) = S_0 h(x) \quad \text{with} \quad S_0 = [I \ 0]
\]

Let \( M_0^c \) denotes the connected component of \( M_0 \) and \( U_0 \) through \( x^0 \):

\[
M_0^c = M_0 \cap U_0 = \{ x \in U_0 : H_0(x) = 0 \}
\]  

(3.31)

\( M_1 \), the first stage in restriction of the output zeroing submanifold can now be computed as the set of all \( x \in M_0^c \) such that the equation

\[
L_h(x) + L_g H_0(x) u = 0
\]  

(3.32)

is solvable in \( u \). This can be elaborated as follows. Suppose the matrix \( L_g H_0(x) \) has a constant rank \( r_0 \) for all \( x \in M_0^c \). Then define a \((s_0-r_0) \times s_0\) matrix \( R_0 \) whose rows annihilate the columns of \( L_g H_0(x) \). In particular

\[
R_0(x) L_g H_0(x) = 0
\]

The equation (3.32) can now be solved in \( u \) if and only if \( x \) is such that

\[
R_0(x) L_h(x) = 0
\]

Setting

\[
\Phi_0(x) = R_0(x) L_h(x)
\]  

(3.33)

the algorithm can be continued.

**step 1:**
For some neighbourhood \( U_1 \) of \( x^0 \), express the set

\[
M_1^c = M_1 \cap U_1 = \{ x \in U_1 : H_0(x) = 0 \quad \text{and} \quad \Phi_0(x) = 0 \}
\]  

(3.34)

as a smooth manifold. If the smooth mapping \( \text{col}(H_0(x), \Phi_0(x)) \) has a constant rank, say \( s_0 + s_1 \), near \( x^0 \) then in this first step the mappings \( H_1(x) \) and \( \Phi_1(x) \) will be constructed and the algorithm may be continued with the second step, etc.
In a general notation we can denote for each step $k \geq 1$

At step $k$ the iteration is started with the mappings $H_k(x)$ and $\Phi_k(x)$, where $H_k(x)$ is such that the rank of its differential is exactly equal to the number $s_0 + \ldots + s_{k-1}$ of its rows and $\Phi_k(x) = R_{s_0 + \ldots + s_{k-1}} H_k(x)$ is such that $R_{s_1}(x)L_{s_1} H_k(x) = 0$.

Set, for some neighbourhood $U_k$ of $x$,

$$M_k^* = M_k \cap U_k = \{ x \in U_k : H_k(x) = 0 \text{ and } \Phi_k(x) = 0 \} \quad (3.35)$$

as a smooth manifold. Suppose the mapping $\text{col}(H_k(x), \Phi_k(x))$ has a constant rank $(s_0 + \ldots + s_k)$ near $x$. Let $S_k$ denote the permutation matrix which selects the first $s_k$ functions of $\Phi_k(x)$ which correspond to the first $s_k$ independent rows of the differential of $\Phi_k(x)$, and set

$$H_k(x) = \text{col}(H_k(x), S_k \Phi_k(x))$$

obviously

$$M_k^* = M_k \cap U_k = \{ x \in U_k : H_k(x) = 0 \} \quad (3.36)$$

In order to restrict the output zeroing submanifold in the present step, one has to look at the equation

$$L_{s} H_k(x) + L_{x} H_k(x) u = 0 \quad (3.37)$$

Suppose the matrix $L_j H_k(x)$ has a constant rank $r_j$ for all $x \in M_k^*$. Then define a $(s_0 + \ldots + s_k) \times (s_0 + \ldots + s_k)$ matrix $R_k$ whose rows annihilate the columns of $L_j H_k(x)$. In particular it is possible to choose submatrices $P$ and $Q$ in

$$R_k(x) = \begin{bmatrix}
R_{s_k}(x) & 0 \\
0 & R_k(x)
\end{bmatrix}$$

Such that

$$R_k(x)L_j H_k(x) = 0$$

By setting the new mapping $\Phi_k(x)$ as (Note that $\Phi_k$ has $(s_0 + r_k)$ rows.)

$$\Phi_k(x) = P(x)L_j H_k(x) + Q(x)L_{s_0 + s_k} \Phi_k(x) \quad (3.38)$$

the construction of the output zeroing submanifold may be continued with the next step $k+1$.

The algorithm is terminated if, for some value of $k$, the matrix $L_j H_k(x)$ gets full rank $m$. At this stage no mapping $\Phi_k(x)$ can be created any more and the submanifold

$$M_k = \{ H_k(x) = 0 \} \quad (3.39)$$

is the locally maximal output zeroing submanifold. From this, the zero dynamics are constructed in the following way. First, the "output zeroing inputs" $u$ are solved from the equation

$$L_j H_k(x) + L_{x} H_k(x) u = 0 \big|_{x \in M_k} \quad (3.40)$$

Imposing this feedback on the system in which (also) the submanifold $M_k$ has been substituted lead to the zero dynamics of the system.
IV Exact linearization of the input-state equations

In this section is shown how a system \((Nl)\) having the same number of \(m\) inputs and outputs can be transformed into a linear and fully controllable system by means of static state feedback and change of coordinates in the state space. The order in which these two operations are performed does not influence the results.

Consider a nonlinear system having a relative degree \(\{r_1, \ldots, r_m\}\) satisfying \(r_1 + \ldots + r_m = n\). Starting from the system equations in normal form.

\[
\begin{align*}
\dot{\xi}_1^i &= \xi_2^i \\
\dot{\xi}_2^i &= \xi_3^i \\
&\vdots \\
\dot{\xi}_{r-1}^i &= \xi_r^i \\
\dot{\xi}_r^i &= b_i(\xi) + \sum_{j=1}^{m} a_{ij}(\xi) u_j
\end{align*}
\]

(3.41)

Introduce a new reference input \(v\) of the form:

\[
v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{bmatrix} = b(\xi) + A(\xi) u
\]

(3.42)

The above equations can be solved for \(u\) because in the neighbourhood of \(\xi_0 = \Phi^{-1}(x_0)\) the matrix \(A(\xi)\) is nonsingular. Now imposing the static state feedback

\[
u = A^{-1}(\xi)[-b(\xi) + v]
\]

(3.43)

yields a system characterized by the \(m\)-sets of equations:

\[
\begin{align*}
\dot{\xi}_1^i &= \xi_2^i \\
\dot{\xi}_2^i &= \xi_3^i \\
&\vdots \\
\dot{\xi}_{r-1}^i &= \xi_r^i \\
\dot{\xi}_r^i &= v_i
\end{align*}
\]

(3.44)

which is clearly linear (from the new input \(v\) to the new state \(\xi\)) and controllable. The special structure of this linearized system is characterized by \(m\) chains of \(r_i\) integrators each.

From this straightforward kind of view follows that having a full order relative degree is a sufficient condition for a system to be rendered linear and controllable. In [1](lemma 5.2.1), Isidori showed this is also a necessary condition.

Because the order in which the feedback and transformation operations are performed does not matter, it is clear that also in terms of the original description of the system a linearizing feedback is defined:

\[
u = \alpha(x) + \beta(x) v
\]

(3.45)

\[
\alpha(x) = -A^{-1}(x) b(x) \quad \beta(x) = A^{-1}(x)
\]
with $A(x)$ and $b(x)$ as in (3.2) and (3.24), however, only after the linearizing change of coordinates

$$
\zeta^i_k(x) = L_{ij}^{-1}h_i(x)
$$

the special properties of the system will arise.

From the previous sections is clear that the relative degree depends on the chosen output of the system. We also know that a full order relative degree is a necessary condition for the existence of a solution for the exact linearization problem. So, if we want to linearize the system we should find $m$ output functions for which the system has a full order relative degree. In other words: the conditions for the existence of such output functions are the same conditions as for the existence of a solution for the exact linearization problem.

From the definition of the relative degree (3.2) follows that the functions $\lambda_1, ..., \lambda_m$ we look for should be solutions of the partial differential equations:

$$
L_{\zeta_k}^i \lambda_i(x) = 0 \quad \forall x \text{ near } x^0 \quad \forall 0 \leq k \leq r_i - 2 \quad \forall 1 \leq i, j \leq m
$$

(3.47)

With, as a nontriviality condition the nonsingularity of the matrix $A$, and additionally the condition $r_1 + r_2 + ... + r_m = n$.

By [1](lemma 4.1.3) these equations are equivalent to equations of the form:

$$
\left< d\lambda_i(x), \quad ad^*_j g(x) \right> = 0 \quad \forall x \text{ near } x^0 \quad \forall 0 \leq k \leq r_i - 2 \quad \forall 1 \leq i, j \leq m
$$

(3.48)

Which is clearly a set of first order partial differential equations:

$$
L_{\zeta_k}^i \lambda_i(x) = L_{\zeta_k}^i \lambda_i(x) = L_{\zeta_k}^i \lambda_i(x) = 0 \quad \forall x \text{ near } x^0 \quad \forall 1 \leq i, j \leq m
$$

(3.49)

In [1], Isidori derived necessary and sufficient conditions for the existence of solutions satisfying equations (3.47). The conditions are stated in terms of properties of suitable distributions spanned by vector fields of the form:

$$
G_0 = \text{span}\{g_1, ..., g_m\}
$$

$$
G_1 = \text{span}\{g_1, ..., g_m, ad_1 g_1, ..., ad_m g_m\}
$$

$$
... \quad 1 \leq i, j \leq m \quad i = 0, 1, ..., n-1
$$

(3.50)

In [1](lemma 5.2.3) is proved:

Suppose the matrix $g(x^0)$ has rank $m$. Then there exist a neighbourhood $U$ of $x^0$ and $m$ real-valued functions $\lambda_1, ..., \lambda_m$ defined on $U$ such that the system

$$
x = f(x) + g(x)u
$$

$$
y = \lambda(x)
$$

(3.51)

has a relative degree $\{r_1, ..., r_m\}$ at $x^0$, with
\[
\begin{align*}
  r_1 + r_2 + \cdots + r_m &= n \\
  \text{if and only if:}
\end{align*}
\]

(i) for each 0 ≤ i ≤ n-1, the distribution \( G_i \) has constant dimension near \( x^0 \)

(ii) the distribution \( G_n \) has dimension \( n \)

(iii) for each 0 ≤ i ≤ n-2, the distribution \( G_i \) is involutive.

In the proof of this result is indicated how the functions \( \lambda_1(x), \ldots, \lambda_m(x) \) can be constructed. This can be done in \( k \) iteration steps, where the integer \( k \) is such that \( \dim(G_{n-1}) < n \) \( \dim(G_{n}) = n \).

From (3.48) follows that for each value of \( i \) the differential \( d\lambda_i(x) \) must be a vector belonging to the distribution

\[
\left( \text{span}\left\{ ad_j^i G : 0 \leq k \leq r_i, 1 \leq j \leq m \right\} \right)^\perp = G_{n-2}^i
\]

This observation forms the basis of the construction of the output functions in successive iteration steps.

In the first step \( m_1 \)

\[
m_1 = n - \dim(G_{n-1})
\]

functions \( \lambda_i \) can be constructed such that

\[
\text{span}\{ d\lambda_i : 1 \leq i \leq m_1 \} = G_{n-1}^i
\]

If \( m_1 \) is strictly less than \( m \), then the algorithm may be continued. The idea is to look at the distribution \( (G_{n-1}^i) \). Note that

\[
G_{n-1}^i \subset G_{n-2}^i
\]

because \( G_{n-2}^i \subset G_{n-1}^i \) by definition.

From this distribution \( m_2 \) new output functions can be constructed, which differentials should be independent from the differentials of the functions \( \lambda_i \) found in the first step \( \text{and} \) their derivatives \( L\lambda_i \). So in the second step

\[
m_2 = \dim(G_{n-2}^i) - 2m_1
\]

functions \( \lambda_i \) can be constructed such that

\[
\text{span}\left\{ d\lambda_i : m_1 + 1 \leq i \leq m_1 + m_2 \right\} = \left( \text{span}\left\{ G_{n-2}^i \cup \text{span}\{ d\lambda_i, dL\lambda_i(x) : 1 \leq i \leq m_1 \} \right\} \right)^\perp
\]

If \( m_1 + m_2 \) is strictly less than \( m \), the algorithm may be continued and distribution \( (G_{n-2}^i) \) is regarded. Also the differentials of all functions \( \lambda_i \) and their derivatives already used, are considered in the same way as in the previous step. In each step \( i \), \( m_i \) new functions can be constructed

\[
m_i = n - \dim(G_{n-1}^{i}) - im_1 - (i-1)m_2 - \cdots - 2m_{i-1}
\]
Finally, in the last iteration step, the distribution \((G_d)^t\) is considered. A result of the previous steps is that a basis of \(G^t_d\) is constructed: All the differentials of functions considered in the previous steps together with the differentials of the functions that are found in the last step span a basis of \(G^t_d\):

\[
\begin{align*}
\frac{d}{dx} f_1(x), \frac{d}{dx} f_2(x), \ldots, \frac{d}{dx} f_m(x) & \quad \forall 1\leq i \leq m_1 \\
\frac{d}{dx} f_1(x), \frac{d}{dx} f_2(x), \ldots, \frac{d}{dx} f_m(x) & \quad \forall m_1+1\leq i \leq m_2 \\
\vdots & \quad \vdots \\
\frac{d}{dx} f_1(x) & \quad \forall m_1+\ldots+m_{m-2}+1\leq i \leq m_{m-1} \\
\end{align*}
\tag{3.61}
\]

Since \(G^t_d\) has dimension \(m\) by assumption, the total number of differentials in this table is \(n-m\). In these differentials, the functions \(\lambda_i\) are the functions that solve the equations (3.47) and all \(m\) output functions are constructed. As stated before, with these output functions it is possible to linearize the input state equations of the corresponding system.

**V Exact linearization of the input-output response**

In the previous section is shown that if a system has a relative degree \(\{r_1, \ldots, r_m\}\) at a point \(x^0\) and

\[r_1 + r_2 + \ldots + r_m = n\]

then this system can be rendered linear by means of a feedback and a change of coordinates. This was called exact linearization of the input-state equations. If the latter condition is not satisfied, but the system still has a relative degree \(\{r_1, \ldots, r_m\}\) at a certain point, one can at least obtain a system whose input-output behaviour is linear. As a matter of fact, we will see in this section that this can already be achieved by means of the (standard) static state feedback:

\[u(x) = -A^{-1}(x)b(x) + A^{-1}(x)v\]

However, the possibility of using feedback in order to achieve linearity in the input-output response is not restricted to systems having a certain relative degree. In this section will be shown what feedback linearization of the input-output mapping exactly means and how a feedback producing a linear input-output behaviour can be designed with the so-called Structure Algorithm.

First a formulation of the meaning of linear input-output behaviour is given. This can be shown on basis of a nonlinear system having relative degree \(\{r_1, \ldots, r_m\}\) on which the feedback (3.45) has been imposed. For this system it is possible (II(5.4.1)) to represent the output response in the form of a Volterra series expansion:

\[y(t) = Q(t,x^0) + \sum_{i=1}^{m} \int_{0}^{t} \omega_i(t-\tau,\tau)\nu(\tau)\,d\tau_i\]

This is an output response in which the first order kernels \(\omega_i(t,\tau)\) depend only on the difference \((t-\tau)\) and not on \(x^0\), and all higher order kernels are vanishing. From this expression of the output response it is easy to see that the output response of the closed loop system is the

\footnote{The input-output behaviour of a nonlinear system of the form (NI) may be represented by means of a series of generalized convolution integrals. One way to do so is the Volterra Series Expansion. The Volterra series expansion describes a causal functional \(u_0, \ldots, u_m\) which is absolutely and uniformly convergent. For further information refer to the literature, especially [II](Ch.3.2).}
sum of the response under zero input (which is a function of time and of the initial state condition only) and of a response depending on the input and not on the initial state, which is linear in the input itself. This property is exactly what is meant by linear input-output behaviour.

Our goal is to construct a feedback in order to achieve, for a broader class of systems than those with a relative degree, a response in which the first order kernels \( \omega_0(t, x_0) \) depend only on the difference \((t-x_0)\) and not on \(x_0\), and all higher order kernels are vanishing. Via a Taylor series expansion of \( \omega_0(t, x_0) \) it is found ([1](3.2.9)) that a necessary and sufficient condition for such a response is that

\[
L_{x_k} L_{x}^j h(x) = \text{independent of } x \quad \forall \, k \geq 0 \quad 1 \leq i, j \leq m
\]  

(3.63)

Note that we will have to satisfy this condition via feedback, because in general it will not be satisfied for a specific nonlinear system. Note also that if the output functions of the system are such that a full order relative degree is obtained, condition (3.65) is automatically satisfied because of (3.47)

A necessary and sufficient condition under which the exact linearizing input-output problem is solvable is proved in ([1](theorem 5.4.2)). This condition is based on a sequence of Toeplitz matrices \( M_k \)

\[
M_k(x) = \begin{bmatrix}
T_0(x) & T_1(x) & \cdots & T_k(x) \\
0 & T_0(x) & \cdots & T_{k-1}(x) \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & T_0(x)
\end{bmatrix} \quad 0 \leq k \leq 2n-1
\]  

(3.64)

with

\[
T_0(x) = \begin{bmatrix}
L_{x_k} L_{x}^j h_0(x) & \cdots & L_{x_k} L_{x}^j h_1(x) \\
\vdots & \ddots & \vdots \\
L_{x_k} L_{x}^j h_0(x) & \cdots & L_{x_k} L_{x}^j h_1(x)
\end{bmatrix} \quad 0 \leq k \leq 2n-1
\]

There exist a solution at \( x_0 \) to the exact linearization problem if and only if for all \( 0 \leq i \leq 2n-1 \), the point \( x_0 \) is a regular point of the Toeplitz matrix \( M_i \) and \( r_\mathbb{R}(M) = r_\mathbb{R}(M)^\prime \).

A mathematical explanation of this condition is very complicated and therefore omitted. Now the Structure Algorithm will be discussed. This algorithm, presented by Isidori, is a test for the fulfilment of the solvability condition as well as a procedure to construct the linearizing feedback. Expressed in a formal statement, the Structure Algorithm tries to find (given a set of

---

3 In this theorem two completely equivalent conditions are proved. Here, only one is mentioned.

4 In which \( r_\mathbb{R}(M) \) is the integer rank(\( M(x_0) \)) and \( r_\mathbb{R}(M) \) is the dimension of the vector space defined by taking linear combinations of rows of \( M \) over the set of real numbers. (This is the special situation in which linear dependence between rows may be tested by taking linear combinations with constant coefficients only.) Clearly the two integers are such that \( r_\mathbb{R}(M) = r_\mathbb{R}(M)^\prime \). If \( M \) is multiplied on the left by a permutation matrix \( V \) of real numbers with the purpose of annihilating the maximal number of rows in \( V^\top M \), then the two integers are equal if and only if the number of nonzero rows in \( V^\top M \) is equal to \( r_\mathbb{R}(M) \).
vector fields $f(x, g_1(x), \ldots, g_m(x))$, a set of real-valued output functions $h_1(x), \ldots, h_m(x)$ and an initial state $x^0$ a neighbourhood $U$ of $x^0$ and a pair of feedback functions $\alpha(x)$ and $\beta(x)$ defined on $U$, such that

$$L_{(g\beta)}^k L_{(g\alpha)}^k h_j(x) = \text{independent of } x \quad \forall \ k \geq 0 \quad 1 \leq i, j \leq m$$  \hspace{1cm} (3.65)

The Structure Algorithm is a recursive algorithm which operates on the sequences (3.64).

**Structure Algorithm**

*Step 1*, let $x^0$ be a regular point of $T_0 = Lh(x)$ and suppose $r_\delta(T_0) = r_\delta(T_\beta)$. Then, there exist a nonsingular matrix of real numbers, denoted by

$$V_i = \begin{bmatrix} P_1 \\ K_1^i \end{bmatrix}$$  \hspace{1cm} (3.66)

where $P_1$ performs row permutations, such that

$$V_i T_\delta(x) = \begin{bmatrix} S_i(x) \\ 0 \end{bmatrix}$$  \hspace{1cm} (3.67)

where $S_i(x)$ is a $r_\delta \times m$ matrix and $\text{rank}(S_i(x)) = r_\delta$. (In case the matrix $T_\delta(x^0) = 0$ then $P_1$ must be considered as a matrix with no rows and $K_1^i$ is the identity matrix.) Conclude the first step by setting:

$$\delta_1 = r_\delta$$
$$\gamma_1(x) = P_1 h(x)$$
$$\overline{\gamma}_1(x) = K_1^i h(x)$$  \hspace{1cm} (3.68)

*Step i*, consider the matrix:

$$T = \begin{bmatrix} L_{\gamma_1^i}(x) \\ \vdots \\ L_{\gamma_{i+1}}(x) \\ L_x L_{\gamma_{i+1}}(x) \end{bmatrix} = \begin{bmatrix} S_{i+1}(x) \\ L_x L_{\gamma_{i+1}}(x) \end{bmatrix}$$  \hspace{1cm} (3.69)

Let $x^0$ be a regular point of this matrix and suppose

$$r_\delta(T) = r_\delta(T)$$  \hspace{1cm} (3.70)

Then, there exists a nonsingular matrix of real numbers, denoted by

$$V_i = \begin{bmatrix} I_{i-1} & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & -I_{i-1} & 0 \\ 0 & 0 & P_i \\ K_i^i & -K_{i-1}^i & K_i^i \end{bmatrix}$$  \hspace{1cm} (3.71)
where \( P_i \) performs row permutations, such that
\[
V_T = \begin{bmatrix} S_i(x) \\
0 \end{bmatrix}
\]  
(3.72)

where \( S_i(x) \) is a \( r_{i+} \times m \) matrix and \( \text{rank}(S_i(x)) = r_{i+} \). Conclude this step by setting
\[
\delta_i = r_{i+} - r_{i-2} \\
\gamma_i(x) = P_i L_i \tilde{y}_{i-1}(x) \\
\bar{y}_i = K_i \gamma_i(x) + \ldots + K_i L_i \tilde{y}_{i-1}(x)
\]  
(3.73)

and note that after each step
\[
S_i(x) = \begin{bmatrix} L_y \gamma_i(x) \\
0 \\
L_y \bar{y}_i(x) \end{bmatrix} \quad L_y \gamma_i(x) = 0
\]  
(3.74)

holds. It may occur that the present step degenerates, i.e. the last \((m-r_{i+})\) rows of the matrix \( T \) depend on the first \( r_{i+} \). If this is the case, then \( P_i \) must be considered a matrix with no rows, \( K_i \) is the identity matrix, \( \delta_i = 0 \) and \( S_i(x) = S_{i+1}(x) \).

If the Structure Algorithm is continued up to the 2\( n \)-th iteration, two possibilities to terminate the algorithm may occur. Either there is a step \( q \leq 2n \) such that the matrix \( T \) has full rank \( m \) at \( x^0 \), or else, from a certain step \( q+1 \) on all further steps are degenerate. In both cases one can still set a function \( \gamma_q \) in the normal way. From the functions \( \gamma_0, \ldots, \gamma_q \) generated by the Structure Algorithm, a linearizing feedback can be constructed in the following way. Set:
\[
\Gamma(x) = \begin{bmatrix} \gamma_1(x) \\
0 \\
\gamma_q(x) \end{bmatrix}
\]  
(3.75)

and recall that \( S_q = L_y \Gamma \) is a \( r_{q+1} \times m \) matrix of rank \( r_{q+1} \) at \( x^0 \).

Then the equations
\[
(L_y \Gamma(x)) \alpha(x) = -L \Gamma(x) \\
(L_y \Gamma(x)) \beta(x) = \begin{bmatrix} I_{r_{q+1}} \\
0 \end{bmatrix}
\]  
(3.76)

are solved on a suitable neighbourhood \( U \) of \( x^0 \) by a pair of smooth functions \( \alpha(x) \) and \( \beta(x) \) which describe the exact linearizing feedback
\[
u = \alpha(x) + \beta(x) v
\]  
(3.77)

satisfying (3.65).

Again (see (3.63)), a sufficient condition for the solvability of an exact linearizing Input-Output feedback is that the system has a relative degree \( \{r_1, \ldots, r_m\} \) at \( x^0 \). If this is the case then the Structure Algorithm may be continued up to stage \( q = \text{max}\{r_1, \ldots, r_m\} \) yielding \( S_q = A(x) \) and \( L \Gamma = b(x) \) resulting in the feedback (3.45).

Concluding the discussion of the Structure Algorithm its importance is stressed, a remarkable shortcoming however, is that the algorithm just provides a "manipulation" of the mathematical model of the system. A "physical" explanation of the algorithm is hard to find.
Chapter 4 MAPLE implementation

Introduction

In this chapter, the implementation -in the symbolic computation system MAPLE- of the algorithms treated in the previous chapter will be presented and discussed. With this implementation an attempt is made to automate the analyses for arbitrary nonlinear (control) systems. Symbolic computation offers an enormous computational facility. With the implementations developed in this thesis, it is made possible to compute (and analyze) symbolically

- the (locally defined) relative degree.
- a change of coordinates leading to the normal form, possibly by solving the set of partial differential equations (3.18).
- the normal form, the zero dynamics and the corresponding unique zeroing input.
- a (locally defined) static state feedback which, if it exists, renders the system linear and controllable.
- a set of output functions which, if they exist, give the system a full order relative degree. If such an output function does not exist, an output function which maximizes the relative degree can be computed, but only for SISO systems.
- a (locally defined) static state feedback which, if it exists, renders the system linear in an input-output sense.

The algorithms are implemented in separate procedures, according to the treatment in Chapter 3. The procedures are written in the MAPLE programming language, mentioned in Chapter 2 and therefore only usable within MAPLE. In most cases, the procedure is a straightforward implementation of the algorithm presented and discussed in Chapter 3, and for the final implementation, how interesting and time-consuming its development may have been, should be referred to the four appendices of this thesis, in which all procedures are treated extensively. Appendix A presents a complete description of the purpose, final results, and the use of each procedure, and contains enough information to work with the procedures. In Appendix B some worked examples are included, in Appendix C flowcharts of some particularly interesting procedures are presented, and in Appendix D complete listings of all procedures are available.

In this chapter some important additional information is collected concerning the relation between the abstract algorithms and the implementation, mathematical background and programming highlights of the implementation, and the use of MAPLE. Some particularly interesting notions will be discussed in subsequent sections. First, a structure in which the procedures can be ordered will be presented. It makes sense to distinguish three categories with respect to the purpose of each procedure. In the following section, the properties of the procedures in each category will be briefly discussed. In this section also some specific implementations will be discussed separately, among which an important extension of MAPLE: the procedures \texttt{psolve} and \texttt{extdsolve} which provide facilities for solving (partial) differential equations in MAPLE, the implementation of the algorithm of \texttt{outputfunc}, the implementation of
the Structure Algorithm in inoutlin, and the implementation of the Zero Dynamics Algorithm in extnormform. These discussions are also supported by a worked example in Appendix B. Finally, in the last section of this chapter, some general remarks on the use of MAPLE are made. Results, difficulties in programming in MAPLE, shortcomings in MAPLE and recommendations for future extension of MAPLE will be briefly discussed.

Classes in which the procedures can be ordered

In total, 19 procedures have been written. These procedures are collected in two packages. The most important analysis tools are collected in the Zerodyn package, while a number of useful computational tools -used extensively by the procedures in zerdyn- are collected in the Tools package. Another division can be made considering the object of each procedure. In this point of view, three categories can be distinguished. In the first class, a number of procedures concerning basic mathematical computations like special differential operators and procedures to solve (sets) of partial differential equations are written. Second, a few existing (standard) MAPLE procedures, mainly concerning Linear Algebra, are adjusted and extended to fit in the present procedures. And finally, a number of procedures concerning the analyses themselves are written, which make extensive use of the procedures mentioned in the other two parts. It is clear that the procedures from the last category are in the Zerodyn package and the procedures from the other categories are in the Tools package. In the following overview these three classes are presented, with the names of the procedures:

- **basic mathematical computations**
  - Idiff, vdiff, liebrack, psolve, extdsolve, involutive

- **extended MAPLE procedures**
  - extrank, extgausselim, extgaussjord, extrowspace, extcolspace

- **procedures to perform the analysis**
  - reldeg, transform, normform, extnormform, statelin, outputfunc, inoutlin

The procedure mklog does not fit in any of these categories, it is a script file which calls the analysis procedures and is an easy way to obtain a complete view of results. (See examples in Appendix B).

Discussion of the implementations

In this section particular details and problems in the implementations, functionality and constraints of the procedures and outlines of implementations are discussed. First, in three subsections, the general properties of the procedures in each of the three categories mentioned above, will be treated. Doing this, often a reference to the appendices will be made. Further, in subsequent subsections, five procedures are discussed in particular. This is because the implementation is rather complicated, the algorithm treated in Chapter 3 is very difficult to comprehend, or the theory is not treated yet in the previous chapter because it is not a part of Nonlinear Control Theory. The following procedures are treated this way, additional supported by an example in Appendix B: psolve/extdsolve, outputfunc, inoutlin, extnormform.
Basic mathematical computations

The procedures in this class are frequently by other procedures in the packages. The procedures \texttt{diff}, \texttt{ndiff}, and \texttt{liebrack} perform differential operations and are straightforward, though powerful symbolic implementations of the differential operations described in \textit{Notations and Terminology}. These implementations are based on the standard MAPLE function \texttt{diff}. The procedure \texttt{involutive} tests the involutivity of a distribution. A lot of \cite{i.e. the faculty of the number of vector fields} rank computations are involved. The symbolic rank computation is not without problems and is discussed in the following subsection. Finally, also the procedures \texttt{psolve} and \texttt{extdsolve} are in this category.

Extended MAPLE procedures

This class contains five procedures, which are based on standard MAPLE procedures concerning Linear Algebra. Few ad-hoc changes are made in order to use these procedures in a broader view. The procedures \texttt{extrank}, \texttt{extgausselim}, \texttt{extgaussjord}, and \texttt{extrowspace/extcolspace} are modified to handle non-rational entries of matrices (like goniometric or exponential functions). The adjustments to provide these extra possibilities are very simple. Just the test for non-rational entries is bypassed and at the appropriate places the MAPLE command \texttt{normal} is added to the command \texttt{normal}. The results of these “new” procedures may not be guaranteed for arbitrary entries (which is obviously the reason why this elementary restrictions are enforced in the standard versions of the MAPLE procedures). In my research, however, the results established with goniometric and exponential entries are very satisfactory and in complete accordance with results already computed in worked examples in literature.

Further, in the procedures \texttt{extgausselim}, \texttt{extgaussjord} some more ad-hoc extensions are provided. An extra parameter is added in which the elements of each column which is used as a pivot in gaussian elimination routines are stored. This extra parameter is very useful in augmentation of a non-square matrix to a full-rank square matrix, an operation which is often performed in the implementations.

The ad-hoc changes in the procedures of this class are all marked with the initials \texttt{(HvE)}, so future releases of MAPLE can be easily patched by users. Finally, it is stressed that apart from the mentioned changes, the procedures are exactly the same as the standard MAPLE procedures.

Procedures to perform the analysis.

The procedures in this class are the main results of this thesis, they offer symbolic computational tools to perform analysis and design strategies for nonlinear (control) systems. The emphasis in these analyses is on an extensive and consistent treatment of the Zero Dynamics and the Exact Linearization Theories. Regarding these two fields of interest, the written procedures can be placed in the following scheme.

From this scheme, two interesting remarks can be made: In the first place the two fields of analysis can be clearly distinguished (columns left and right), but also their common foundation in the relative degree can be recognized. In the second place a clear distinction can be made (dashed line) between the analyses that are applicable on systems for which the relative degree can be defined \textit{(normform, statelin)} and analyses applicable on a broader class of systems \textit{(extnormform, inoutlin)}. As stated in Chapter 3, the results of both approaches are equal \cite{apart from a different coordinate system} in case of a well defined relative degree.
**Zero Dynamics**

- relative degree (reldeg)
  - coordinate transformation to the normal form (transform)

the normal form, the zero dynamics, and the zeroing inputs, obtained via the relative degree and the constraint of an output map forced to be identically zero (normform)

**Exact Linearization**

- computation of an output map for which the system obtains a full order relative degree (outputfunc)
  - design of an exact input-state linearizing feedback, obtained via the relative degree (statelin)

The implementation of the procedures `outputfunc`, `inoutlin`, and `extnormform` is discussed in subsections in the remainder of this section. For further information on the other procedures mentioned in the scheme please refer to the appendices. For all these procedures (except for `reldeg`) is a flowchart available in Appendix C.

**psolve and extdsolve**

In this subsection the procedures `psolve` and `extdsolve` are discussed. `psolve` tries to construct solutions of a special system of partial differential equations of the first order. First the solvability of this system is determined. `extdsolve` provides a strategy to solve a set of ordinary differential equations (and is used within `psolve`). MAPLE does not provide facilities to solve partial differential equations and has difficulties with solving sets of ordinary differential equations. Solutions of partial differential equations are part of the algorithms in Chapter 3, and therefore they are developed in this thesis. It is stressed that the procedures `psolve` and `extdsolve` are extensions of MAPLE and can be used in a general way to solve (partial) differential equations.

The algorithm which solves the equations is based on the *Frobenius theorem*. In fact the solutions are constructed by means of a constructive proof of this theorem. This proof is treated...
in [1](1.4). In this section, the algorithm which performs the solvability test and the construction of the solutions is treated in a theoretical manner. In example 5 in Appendix B, the implementation of the algorithm in MAPLE is explained with an example.

Consider the partial differential equation that is given by

$$\frac{\partial \lambda_i}{\partial x}(g_1(x) \ldots g_d(x)) = \frac{\partial \lambda_i}{\partial x}g(x) = 0 \quad (4.1)$$

Where \(g_1 \ldots g_d\) are smooth vector fields and matrix \(g(x)\) has rank \(d\) at a point \(x^0\).

**Part I: solvability**

Consider the (corresponding) smooth, nonsingular distribution \(\Delta\), defined on an open subset \(U\) of \(\mathbb{R}^n\), with dimension \(d\), spanned by the vector fields:

$$\Delta(x) = \text{span}\{g_1(x), \ldots, g_d(x)\} \quad (4.2)$$

The codistribution \(\Omega = \Delta^\perp\) is also smooth and nonsingular, has dimension \(n-d\), and is (locally around \(x^0\)) spanned by \(n-d\) covector fields \(\omega_1 \ldots \omega_{n-d}\). Each covector field solves the equation:

$$\omega_j(x)g(x) = 0 \quad \forall \; x \in U^0 \quad \forall \; 1 \leq j \leq n-d \quad (4.3)$$

This equation can be simply regarded as a linear homogeneous equation. The solutions form a basis of the null space of the matrix \(g(x)\).

Now the partial differential equation (4.1) itself is considered. The problem is essentially the same, but instead of accepting any solution of (4.3) only solutions of the form

$$\omega_j = \frac{\partial \lambda_j}{\partial x} \quad (4.4)$$

for suitable real-valued functions \(\lambda_j\) are allowed. Suppose we try to find \(n-d\) independent solutions \(\lambda\). By independent is meant that the row vectors

$$\frac{\partial \lambda_1}{\partial x}, \ldots, \frac{\partial \lambda_{n-d}}{\partial x} \quad (4.5)$$

are independent at \(x^0\).

The solutions \(\lambda\) are differentials of real-valued functions. This means that the annihilator \(\Delta^\perp\) should be spanned such differentials. This is the case when the distribution \(\Delta\) is said to be completely integrable or equivalent (from the Frobenius Theorem, Notation and Terminology), when the distribution \(\Delta\) is involutive. This result can be summarized by: There exist \(n-d\) independent solutions for the partial differential equation (4.1) if and only if the corresponding distribution \(\Delta\) is involutive.

**PART II: construction of solutions**

Let \(g_{d+1}, \ldots, g_s\) be a complementary set of vector fields, defined on \(U^0\), with the property that

$$\text{span} \{g_1(x), \ldots, g_s(x)\} = \mathbb{R}^s \quad (4.6)$$

at each \(x\) in \(U^0\). Further, let \(\Phi_t(x)\) denote the flow of the vector field \(g\), i.e. the smooth function of \(x\) and flow variable \(t\) with the property that the function \(x(t) = \Phi_t(x^0)\) solves the ordinary differential equation

$$\dot{x} = g(x) \quad (4.7)$$

with initial condition \(x(0) = x^0\). Moreover, the flow of the vector filed \(g\) also satisfies
In [I] is proved that solutions of (4,1) can be constructed by taking an appropriate composition of the flows associated with the vector fields \( g_1, \ldots, g_n \). This composition is a mapping \( F = (x_1, \ldots, x_n) \mapsto (z_1, \ldots, z_n) \) where \( z_1, \ldots, z_n \) are the flow variables for each vector field \( g_i \) as in

\[
\Phi^i_t(x) = g(\Phi^i_t(x)) \quad \Phi^i_0(x) = x
\]  

(4.8)

Because the mapping \( F \) is a diffeomorphism on its image \( U^0 \) (I1), the inverse mapping exists and is also a smooth mapping defined on \( U^0 \): \( F^{-1} = (z_1, \ldots, z_n) \mapsto (x_1, \ldots, x_n) \). It is proved that the last \( n-d \) components of the inverse mapping solve the partial differential equation (4,1). In example 5 in Appendix B the composition of the mapping \( F \) and its inverse mapping will become clear from a worked example.

In essence, the partial differential equation is transformed to \( n \) systems of \( n \) ordinary differential equations. The flow is a representation of the solution of such a system of ordinary differential equations. Subsequent solving each flow and composing the mapping \( F \) will construct solutions of the partial differential equation via the inverse mapping \( F^{-1} \). This is where the procedure \textit{extdsolve} comes up. This procedure provides a strategy to solve a set of ordinary differential equations by subsequently solving the separate equations in order of increasing complexity and substituting the results in the next equation to be solved. The procedure \textit{extdsolve} is designed to compute the flow of a vector field. Also the strategy of \textit{extdsolve} will be explained in example 5.

\textit{outputfunc}

The procedure \textit{outputfunc} tries to find output functions \( \lambda_i \) so that the system gets a full order relative degree. The algorithm supporting this procedure is treated in Chapter 3, section IV. At this place an outline of the implementation is discussed. Additional an elaborated example is included in Appendix B, example 6.

Essentially, the implementation consists of two parts. In the first part the distributions (3,50) are created and the corresponding tests are evaluated. If the system fulfills the conditions for the existence of the demanded output functions, after this first part all relevant distributions are set. Also an integer \( krel \) is set. In the second part the actual construction of the output functions is performed in \( krel \) steps. If, however, the tests are not passed, the first part is interrupted. For SISO systems only an output function is constructed which maximizes the relative degree, for MIMO systems the algorithm is terminated.

Considering the previous theory, the first part of the implementation is rather clear. The structure of the implementation is shown in the flowchart (Appendix C) and conveniently arranged in example 6, (Appendix B). There are two interesting elements in this implementation. In the first place the simplification of "distributions" is worth mentioning. This simplification is performed by replacing the corresponding matrix by a matrix containing basis vectors of the vector space spanned by the columns of the original matrix. These basis vectors are computed by gaussian elimination in the MAPLE procedure \textit{extcolspace}. These simplifications provided an enormous computational ease in testing dimensions and involutivity of the distributions (3,50). Occasionally, however, this simplification method fails because terms like \( 1/x_i \) are easily introduced by the gaussian elimination method of \textit{extcolspace}. These cause division by zero when, for example, the initial value of \( x_i \) is chosen zero. In such cases the simplification is omitted.

In the second place the computation of an output function which maximizes the relative degree of a SISO system is rather interesting. (I1)(theorem 4.8.2). This computation is essentially
analogous, with the modification that, if a new computed distribution turns out to be not involutive -and the MIMO algorithm is terminated- this distribution is replaced by its involutive closure (inv) and the algorithm is continued. Finally, an integer \( \nu \) can be obtained for which

\[
\dim(\text{inv}(\text{span}(g, \text{ad}_1g, \ldots, \text{ad}_{\nu-1}g))) \leq n
\]

\[
\dim(\text{inv}(\text{span}(g, \text{ad}_1g, \ldots, \text{ad}_{\nu-1}g))) = n
\]

The output function \( \lambda \) is found by solving the partial differential equation

\[
d\lambda(x)\left(\text{inv}(\text{span}(g, \text{ad}_1g, \ldots, \text{ad}_{\nu-1}g))\right) = 0
\]

The involutive closure is obtained by adding the appropriate vector field(s) to the distribution and then simplify the matrix which represents the distribution with the procedure \text{extcolspace} as above. The partial differential equation is solved by the procedures \text{psolve} and \text{extdsolve}.

The second part of this procedure is a straightforward implementation of the algorithm presented in Chapter 3, section IV. In example 6 also this part of the algorithm is explained.

\textit{inoutlin}

The procedure \textit{inoutlin} is an implementation of the so-called Structure Algorithm. This algorithm, presented in Chapter 3, section V, has successfully been implemented. A worked example in Appendix B explains this method, the example can be easily attended with the aid of the discussion of the algorithm in Chapter 3 and the flowchart in Appendix C.

Some particular interesting details of the implementation can be mentioned. (1) The test \( r_N(T) = r_T(T) \) (3.70) is not evaluated in the implementation. In the procedure \textit{inoutlin} this test is rather implicit evaluated by simply testing wether matrix \( T \) has full rank and wether the elements computed in permutation matrix \( V \) are real constants. Further, (2) the computation of the matrix decompositions (3,67) and (3,72) is assisted by the (MAPLE) function \textit{kernel}, which is able to compute the nullspace of a vector space. This procedure is very powerful and often used, also in the other procedures of the \textit{Zerodyn} package.

\textit{extnormform}

The procedure \textit{extnormform} is an implementation of the \textit{Zero Dynamics Algorithm}, presented in Chapter 3, section III. A worked example in Appendix B explains this method, the example can be easily attended with the aid of the discussion of the algorithm in Chapter 3 and the flowchart in Appendix C. It should be stressed that this implementation is not fully completed yet: A few problems are still present in this implementation. It is, however, expected that these deficiencies can be solved without great difficulties. This is left for future expansion of the procedure due to lack of time.

A first problem regards to the (locally) output zeroing submanifold (3,39), which is constructed by the algorithm in Chapter 3. This submanifold is implemented as a list (an ordered sequence) of restrictions formulated in the state elements. This submanifold is "augmented" every step with new restrictions, till it is maximal. Only new (i.e. different) restrictions are added, restrictions which are already computed in previous steps are neglected. A problem is encountered when this submanifold is to be substituted in an expression, for example in (3,40). The order in which the restrictions are substituted does often matter, while one can not easily automate this substitution order. For instance, the submanifold may contain (among other terms)
the following restrictions: \[ x_1 = x_2 ; \quad x_3 = 0 \]. An obvious result is that \( x_1 = x_2 = 0 \), but it is clear that the substitution order does matter. Intricate restrictions and a larger number of restrictions in the submanifold often change this problem for the worse. It is stressed that simultaneous substitution of all restrictions is not an adequate solution.

A second problem concerns the constraints which may be caused by the neighbourhood \( U_k \) (3,35) for which the output zeroing submanifold is smooth (for each step \( k \)). At the moment, these constraints are computed and are displayed, but they are not integrated in the final results of the procedure. Therefore it is left to the user to draw conclusions on the validity of the results.

**Discussion of the use of MAPLE**

Without too much difficulties, it has been possible to implement the algorithms of Chapter 3 directly in the MAPLE programming language. Doing this, a number of existing (standard) MAPLE functions have been used, mainly relating to Linear Algebra, array manipulation, differentiation and integration, and procedures to solve (differential) equations. Also some extensions of MAPLE -specially defined computational tools- has been found necessary. Of course, a lot of problems were encountered. In this section, the use of MAPLE, some difficulties in programming in MAPLE, constraints and shortcomings in MAPLE, and recommendations for future extension of MAPLE and the procedures developed in this thesis will be discussed. In subsequent subsections, results established so far, differential equation in MAPLE, parameters in MAPLE, and evaluation of results in MAPLE will be treated.

**Results established so far**

The results established so far with the procedures seem to be very promising. In Appendix B eight examples are included which provide a rough overview of the possibilities of the procedures. It is stressed, however, that additional research will be required to find out the limits of the procedures and the use of MAPLE.

Concerning the limits of the procedures, these may be in the implementation. Occurring errors, can often be corrected by an appropriate extension of the procedure, which provide possibilities for the special problem that is encountered. Of course, and most likely, the limits may also be in the algorithm, or in the properties of the system that is being analyzed. In most of the cases an suitable ERROR message is returned, provided by either MAPLE or the procedure itself. Concerning the limits of MAPLE, little can be said. The built-in possibilities of MAPLE are strictly limited (e.g. not all systems of equations can be solved), and the computation will also fail when the computer runs out of memory. This occurs only if symbolic computations are performed on extremely large data structures or data structures which contain extremely large expressions.

In general, the MAPLE computations are surprisingly fast. (runtimes -on the Sun Spare- are often below 60 seconds for rather complicated computations), though the efficiency of the implementations has never been a topic developing the procedures. In fact very little is known from efficiency of symbolic computations.
Differential equations
One of the most prominent shortcomings of MAPLE, encountered programming the implementations of the analyses in this thesis, is that MAPLE is not capable to solve partial differential equations and has serious difficulties in solving sets of ordinary differential equations. By means of an implementation of the Frobenius theorem, an extension of MAPLE is provided -psolve and exdsolve, discussed in the previous section- which satisfies the demands of the present implementations. Being clear that these extensions cover some special cases and are far from being perfect, it is highly recommended that the theories of these and other strategies are further analyzed in the literature and are added to MAPLE in a generalized form, in harmony with the (internal) MAPLE semantics.

Use of parameters
In MAPLE two forms of parameters can be distinguished. Global and local parameters. The difference between these is that local parameters can only be declared within procedures, they do not have any influence on the parameters outside the procedure. The vast majority of parameters used in the procedures are declared to be local. At some specific places, in very few procedures, however, some global parameters are introduced. These are only used when they can hardly be defined local, because an unknown number of parameters is to be used, e.g. arbitrary number of inputs.
I am quit aware of the fact that the use of such parameters in programming is not done. Also, I am convinced that this could have been avoided after further analysis. The main purpose, however, was to provide a working implementation of some analyses described in a rather mathematical way and which structure I wanted to retain recognizable in the implementations. Maybe the concerning procedures can be adjusted in a later stage. Except for the easy structure of the implementations, an obvious advantage of the present implementation is that these parameters are easily available for analyses after computation. A serious disadvantage is that there may be a small chance of conflicts with names defined by the user. In order to avoid these problems, the "global" parameters have, without exception, a name which starts with the prefix loc. When users avoid this prefix in their own names, conflicts will not occur. After all, these "global" parameters become unevaluated after their use by means of an assignment with their own name.

Evaluation of results
In order to make the results of MAPLE as reliable as possible, a lot of tests are implemented which check the (interim) results of MAPLE. Especially results from the MAPLE procedures solve (solutions of systems of (nonlinear) equations, for example in order to construct inverse mappings), dsolve (solutions of ordinary differential equations), and diff (differentiation) are tested. MAPLE results are not always easy to check. In this subsection some remarks on the important point of evaluation are made.

- Long, intricate expressions
Inherent to the use of symbolic computation is the creation of results which contain very long, intricate expressions. Even in the rather simple examples of Appendix B this becomes clear. The clarity and evaluation possibilities of such results are seriously restricted.
A way to (partly) avoid these problems, is to use numeric system parameters as much as possible. The advantages of symbolic computation (analytic analysis, fair insight in the results etc.) are completely retained, but the clarity of results will be significantly improved. Of course, in most cases it is also possible to substitute the numeric values for symbolic system parameters afterwards.
Problems with evaluation of expressions in MAPLE

The solutions which MAPLE computes are often not explicitly available, but only in the form of sets of equations. For example: The solution of the equation \(2x = 4\) is given by the set \(\{x = 2\}\). In order to test this solution it is not possible to assign \(x := 2\), because then the original equation \(2x = 4\) does not exist any more - \(x\) is evaluated to 2 at all places - while this equation might still be needed.

The only possible way to check the solution is to substitute the set of solutions \(\{x = 2\}\) in the set of equations \(\{2x = 4\}\) and to evaluate whether or not this equation is an equality. Providing this substitution and evaluation, several problems are encountered, which sometimes causes the impossibility of testing the solutions:

- In the case of substitution, some obviously equal terms appear to have a different notation or are otherwise not recognised as equal by the MAPLE substitution command. This is a fair problem because the user does not always have insight in the way MAPLE handles its parameters in internal computation. The syntax of MAPLE's datastructures is not always transparent to the user.\(^1\)

- In case of evaluation, a simple Boolean test of the equality after substitution, e.g. \(\{2*2 = 4\} \text{ true or false}\), is only in very simple cases possible. In case of intricate or long expressions at one of both sides, often an error message is returned. A way to bypass this problem is to separate both sides of the equality and subtract them. The result can be excessively simplified and should be identically zero. In practice however, it is a serious problem to, in all occurring situations, separate the both sides of an equality.

Concluding, most problems considering the evaluation of expressions seem to be solved. A last problem, however, is the evaluation of the results of \texttt{extdsolve}: checking the flows computed by \texttt{extdsolve}. In the present implementation this test often fails while the results computed by MAPLE seems to be correct. It is expected that this test can be improved after further analysis of \texttt{extdsolve} and MAPLE. Moreover, also for the other tests, satisfactory results in all occurring situations can not be guaranteed. In cases where a test fails it is recommended to bypass the test (in the source text) and have the results checked in a later stage. It is stressed that results computed by MAPLE may be correct, but it is only the test that fails!

* Numerical evaluation

Numerical evaluation of results computed by MAPLE is expected to provide a good alternative of symbolic evaluation. Numerical evaluation is possible within or outside MAPLE (e.g. MATLAB, DSTOOLS) and will provide additional advantages:

- The creation of clear graphs by means of numerical simulations gives more insight in the results.

- Numerical simulation may provide a first step in additional analysis, for instance of the stability of the symbolically computed zero dynamics.

\(^1\) For example, the equivalent notations \(\text{diff}(x(z), z)\) and \(D(x)(z)\) are different in case of substitution. Another example, the only way to be able to substitute the elements of \texttt{zeta} and \texttt{eta} (names) within the procedure \texttt{normform} is to carry the "empty" (unassigned) parameters \texttt{zeta} and \texttt{eta} via the parameter list of \texttt{transform}. It is certainly not enough to declare the "empty" arrays \texttt{zeta} and \texttt{eta} in \texttt{normform} as local parameters.
Chapter 5 Conclusions and Recommendations

Conclusions

- The collection of procedures developed in this thesis forms a valuable contribution to the development of systematic design methodologies for nonlinear control systems.

- The procedures offer powerful computational tools which provide possibilities for symbolic analysis and design of nonlinear control systems. Together, the procedures provide a complete and consistent treatment of two important fields of interest in the (analytic) analysis and design of nonlinear control systems: the exact linearization theories and the zero dynamics.

- It is made possible to compute and analyze symbolically -for a class of (MIMO as well as SISO) systems for which the relative degree can be defined- the relative degree, the normal form, the zero dynamics, an exact linearization of the input-state equations, and an exact linearization of the input-output behaviour. Also for a broader class of systems the zero dynamics and the two forms of exact linearization can be computed symbolically. In this broader class, the systems which do have a relative degree are fully incorporated.

- The results established so far with the procedures seem to be promising, though their value should be proved in practice, when elaborate tests with realistic systems are being performed.

- Following from the successful implementation in the MAPLE programming language, the Symbolic Computation System MAPLE seems to be a suitable environment for the symbolic application and development of analytic control strategies. The MAPLE syntax, however, seems to cause some small difficulties, but it is expected that these can be anticipated in neat programming.

- Symbolic Computation is found to be a powerful research tool, which importance for science and educational purposes is stressed.

- An important shortcoming in MAPLE is found to be the inability of solving partial differential equations. For application in the procedures developed in this thesis, these deficiencies are redressed by appropriate extension of MAPLE. It is highly recommended that these extensions are further analyzed and added to MAPLE in a generalized form.
Conclusions and Recommendations

Recommendations

Provide extensions of the existing procedures by:

- Implementation of additional control objectives: (Also in order to shift the attention from strict analysis to more complete synthesis of control problems.)
  * (Asymptotic) stabilization via static state feedback ([1],[2]).
  * (Asymptotic) tracking ([II]).
  * Disturbance attenuation ([I]).

- Improvement of the possibilities for (symbolic) evaluation of the MAPLE results.

- Providing additional possibilities for (partly) numerical evaluation of the MAPLE results.

- Improvement of the user-friendliness of the procedures:
  * Offering choices whether or not some tests or parts of the program should be performed.
  * Optimization of ERROR messages, in order to return as much information as possible.
  * Providing facilities for easy data exchange between MAPLE and (numerical) programs like MATLAB.

Perform elaborate tests on the existing procedure with a number of different system models, in which also realistic models should be incorporated.

- Evaluation of results and occurring ERRORS, introduce improvements.

- Evaluation of the control strategies introduced in this thesis and implemented in the procedures when they are imposed on real control problems.

Study "new" control strategies, evaluate them with respect to symbolic computation and eventually fit them into the existing implementations.

- Approximate exact linearization. A strategy which may provide satisfactory results in case the system can not be exact linearized. The system is approximated to a system which is as close as possible to the original, but is exact linearizable. ([II]).

- Dynamic state feedback ([I]).
References

[1] Isidori, Alberto; 

[2] Byrnes, Christopher and Isidori, Alberto; 

[3] Henson, Michael and Seborg, Dale; 


[9] Isidori, A.; 


Appendix A  Description of procedures

In this appendix a description of the purpose, calling sequence and results of all procedures in the packages zerodyn and tools, written in the MAPLE programming language is summarized. Respectively, the zerodyn and the tools package get a chance. The order of treatment is the same as the order of the listings of both packages. The description presented for each procedure should contain enough information to work with the procedure. For further information on the algorithms and theoretical backgrounds of the functions refer to Chapter 3 and the book [1] of Isidori. Some specific information on the use of MAPLE and the procedures described in this appendix can be found in Chapter 4. Of course information on a particular function can also be found in the header of the procedure in the listings in Appendix D or in the corresponding flowcharts which are collected in Appendix C.

Contents

Structure scheme  A.2
Required model  A.2

Zerodyn package  A.3
  reldeg  A.3
  normform  A.4
  extnormform  A.6
  statelin  A.6
  inoutlin  A.7
  outputfunc  A.7

Tools package  A.8
  ldiff  A.8
  vdiff  A.9
  liebrack  A.9
  extrank  A.9
  extgausselin  A.9
  extgaussjord  A.10
  exttoolspace/extrowspace  A.10
  transform  A.10
  psolve  A.11
  extsolve  A.12
  involutive  A.12
  mkmatr/mkmatc  A.13
  mklog  A.13
Structural scheme

The structure in which the descriptions are presented is for all procedures the same and corresponds highly to the MAPLE notations.

- **procedure name:**
- **calling sequence:**
  (sometimes several calling sequences are presented)
- **synopsis:**
- **input parameters:**
- **output parameters:**

Required model

The implementations can be applied to nonlinear SISO and MIMO models, which should be described in the state space form that is considered throughout this thesis, and which only constraint is that the number inputs $m$ is equal—for some procedures the constraint larger than or equal is allowed—to the number of outputs $p$.

\[
\dot{x} = f(x) + \sum_{i=1}^m g(x)u_i \\
y_1 = h_1(x) \\
... \\
y_p = h_p(x)
\]

Of course the model that will be analyzed should satisfy a certain notation. The nonlinear state space model should be described in a MAPLE form by:

\[
\dot{x} = f(x) + g(x)u \\
y = h(x)
\]

Where the states $x_1, x_2, \ldots, x_n$ are unassigned elements of the $n$-dimensional array $x$, so they are referred to as $x[1], \ldots, x[n]$ and $f(x), g(x), h(x)$ are respectively defined like a $n$-dimensional, a $n \times m$-dimensional and a $m$-dimensional array whose entries are functions of the state elements $x[1], \ldots, x[n]$. It should be clear that the columns of the matrix $g$ are equal to the vector fields $g_1, \ldots, g_m$. In MAPLE procedures the following data structures are defined:

\[
x := \text{array}(1..n,1) \\
f := \text{array}(1..n,1,(1)=\ldots, (2)=\ldots, (n)=\ldots)) \\
g := \text{array}(1..n,1..m,1,(1,1)=\ldots, ..., (n,m)=\ldots)) \\
h := \text{array}(1..m,1\ldots))
\]

A workpoint $x^0 = xnull$ in the state space should be defined in the following set:

\[
xnull := \{x[1] = \ldots, x[2] = \ldots, \ldots, x[n] = \ldots \}
\]

Where at the dots it is possible to put a numeric value, as well as an unevaluated name.

Always where $x, xnull, f, g, h$ are mentioned, the above data structures and state space model are meant. Of course they do not need to have the same name.
zerodyn package version 1.0

This package provides facilities to symbolically compute and analyze the relative degree, the normal form, the zero dynamics and corresponding zeroing inputs of nonlinear SISO and MIMO systems. Also two static state feedbacks which (locally) linearize the nonlinear systems, and (dummy) output functions for which the system gets a full order relative degree can be computed.

The following procedures are available:

- **reldeg**: Computation and analysis of the relative degree in a working point \( x^0 \) of the state space.
- **normform**: Computation of a local state transformation to a normal form, explicit rendering of the system dynamics in normal form, computation of the zero dynamics and the zeroing inputs.
- **extnormform**: Computation of the zero dynamics for a broader class of systems. Also systems which do not have a (vector) relative degree are incorporated.
- **statelin**: Computation of a static state feedback and a state transformation to derive (locally) exact linearized input-state equations: the closed-loop system in the new coordinates can be described as linear and is fully controllable.
- **inoutlin**: Computation of a static state feedback to derive (locally) exact linearized input-output behaviour.
- **outputfunc**: Computation of (dummy) output function(s) for which the system gets a full order relative degree.

**procedure reldeg**, reldeg(f,g,h,x,xnull,'rdeg','rto','Adet','bdet','cond')

reldeg computes the (vector) relative degree of SISO systems and MIMO systems with a number of inputs larger than or equal to the number of outputs. Optional, some useful matrices and initial state conditions for which the (local) relative degree is not valid because the matrix Adet is singular are returned. It may occur that the relative degree is not well defined at the chosen working point xnull (see Chapter 3). In that case the function value returns a sequence of such conditions, which may be incomplete.

**input parameters:**

- **f**: state vector function \( f(x) \), a representation of the smooth vector field \( f \), defined in MAPLE as an \( n \) dimensional array.
- **g**: input matrix function \( g(x) \), a representation of the smooth vector fields \( g_i \) with \( i=1 \ldots m \), defined in MAPLE as an \( n \times m \) dimensional array.
- **h**: output vector function \( h(x) \), a representation of the smooth vector field \( h \), defined in MAPLE as an \( p \) dimensional array.
- **x**: state space variable vector, defined in MAPLE as an unassigned \( n \) dimensional array.
- **xnull**: working point in the state space, defined in MAPLE as a \( n \) dimensional set: \{ \[ x[1]=... \ldots , x[n]=... \} \}
Appendix A  Description of the procedures  A.4

\[ \dot{x} = f(x) + g(x)u \]
\[ y = h(x) \]

output parameters (optional in successive order):

- **rdeg**: array which contains the (vector) relative degree
  \[ rdeg = [r_1, \ldots, r_p] \]
- **rrot**: sum of the (vector) relative degree
- **Adeg**: array which contains the non-singularity matrix (3,2)
  \[
  Adeg = \begin{bmatrix}
  L_1 L_i^{r_i-1} h_1(x) & L_k L_i^{r_i-1} h_k(x) \\
  L_1 L_i^{r_i-1} h_2(x) & L_k L_i^{r_i-1} h_k(x) \\
  \vdots & \vdots & \vdots \\
  L_1 L_i^{r_i-1} h_p(x) & L_k L_i^{r_i-1} h_k(x)
  \end{bmatrix}
  \]
- **bdeg**: array which contains the column (3,24)
  \[
  bdeg := \begin{bmatrix}
  L_i^2 h_1(x) \\
  \vdots \\
  L_i^2 h_p(x)
  \end{bmatrix}
  \]
- **cond**: condition(s) on the initial state for which the matrix Adeg turns out to be singular and where the relative degree is no longer valid.

**procedure normform**

- `normform(f, g, h, x, xnull, 'normdyn')`
- `normform(f, g, h, x, xnull, 'fn', 'gn')`
- `normform(f, g, h, x, xnull, 'normdyn', 'zerodyn', 'uzero')`
- `normform(f, g, h, x, xnull, 'fn', 'gn', 'zerodyn', 'uzero')`
- `normform(f, g, h, x, xnull, 'normdyn', 'fn', 'gn', 'zerodyn', 'uzero')`

normform computes a state transformation to a normal form. This normal form can be computed in two different ways. A normal form does only exist if the system has a (vector) relative degree. Also, the zero dynamics and the zeroing input(s) can be computed. normform is suitable for SISO systems and MIMO systems with a number of inputs larger than or equal to the number of outputs.

In the first way to compute the normal form equations, (the computation of 'normdyn' via the 1st, 3rd or 5th call), the state transformation is successively substituted in the left-hand and the right-hand sides of the original equations, after which the necessary differentiation is performed and both sides are combined. The second way, (the computation of 'fn' and 'gn' via the 2nd, 4th or 5th call), is computationally extremely less demanding and makes use of the jacobian of the transformation mapping to circumvent the differentiation. So the second way is preferable. However, if both ways are computed (5th call), the MAPLE results will be automatically compared and checked so the results will have a better reliability.

input: `f, g, h, x, xnull` (for further information see reldeg)
Appendix A. Description of the procedures

output:

**normdyn:** n dimensional set which contains the system equations in normal form in the following way:

\[
\begin{align*}
\dot{\zeta}_1 &= \ldots, \ldots, \eta_{n-\text{rtot}} \\
\dot{\zeta}_2 &= \ldots, \ldots, \eta_{n-\text{rtot}} \\
&\vdots \\
\dot{\zeta}_{\text{rdeg}[1]} &= b_{\zeta}[1] + \sum_{j=1}^{m} a_{\zeta}[1,j] \ast u[1] \\
\dot{\zeta}_{\text{rdeg}[1] + 1} &= \text{normdy}[1] + 2 \\
&\vdots \\
\dot{\zeta}_{\text{rdeg}[1] + \text{rdeg}[2]} &= b_{\zeta}[2] + \sum_{j=1}^{m} a_{\zeta}[2,j] \ast u[2] \\
&\vdots \\
\dot{\zeta}_{\text{rdeg}[1] + \text{rdeg}[2]} &= b_{\zeta}[m] + \sum_{j=1}^{m} a_{\zeta}[m,j] \ast u[m] \\
\end{align*}
\]

**etadyn:** n-rtot dimensional set which contains the explicit zero dynamics of the system in the following way:

\[
\begin{align*}
\dot{\eta}_1 &= q_1(\eta,\zeta) + p_1(\eta,\zeta) \ast u_0 \\
&\vdots \\
\dot{\eta}_{n-\text{rtot}} &= q_{n-\text{rtot}}(\eta,\zeta) + p_{n-\text{rtot}}(\eta,\zeta) \ast u_0 \\
\end{align*}
\]

**uzeo:** array which contains the corresponding unique input that keeps the output zero

\[
\begin{align*}
\text{fn, gn} &\quad \text{arrays which contain the n dimensional vector and n x m matrix which describe the dynamics of the system in normal form:} \\
\dot{x} &= \text{fn}(\zeta) + \text{gn}(\zeta)u \\
\text{where } \zeta \text{ stands for the new state vector}
\end{align*}
\]
**procedure extnormform**

extnormform computes the zero dynamics and the corresponding zeroing input(s). The algorithm used is the Zero Dynamics Algorithm of Isidori (Chapter 3), in which a locally maximal output zeroing submanifold is computed. This algorithm can compute the zero dynamics even if the system does not have a (vector) relative degree. An additional result of this algorithm is that a coordinates transformation to an extended normal form can be computed. This Extended Normal Form is not yet implemented, and it is stressed that also the implementation of the computation of the zero dynamics in this algorithm is still in a developing stage. extnormform is suitable for SISO systems and MIMO systems with an equal number of inputs and outputs.

input: \( f, g, h, x, xnull \) (for further information see reldeg)

output:
- **zerodyn**: set which contains the explicit zero dynamics of the system in the following way:
  \[
  [ \dot{x}[1] = \ldots, \ldots, \dot{x}[n] = \ldots ]
  \]
- **uzero**: array which contains the corresponding unique input that keeps the state evolving on the maximal locally output zeroing submanifold and keeps the output identically zero.

**procedure statelin**

statelin computes the exact linearizing static state feedback \( u \) and the corresponding state transformation which linearize the input-state equations of a nonlinear SISO or square MIMO system under the condition of a full order relative degree: There must exist a(n) (set of) output function(s) for which the sum of the (vector) relative degree is equal to the dimension of the state space of the system, i.e. \( r_{tot} = n \). The state transformation that is used is exactly equal to the transformation to the normal form and is (internally) computed by the procedure transform. The new state coordinates are called \( \zeta \). The closed loop system in the new coordinates is linear and controllable.

input: \( f, g, h, x, xnull \) (for further information see reldeg)

\( v \) : (symbolic) new reference input \( v \), an \( m \)-dimensional vector which may be an unassigned name

output:
- **u**: (array) the linearizing feedback \( u(x) = \alpha(x) + \beta(x) \ast v \)
- **Az**: (array) vector which is the linear system matrix multiplied with the new state vector \( \zeta \) of the closed-loop system
- **Bz**: (array) input matrix of the linearized closed-loop system

\[ \dot{\zeta} = Az(\zeta) + Bz \ast v \]

(optional):
- **alpha**
- **beta**

arrays containing the linearizing feedback \( \alpha(x) \) and \( \beta(x) \)
Appendix A

Description of the procedures

procedure inoutlin, inoutlin(f,g,h,x,xnull,v,'u','0','gl')
inoutlin(f,g,h,x,xnull,v,'u','0','gl','alfa','beta')

inoutlin computes a static state feedback so that the closed loop system obtains a linear input-output behaviour (Chapter 3). The feedback functions alfa(x) and beta(x) are constructed by the so called Structure Algorithm. Also the system equations of the linearized closed-loop system are returned. inoutlin is suitable for nonlinear SISO and square MIMO systems.

input: f, g, h, x, xnull (for further information see reldeg)
v : the new reference input v, the dimensions of the array v should equal those of the original input u, v may be unassigned.

output:
u : (array) the linearizing feedback \( u(x) = alfa(x) + beta(x) \times v \)
fl, gl : (arrays) the dynamics of the (linearized) closed loop system.
\[ \dot{x} = fl(x) + gl(x)u \]

(al)alfa : arrays containing the linearizing feedback alfa(x) and beta(x)

procedure outputfunc, outputfunc(f,g,x,xnull,'lambda')

outputfunc tries to find a(n) (set of) dummy output function(s) lambda(i) so that the system gets a full order relative degree. outputfunc is suitable for SISO systems and square MIMO systems. If such a function does not exist, for SISO systems only, an output function (in some cases several functions) is searched for that maximizes the relative degree, which will be, however, strictly less than the dimension of the state space.

input: f, g, x, xnull (for further information see reldeg)

output: lambda : array containing output function(s)
tools package version 1.0

The tools package version 1.0 belongs to the zerodyn package version 1.0 and contains some useful and necessary tools for the procedures in zerodyn. The tools package contains the following procedures:

- **ldiff**: computes the directional derivative of a scalar valued function along a vector field
- **vdiff**: computes the directional derivative of a vector valued function along a vector field
- **liebrack**: computes the Lie-derivative or Liebracket of two vector fields
- **extrank**: extension of the standard MAPLE procedure 'rank'
- **extgausselim**: extension of the standard MAPLE procedure 'gausselim'
- **extgaussjord**: extension of the standard MAPLE procedure 'gaussjord'
- **extcolspace/ extrowspace**: extension of the standard MAPLE procedures 'clospace'/'rowspace'
- **transform**: computes a state space transformation to a normal form
- **psolve**: computes a solution to a (set of) partial differential equations.
- **extdsolve**: this procedure provides a strategy to solve a set of ordinary differential equations on the basis of the standard MAPLE procedure dsolve.
- **involutive**: test for an involutive distribution
- **mkmatr/ mkmatc**: rather elementary functions which perform a transformation from a vector sized array to a matrix sized array for computational reasons only.
- **mklog**: provides facility to easily use the analysis and design tools in the zerodyn package

**procedure ldiff, ldiff(f,h,x)**

ldiff computes the directional derivative of a scalar valued function h(x) along a vector field f(x) and is often written as \( L_f h \). (N6),

\[
L_f h(x) = \frac{\partial h}{\partial x} f(x)
\]
procedure vdiff, vdiff(f,h,x)

vdiff is equal to ldiff with a slight difference that the second argument is not of the type 'scalar' but can be defined as a 'vector', i.e. a p-dimensional array. so the result of vdiff is an array. (N8).

\[
\begin{bmatrix}
\frac{\partial h_1}{\partial x} \\
\vdots \\
\frac{\partial h_p}{\partial x}
\end{bmatrix} = f(x)
\]

procedure liebrack, liebrack(a,b,x)

liebrack computes the liebracket of the two input vectors, (N10).

liebrack(a,b,x) = vdiff(a,b,x) - vdiff(b,a,x). This is the vector valued function \([a,b](x)\)

procedure extrank, extrank(A)

extrank computes the rank of the matrix A by means of the procedure extgausselim. extrank is not restricted to rational entries. However, if the entries of A fulfil the original demands, then the procedure fgausselim is used. This fraction free gaussian elimination has a better reliability. Extrank is based on the original MAPLE procedure 'rank', Copyright 1990 by the University of Waterloo;

See also '[linalg](rank)'.

procedure extgausselim, extgausselim(A,rm)

Gaussian Elimination: This algorithm is based on the standard MAPLE 'gausselim' procedure. Few ad-hoc changes are made to handle non-rational entries and to store which element of each column is used as a pivot. The gaussian elimination routine reduces the matrix A to triangular form. Based on the original version 'gausselim': Copyright 1990 by the University of Waterloo.

See also '[linalg](gausselim)'.

optional argument:

rm : elimination to stop at column rm
Appendix A

Description of the procedures

Output:

function value : upper triangular matrix

(optional:)
'extrank' : rank of A
'pivot' : vector which contains the pivot elements of A
'det' : determinant of A

**procedure extgaussjord**, extgaussjord(A, rmar)

Gaussian Elimination: This algorithm is based on the standard MAPLE 'gaussjord' procedure. Few ad-hoc changes are made to handle non-rational entries and to store which element of each column is used as a pivot. extgaussjord reduces the matrix A to diagonal (Gauss Jordan) form. Based on the original version 'gaussjord': Copyright 1990 by the University of Waterloo. See also extgaussjelg.

See also '[linalg](gaussjord)'.

**procedure extrowspace, extcolspace**, extrowspace(A, 'dim')

Computes the row (column) space of the matrix A and optionally assigns dim, the dimension of the row (column) space = rank(A). The row (column) space returned is a set of vectors. Few ad-hoc changes are made to handle non-rational entries. Based on the original version linalg[colspace]/rowspace, copyright 1990 by the University of Waterloo.

See also '[linalg](colspace/rowspace)'.

**procedure transform**, transform(f, g, h, x, xnull, 'phi')

transform computes a state space transformation to the normal form. Optional the inverse transformation and the jacobian matrix of the transformation are computed and tested. Also the new state variables zeta and eta are then available in the output. transform is used as a tool inside procedures from the zerodyn package (i.e. normform and statelin). The transformation is built up in the following way. The first $n-rotot$ elements of the mapping are defined by the output functions $ti$. The last $n-rotot$ transformations may be constructed in two manners. First, MAPLE tries to solve the partial differential equation $L_\Phi = 0$. If this succeeds, the transformation is completed and a message is returned. Else, if MAPLE fails to solve the differential equation (or cannot find enough solutions), the transformation is completed by adding unit vectors to the jacobian of the transformation. In this case no message is returned.

input: f, g, h, x, xnull (for further information see reldeg)
Appendix A

Description of the procedures

A.11

output:

phi : array which contains the state transformation to a normal form

(optional in successive order:)

phiinv : set which contains the inverse transformation
alpha : (array) jacobian matrix of the transformation phi(x)
zeta : (array) unassigned n*rot dimensional vector representing the first n*rot new state variables
eta : (array) unassigned n*rot dimensional vector representing the last n*rot new state variables

procedure psolve, psolve(gg,x)

psolve tries to solve the (set of) partial differential equation(s) \( L_g \Phi(x) = 0 \) for all \( x \) near \( x_{init} \). The algorithm is based on a constructive proof of the Frobenius theorem. This theorem proofs that a sufficient and necessary condition for the existence of \( n-m \) independent solutions is the involutivity or complete integrability of the distribution spanned by the column vectors of the input matrix \( gg \). The way this procedure must be called (i.e. by means of an array and vectors of variables and initial conditions) differs from the general way in which the MAPLE "solve procedures" are called (i.e. by means of sequences of equations and variables respectively). This difference is due to properties of the constructive proof that is used in the algorithm and also to the main purpose of this procedure in the zerodyn package where it is used to compute state transformations to the normal form. However, the general use of this procedure in order to solve a (set of) partial differential equation(s) is stressed.

input gg : nxm array whose entries are functions of \( x \) in such a way that the set of partial differential equation will be solved around the point \( x = x_{init} \)

\[
\frac{\partial \Phi}{\partial x} = \left[ \begin{array}{c} gg[1,1] \ldots gg[1,m] \\ \vdots \ldots \vdots \\ gg[n,1] \ldots gg[n,m] \end{array} \right] = 0
\]

x : unassigned n dimensional variable array

(optional:)

xinit : set which contains the initial conditions of the variables \( x \) in the form \( \{x[1]=... , , x[n]=...\} \) if this initial condition is not set then the symbolic values locx0.i will be used in the internal computations

output:

function result: - string with message: 'negative test result' or 'can't find solutions'

- array with solutions
Appendix A

Description of the procedures A.12

procedure extdsolve, extdsolve(gg,x)
procedure extdsolve(gg,x,flowvar)
procedure extdsolve(gg,x,flowvar,xinit)

If the MAPLE procedure -dsolve- failed in solving a set of n ordinary differential equations, it may be possible to solve, instead of all equations simultaneously, these n differential equations successively in order of increasing complexity while substituting the solutions of the previous equations. This is done in the procedure extdsolve. Note that also this procedure may easily fail to solve the set of ordinary differential equations. The way this procedure is written differs from the way the MAPLE procedure "dsolve" is written. This departure is due to the ad-hoc use of extdsolve in the procedure psolve.

input

  gg : nx1 array whose entries are functions of x in such a way that the n equations

\[
\frac{d x(\text{flowvar})}{d \text{flowvar}} = \begin{bmatrix} \text{gg[1](x)} \\ \vdots \\ \text{gg[n](x)} \end{bmatrix}
\]

will be solved.

  x : unassigned n dimensional variable array

(optional:)

  flowvar : symbolic name of the temporary flow variable. if this argument is not used then 'z' will be used.

  xinit : set which contains the initial conditions of the variables in the form \{x[1]=... , , x[n]=...\} if this argument is not set then the symbolic names locx0.i will be used.

output:

  function value : n dimensional array which contains the flow of array gg, a smooth function of flowvar and x with the property that the functions flow(xinit) = x(flowvar) solves the set of ordinary differential equations stated above.

procedure involutive, involutive(argseq,x)

involutive tests whether the sequence of input vectors are vector fields that span an involutive distribution. A distribution is called involutive if the Lie bracket of any pair of vector fields belonging to the distribution belongs to the distribution itself i.e. the distribution is closed under the Lie bracket operation.

input:

  argseq : set {} of n-dimensional vectors, this set may exist of names or vector-structures

  x : unassigned vector containing differential variables

output:

  function value : 'true' or 'false'
procedure mkmat, 
    mkmat(r)
    mkmat(c)

mkmat creates a \((r \times 1)\) or \((1 \times c)\) matrix structure of a \(r\) dimensional row or a \(c\) dimensional column array. The purpose of this matrix is to be able to apply (linear algebra) matrix computations on the original array.

input
\[
\begin{align*}
  r &: r \text{ dimensional array} \\
  c &: c \text{ dimensional array}
\end{align*}
\]

output:
function value : the corresponding matrix structure

procedure mklog, 
    mklog()

mklog creates a log -on the screen- of a full analysis of a system. mklog includes reldeg, normform, extnormform, statelin, outputfunc, inoutlin and transform. As much information as possible is returned.
Appendix B   Examples

In this appendix eight worked examples are presented. The first four are created with the aid of the procedure *mklog* and provide a good view of the possibilities and results of the analyses described in this thesis when they are performed by the procedures developed in this thesis. In fact these four examples show the information that is projected on the screen of the computer, using *mklog*. The last four examples provide specific information on some procedures. These examples explain the algorithms and give some insight in theory and analysis. For more information on these examples refer to Chapter 4.

Contents:

Example 1    SISO model of a *Single Link Manipulator with joint elasticity* B.2
Example 2    Model of a SISO system of a *Flexible Single Link Manipulator* B.7
Example 3    Model of a contrived nonlinear MIMO system B.12
Example 4    Model of a MIMO system of a *T-R Robot* B.15
Example 5    Explanation of the procedures *psolve* and *extsolve* B.19
Example 6    Explanation of the procedure *outputfunc* B.22
Example 7    Explanation of the procedure *inoutlin* B.25
Example 8    Explanation of the procedure *extnormform* B.27
EXAMPLE 1

Consider the model

\[
\begin{bmatrix}
  x_2 \\
  \frac{MGL}{Jm} \sin(x_1) - \frac{K}{Jm}(x_1 - x_3) \\
  x_4 \\
  \frac{K}{Jl}(x_1 - x_3)
\end{bmatrix} + \begin{bmatrix}
  0 \\
  0 \\
  0 \\
  1/Jl
\end{bmatrix} u
\]

\( y = x_1 \) (B1)

This model represents a rather simple mechanical system, a single link manipulator with joint elasticity. This elasticity is modelled by a torsional spring, for simplicity a linear spring with stiffness \( K \) is used. Further, \( M \) is the total mass of the link, \( L \) is the distance between the joint axis and the link centre of mass, \( G \) is the gravitational constant, and \( Jm \) and \( Jl \) are inertia of the motorshaft and the link respectively. The state variables are chosen as follows. \( x_1 \) is the link angle and \( x_3 \) is the motor shaft angle. \( x_2 \) and \( x_4 \) are the derivatives with respect to time of \( x_1 \) and \( x_3 \). The results of this example are created with the aid of the procedure \texttt{mklog}. To guide the reader, some additional information is provided.

```
> mklog();
output function(s): [ x[1] ]
```

The working point in the state space is chosen to be \((0,0,0,0)\). The dimension of the state space is 4 and the output function of the system is \( x_1 \). The analysis is started with the computation of the relative degree. Therefore, the procedure \texttt{reldeg} is called.

```
***** the relative degree, (reldeg)*****

* possible conditions on the initial states:
reldeg: no initial states are found for which the matrix \texttt{Adeg} turns out to be \texttt{singular}

* the total sum of the (vector) relative degree and this (vector) relative de
```
The nonsingularity matrix $A_{deg}$ and the matrix $b_{deg}$:

$$
\begin{bmatrix}
K
\end{bmatrix}
\begin{bmatrix}
\text{Im } J_1
\end{bmatrix}
$$

\[
\begin{bmatrix}
x(2) M G L \sin(x(1)) \text{Im } J_1 + J_1 M G L \cos(x(1)) \sin(x(1)) \\
+ J_1 M G L \cos(x(1)) x(1) K - J_1 M G L \cos(x(1)) x(3) K \\
+ J_1 K M G L \sin(x(1)) + J_1 K x(1) - J_1 K x(3) + K \text{Im } x(1) - K \text{Im } x(3)
\end{bmatrix}
\]

The relative degree is found to be 4, which is obviously a full order relative degree. Moreover, MAPLE did not find any initial states for which the matrix $A_{deg}$ will be singular. Therefore, this relative degree is valid in the whole state space.

The next step in the analysis is to compute the normal form of this system. The procedure `normform` is called.

```
***** the normal form, (normform)*****
* the normal form system equations: (normdyn)

[zeta[1]] = zeta[2],
[zeta[2]] = zeta[3],
[zeta[3]] = zeta[4],
+ K M G L sin(zeta[1]) - K u[1]/(Im J_1)

* the normal form system dynamics: (fn and gn)

2
- (- zeta[2] M G L sin(zeta[1]) J_1 + J_1 M G L cos(zeta[1]) zeta[3])
```

```
* the zero dynamic equations and the unique zeroing input: (zerodyn, uzero)

no_zerodyn
no_uzero
```

Because of the full order relative degree, it is obvious that the zero dynamics do not exist for this system. The easy structure of the system equations in normal form is shown.

In Chapter 3, it is stated that the zero dynamics can also be computed in another way, by means of the Zero Dynamics Algorithm. Call of the procedure `exitnormform`. Of course, because for this system a relative degree is defined, the results should be equal to the results obtained by `normform`. The steps in this result correspond to the steps in the algorithm, see chapter 3 and 4.
***** the zero dynamics from extnormform, (extnormform)*****

1, -th step, output zeroing submanifold: , \([x[1] = 0]\)
   1, -th step, constraints on the neighbourhood:
   no constraints on the chosen working point

2, -th step, output zeroing submanifold: , \([x[1] = 0, x[2] = 0]\)
   2, -th step, constraints on the neighbourhood:
   no constraints on the chosen working point

   3, -th step, constraints on the neighbourhood:
   no constraints on the chosen working point

4, -th step, output zeroing submanifold: ,
4, -th step, constraints on the neighbourhood:
   no constraints on the chosen working point

4, -th step, matrix \(L_{gh}\) has full rank \(m\)
* the zero dynamics:(zerodyn)
\[
* the zeroing input:(uzero)
\[ [ 0 ] \]

(In fact, this is a bit complicated way to say that no zero dynamics exist, but it is denoted in a rather general notation.)

The next step is the design of an exact linearizing state feedback, which linearizes the input-state equations. This feedback exists, because a full order relative degree is defined.
Call of the procedure statelin, the special structure of the linearized system is showed clearly.

***** exact linearization of the state input equations, (statelin)*****

* the exact linearizing feedback and linearized system dynamics: \((u, Az, B)\)
\[
\begin{align*}
&\begin{cases}
- (- \text{Im} J1 v[1] + x[2]) M G L \sin(x[1]) \text{Im} J1 \\
+ J1 M G L \cos(x[1]) \sin(x[1]) + J1 M G L \cos(x[1]) x[1] K \\
- J1 M G L \cos(x[1]) x[3] K + J1 K M G L \sin(x[1]) + J1 K x[1] \\
- J1 K x[3] + K \text{Im} x[1] - K \text{Im} x[3])/(K \text{Im})
\end{cases} \\
&\begin{cases}
\text{[zeta[2], zeta[3], zeta[4], 0]}
\end{cases}
\end{align*}
\]

[ [ 0 ] ]
[ [ 0 ] ]
[ [ 0 ] ]
[ [ 1 ] ]
Appendix B

Examples

B.5

* the explicit feedback: (alpha, beta)

\[
\begin{align*}
&\begin{bmatrix} 2 & 2 & 2 \\
&- (x[2] M G L \sin(x[1]) \Im J1 + J1 M G L \cos(x[1]) \sin(x[1]) \\
&\end{bmatrix} \\
&/(K \Im) \\
&\begin{bmatrix} \Im J1 \\
&------ \\
&K \\
&\end{bmatrix}
\end{align*}
\]

In case no output function is defined, or we want to compute an output function for which the system gets a full order relative degree, the procedure outputfunc is called. In the present case, it is obvious that \( x_1 \) is such a function, because for this output function already a full order relative degree is showed.

***** finding output functions which give the system a full order relative degree, (outputfunc)*****

reldeg: no initial states are found for which the matrix Adeg turns out to be \( \text{singular} \)

the output function(s) which fulfill the demands are:

\[
\begin{bmatrix} x[1] \end{bmatrix}
\]

The result of outputfunc is correct. The relative degree of this result is checked and (obviously) no initial states were found for the corresponding matrix Adeg turns out singular.

The next step in the analysis is to compute an exact linearizing state feedback, which linearizes the input-output behaviour of the system. Again, because of the full order relative degree and therefore the existence of an input-state linearizing feedback, the results should be exactly the same as obtained earlier with the procedure statelin. (A linear and controllable system has of course also a linear input-output behaviour.) Interesting, however, is that the results will be presented in the original coordinates \( x \) and not in the variables zeta, because the transformation to the normal form is not defined in this algorithm. Call of the procedure inoutlin.

***** exact linearization the input output mapping, (inoutlin)*****

* the exact linearizing feedback and linearized system dynamics: (u, fl, gl)

\[
\begin{align*}
&\begin{bmatrix} 2 \\
&- (\Im J1 v[1] + x[2] M G L \sin(x[1]) \Im J1 \\
+ J1 M G L \cos(x[1]) \sin(x[1]) + J1 M G L \cos(x[1]) x[1] K \\
- J1 M G L \cos(x[1]) x[3] K + J1 K M G L \sin(x[1]) + J1 K x[1] \\
- J1 K x[3] + K \Im x[1] - K \Im x[3]/(K \Im) \\
&\end{bmatrix} \\
\end{bmatrix}
\end{align*}
\]

***** exact linearization the input output mapping, (inoutlin)*****
Finally, it is worth mentioning the state transformation to the normal form, computed by the procedure transform:

***** transformation to the normal form, (transform)*****
* transformation, inverse transformation and jacobian

\[
\begin{pmatrix}
0 \\
0 \\
0 \\
\text{Im} \\
\text{K}
\end{pmatrix}
\]

* the explicit feedback: \((\alpha, \beta)\)

\[
\begin{pmatrix}
\end{pmatrix}/(K \text{Im})
\]

\[
\begin{pmatrix}
\text{Im} \\
\text{K}
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 \\
0 \\
0 \\
M G L \cos(x[1]) + K \\
- \text{Im} \\
x[2] M G L \sin(x[1]) + \text{Im} \\
x[2] M G L \sin(x[1]) + \text{K}
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 \\
1 \\
0 \\
0 \\
\text{K} \\
\text{Im} \\
\text{K} \\
\text{Im}
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 \\
0 \\
0 \\
\text{Im} \\
x[2] M G L \sin(x[1]) + \text{K} \\
\text{K} \\
\text{Im} \\
\text{Im}
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 \\
0 \\
0 \\
\text{K} \\
\text{Im} \\
\text{Im}
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 \\
0 \\
0 \\
\text{K} \\
\text{Im} \\
\text{Im}
\end{pmatrix}
\]
EXAMPLE 2

Consider a mechanical arm consisting of two rigid links, interconnected by a spring joint, moving on a horizontal plane, and operated by a single actuator at one of the two extremes. Perhaps, this can be considered as the most simplified model of a flexible one-link robot arm. This model is used by Isidori in the 'Carl Cranz Notes' ([9]) and incorporated in the notes of the 'Course on Advanced Control' (TUE-WFW).

\[ \dot{x} = f(x) + g(x)u \]  

(B2)

With

\[
\begin{bmatrix}
    x_3 \\
    x_4 \\
    \frac{bc(x_3+x_4)^2\sin x_2 + (b+c\cos x_2)Kx_2 + c^2x_3\sin x_2\cos x_2}{\Delta} \\
    -\frac{(b+c\cos x_2)(x_4+2x_3)c \sin x_2 + (a+2c\cos x_2)(c^2 \sin x_2 + Kx_2)}{\Delta}
\end{bmatrix}
\]

\[
g(x) = \begin{bmatrix} 0 \\ 0 \\ \frac{b}{\Delta} \\ \frac{-(b+c\cos x_2)}{\Delta}
\end{bmatrix}
\]

\[ \Delta = d - (c\cos x_2)^2 \]

And

\[
a = I_1 + m_1l_1^2 + I_2 + m_2(l_1^2 + l_2^2)
\]

\[
b = I_2 + m_2l_2^2
\]

\[
c = m_2l_2
\]

\[
d = b(a-b)
\]

\[ x_1 \] angular position of the first link

\[ x_2 \] relative angular position of the second link with respect to the first link

\[ u \] torque acting on the first link

Maple

Copyright (c) 1981-1990 by the University of Waterloo.
All rights reserved. Maple is a registered trademark of Waterloo Maple Software.
Type ? for help.

> mklog();# example2;

\[
\text{output function(s):} \quad [x[1]]
\]

\[
\text{initial states:} \quad \{x[1] = 0, x[2] = 0, x[3] = 0, x[4] = 0\} \]
***** the relative degree, (reldeg)*****

* possible conditions on the initial states:

reldeg: no initial states are found for which the matrix $A_{deg}$ turns out to be singular

* the total sum of the (vector) relative degree and this (vector) relative degree:

$$
2
$$

[ 2 ]

* the nonsingularity matrix $A_{deg}$ and the matrix $b_{deg}$:

$$
\begin{align*}
\left[ \begin{array}{c}
 b \\
 - \frac{2}{2} \\
 - b a + b + c \cos(x[2]) \\
\end{array} \right] \\
\left[ \begin{array}{c}
 - (b c \sin(x[2]) x[3] + 2 b c \sin(x[2]) x[3] x[4] + b c \sin(x[2]) x[4]) \\
 + x[2] K b + x[2] K c \cos(x[2]) + c \sin(x[2]) \cos(x[2]) \\
/ (- b a + b + c \cos(x[2])) \\
\end{array} \right]
\end{align*}
$$

***** the normal form, (normform)*****

MAPLE succeeded in solving the differential equations $\text{ldiff}(\phi, x)$

This statement means that MAPLE succeeded in solving the differential equations (3, 18)

* the normal form system equations (normdyn)

$$
\begin{align*}
[zeta[1]]_{\text{dot}} &= zeta[2], \\
[zeta[2]]_{\text{dot}} &= - (2 c zeta[2] \sin(eta[1]) \cos(eta[1]) b + c \sin(eta[1]) eta[2] b - 2 c \sin(eta[1]) eta[2] b zeta[2] \cos(eta[1]) \\
&+ eta[1] K c \cos(eta[1]) b + b w[1]) \\
&/ (b (- b a + b + c \cos(eta[1]) )); \\
[eta[1]]_{\text{dot}} &= - \frac{2}{2} \frac{- eta[2] b + zeta[2] c \cos(eta[1]) + zeta[2] b}{b} \\
[eta[2]]_{\text{dot}} &= - (2 c zeta[2] \sin(eta[1]) \cos(eta[1]) \\
\end{align*}
$$

* the normal form system dynamics (fn and gn)

$$
\begin{align*}
[zeta[2]], \\
- (c zeta[2] \sin(eta[1]) \cos(eta[1]) b + c \sin(eta[1]) eta[2] b \\
&- 2 c \sin(eta[1]) eta[2] b zeta[2] \cos(eta[1]) \\
&+ c \sin(eta[1]) zeta[2] \cos(eta[1]) + eta[1] K b)
\end{align*}
$$
+ \eta[1] K c \cos(\eta[1]) b / \quad \{b + (b a + b + c \cos(\eta[1])^{2})\),
- \quad \frac{2}{b}
- (- c \quad \zeta[2] \quad \sin(\eta[1]) \quad \cos(\eta[1]) + \sin(\eta[1]) \quad \eta[2] \quad \eta[2] \quad \zeta[2] \quad c
+ \eta[1] K b) / \quad \frac{2}{b}

\begin{bmatrix}
0 \\
0 \\
0 \\
- b \\
- b a + b + c \cos(\eta[1]) \\
\end{bmatrix}

\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
- \zeta[2] \\
\end{bmatrix}

* the zero dynamic equations and the unique zeroing input: (zerodyn, uzero)
\begin{align*}
\eta[1] K b \\
\end{align*}

Because the relative degree is strictly less than the dimension of the state space, the coordinates \( \eta \) appear in the normal form equations and the zero dynamics do exist. Because of the existence of the relative degree, the zero dynamics computed in extnormform should be equal:

***** the zero dynamics from extnormform, (extnormform)*****
1. -th step, output zeroing submanifold: \( \{x[1] = 0\} \)
1. -th step, constraints on the neighbourhood:
   no constraints on the chosen working point

2. -th step, output zeroing submanifold: \( \{x[1] = 0, x[3] = 0\} \)
2. -th step, constraints on the neighbourhood:
   no constraints on the chosen working point

2. -th step, matrix \( L_g \) has full rank \( m \)
   * the zero dynamics: (zerodyn)
\begin{align*}
\end{align*}
   * the zeroing input: (uzero)

2
\begin{align*}
\end{align*}

Apart from the notation in the coordinate \( x \) (See the state transformation at the end of this example), indeed the zero dynamics are equal.
Appendix B

Examples

B.10

***** exact linearization of the state input equations, (statelin)*****

statelin:

for the present choice of output function(s), this system does not have a full-
order (vector) relative degree. It is not possible to design an exact input-
state linearizing feedback. You may try to define a(n) (set of) dummy output fu-
nction(s) h(x) for which rtot=n by running -outputfunc- first or else to defin-
e an input-output linearizing feedback by running -inoutlin-

***** finding output functions which give the system a full order relative deg-
ree, (outputfunc)*****

for this siso system an output function will be searched for, which give the s-
ystem a maximal relative degree, however strictly less than the dimension of t\he state space

MAPLE failed in solving the flow of the , 2, -th column of ,

\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
- & b + c \cos(x[2]) & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{bmatrix}
\]

although this flow exists.

outputfunc:

MAPLE failed in finding solutions for a partial differential equation that sho\uld be solved in order to find the output function

\[ \lambda \]

Note that this is the case were the test on the results of extdsolve fail but it is not sure wether the actual results are correct or not. In this case the result is very complicated and one can not easily say. I assume that further analysis is not relevant in framework of this example and it is therefore omitted.

Though an exact input-state linearization does not exists, an input-output linearizing feedback can be computed with the procedure inoutlin

***** exact linearization the input output mapping, (inoutlin)*****

* the exact linearizing feedback and linearized system dynamics: (u, f1, g1)

\[
\begin{bmatrix}
2 & 2 \\
+ x[2] K c \cos(x[2]) + c x[3] \sin(x[2]) \cos(x[2]))/b \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 1 \\
0 & 1 \\
0 & 1 \\
b + c \cos(x[2]) \\
- b \\
\end{bmatrix}
\]
Appendix B

Examples

B.11

* the explicit feedback: (alpha, beta)

\[ \begin{align*}
\end{align*} \]

\[ \begin{bmatrix}
- b a + b + c \cos(x[2]) \\
- \frac{2}{b}
\end{bmatrix} \]

***** transformation to the normal form, (transform)*****

MAPLE succeeded in solving the differential equations ldiff(g, phi, x)

* transformation, inverse transformation and jacobian

\[ \begin{align*}
\{ x[1], x[3], x[2], \frac{1}{b} \}
\end{align*} \]

\[ \begin{align*}
x[1] &= \text{zeta}[1], x[3] = \text{zeta}[2], \\
x[4] &= \frac{1}{b} \text{eta}[2] b + \text{zeta}[2] c \cos(\text{eta}[1]) + \text{zeta}[2] b
\end{align*} \]

\[ \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
\frac{x[3] c \sin(x[2]) + b c \cos(x[2])}{b} & 0 & 0 & 1
\end{bmatrix} \]
EXAMPLE 3

Consider the contrived model ((1)(example 5.2.1)):
\[
\begin{bmatrix}
 x_2 + x_2^2 \\
 x_3 - x_5 + x_5^2 \\
 x_4 + x_5^2 \\
 x_3 \\
 x_5
\end{bmatrix}
+ \begin{bmatrix}
 0 \\
 0 \\
 \cos(x_1 - x_5) \\
 0 \\
 0
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2
\end{bmatrix}
\]

\[y = \begin{bmatrix} x_1 - x_5 \\
x_4 \end{bmatrix}\]  

***** the relative degree, (reldeg)*****

* possible conditions on the initial states:
reldeg: the initial state conditions for which the matrix Adeg turns out to be singular are:
\[\{x[1] = 1/2 \pi + x[5]\}\]

* the total sum of the (vector) relative degree and this (vector) relative degree:
5

* the nonsingularity matrix Adeg and the matrix bdeg:
\[
\begin{bmatrix}
\cos(x[1] - x[5]) & 1 \\
0 & 1
\end{bmatrix}
\begin{bmatrix} 2 \\
x[2]
\end{bmatrix}
\]

Note that this system has a full order vector relative degree and that this relative degree is only locally defined.

***** the normal form, (normform)*****

* the normal form system equations:(normdyn)
\[
\]

* the normal form system dynamics:(fn and gn)
\[
\begin{bmatrix} 2 \\
zeta[2], zeta[3], 0, zeta[5], zeta[2] \end{bmatrix}
\]
Examples

B.13

**the zero dynamic equations and the unique zeroing input:** (zerodyn, uzero)

\[
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
\cos(\zeta[1]) & 1 \\
0 & 0 \\
0 & 1
\end{bmatrix}
\]

* the zero dynamic equations and the unique zeroing input: (zerodyn, uzero)

\[
\text{no_zerodyn}
\]

\[
\text{no_uzero}
\]

***** the zero dynamics from extnormform, (extnormform)*****

1, -th step, output zeroing submanifold: , \([x[4] = 0, x[1] = x[5]]\)

1, -th step, constraints on the neighbourhood:

no constraints on the chosen working point

2, -th step, output zeroing submanifold: ,


2, -th step, constraints on the neighbourhood:

no constraints on the chosen working point

3, -th step, output zeroing submanifold: ,


3, -th step, constraints on the neighbourhood:

no constraints on the chosen working point

3, -th step, matrix \(L_\theta H\) has full rank \(m\)

* the zero dynamics: (zerodyn)


* the zeroing input: (uzero)

\[
\begin{bmatrix}
0 \\
0
\end{bmatrix}
\]

***** exact linearization of the state input equations, (statelin)*****

* the exact linearizing feedback and linearized system dynamics:

\[
2
\begin{bmatrix}
\cos(x[1] - x[5])
\end{bmatrix}
\begin{bmatrix}
\zeta[2] \\
\zeta[3] \\
0 \\
\zeta[5] \\
0
\end{bmatrix}
\]

* the explicit feedback: (alfa, beta)

\[
2
\begin{bmatrix}
x[2] \\
\cos(x[1] - x[5])
\end{bmatrix}
\begin{bmatrix}
\zeta[2] \\
0
\end{bmatrix}
\]
Appendix B

Examples

B.14

***** finding output functions which give the system a full order relative degree, (outputfunc)*****

reldeg: the initial state conditions for which the matrix Adeg turns out to be singular are:

\[
\{ x[1] = 1/2 \pi + x[5] \}
\]

the output function(s) which fulfil the demands are:

\[
\]

***** exact linearization the input output mapping, (inoutlin)*****

* the exact linearizing feedback and linearized system dynamics: (u, f1, g1)

\[
\begin{bmatrix}
\frac{2}{\cos(x[1] - x[5])} - x[2] \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{2}{\cos(x[1] - x[5])} - x[2] \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]

* the explicit feedback: (alfa, beta)

\[
\begin{bmatrix}
\frac{2}{\cos(x[1] - x[5])} - x[2] \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{2}{\cos(x[1] - x[5])} - x[2] \\
\end{bmatrix}
\]

***** transformation to the normal form, (transform)*****

* transformation, inverse transformation and jacobian

\[
\begin{bmatrix}
\end{bmatrix}
\]

\[
\begin{bmatrix}
x[5] = \text{zeta}[5], x[1] = \text{zeta}[1] + \text{zeta}[5], x[4] = \text{zeta}[4], x[2] = \text{zeta}[2], \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 0 0 0 -1 \\
0 1 0 0 0 \\
0 0 0 1 0 \\
0 0 0 0 1
\end{bmatrix}
\]
Consider the following nonlinear model of a RT Robot with two inputs (F and M) and also two outputs (r and ϕ).

\[
f(x) = \begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{bmatrix} = \begin{bmatrix}
x_1 - \frac{P_2 x_2^2}{P_1} \\
-2(P_2 x_2^2 - P_1 x_1^2 x_3)
\end{bmatrix}
\]

\[
g(x) = \begin{bmatrix}
1 \\
0 \\
0 \\
0
\end{bmatrix}
\]

(B6)

\[
[x_1, x_2, x_3, x_4] = [\dot{r}, \dot{\phi}, \phi]
\]

(B7)

**EXAMPLE 4**

In fact, this statement means that symbolic initial states are introduced. In practice this means that the results will hold for all \(x\), and therefore have a global value.

***** the relative degree, (reldeg)*****

* possible conditions on the initial states:

reldeg: no initial states are found for which the matrix \(\text{Adeg}\) turns out to be \(\text{singular}\)

* the total sum of the (vector) relative degree and this (vector) relative degree:

\[
\begin{bmatrix}
2 \\
2
\end{bmatrix}
\]

* the nonsingularity matrix \(\text{Adeg}\) and the matrix \(\text{bdeg}\):

\[
\begin{bmatrix}
1 \\
P_1 \\
1 \\
P_3 - 2P_2 x_2 + P_1 x_2^2
\end{bmatrix}
\]
\[
\begin{aligned}
\begin{bmatrix}
(P_1 x[2] - P_2) \\
-2 P_2 x[2] + P_1 x[2]
\end{bmatrix} \\
\begin{bmatrix}
\theta_1 \\
\theta_2 \\
\theta_3
\end{bmatrix}
\end{aligned}
\]

***** the normal form, (normform)*****

* the normal form system equations: (normdyn)

\[
\begin{aligned}
2 & \begin{bmatrix}
\dot{\zeta}_1 \\
\dot{\zeta}_2 \\
\dot{\zeta}_3 \\
\dot{\zeta}_4
\end{bmatrix} = \begin{bmatrix}
P_1 - 2 P_2 \zeta_4 + u[1] \\
\zeta_1 - 2 \zeta_2 \zeta_4 \\
\zeta_1 - 2 P_2 \zeta_1 + P_1 \zeta_1
\end{bmatrix}
\end{aligned}
\]

***** the zero dynamics from extnormform, (extnormform)*****

1, -th step, output zeroing submanifold: \( [x[2] = 0, x[4] = 0] \)

1, -th step, constraints on the neighbourhood:

no constraints on the chosen working point


2, -th step, constraints on the neighbourhood:

no constraints on the chosen working point

2, -th step, matrix \( L_gH \) has full rank \( m \)

* the zero dynamics: (zerodyn)

\[
\begin{aligned}
\{x[4] \dot{=} 0, x[1] \dot{=} 0, x[2] \dot{=} 0, x[3] \dot{=} 0\}
\end{aligned}
\]
Appendix B

Examples B.17

* the zeroing input: (uzero)
  [ 0, 0 ]

***** exact linearization of the state input equations, (statelin)*****
* the exact linearizing feedback and linearized system dynamics:

\[
\begin{array}{c}
\begin{pmatrix}
0 & 0 \\
1 & 0 \\
0 & 0 \\
0 & 1
\end{pmatrix}
\end{array}
\]

***** finding output functions which give the system a full order relative degree, (outputfunc)*****
reldeg: no initial states are found for which the matrix Adeg turns out to be \(A_{\text{deg}}\) singular

the output function(s) which fulfil the demands are:

[ x[4], x[2] ]

***** exact linearization the input output mapping, (inoutlin)*****
* the exact linearizing feedback and linearized system dynamics: (u, fl, gl)

\[
\begin{array}{c}
\begin{pmatrix}
1 & 0 \\
0 & 0 \\
0 & 1 \\
0 & 0
\end{pmatrix}
\end{array}
\]

* the explicit feedback: (alfa, beta)

\[
\begin{array}{c}
\begin{pmatrix}
-1 & 0 \\
-1 & 0
\end{pmatrix}
\end{array}
\]
\[
\begin{bmatrix}
    P1 & 0 \\
    0 & 2
\end{bmatrix}
\begin{bmatrix}
\end{bmatrix}
\]

***** transformation to the normal form, (transform)*****

* transformation, inverse transformation and jacobian

\[
\begin{bmatrix}
    x[2], x[1], x[4], x[3]
\end{bmatrix}
\]


\[
\begin{bmatrix}
    0 & 1 & 0 & 0 \\
    1 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1 \\
    0 & 0 & 1 & 0
\end{bmatrix}
\]
EXAMPLE 5

This example is to explain and illustrate the procedures psolve and extdsolve. For additional information refer to the theoretical description in Chapter 4, to the flowcharts in Appendix C, and if desirable to the listings in Appendix D.

Consider the partial differential equations (I)(example 1.4.3):

\[
\frac{\partial \lambda}{\partial x} = \begin{bmatrix}
2x_3 & -x_1 \\
-1 & -2x_2 \\
0 & x_3
\end{bmatrix}
= \begin{bmatrix}
0 \\
0
\end{bmatrix}
\]  

(B8)

So the \(n \times m\) input matrix \(g\) for the procedure psolve is:

\[
g = \begin{bmatrix}
2x_3 & -x_1 \\
-1 & -2x_2 \\
0 & x_3
\end{bmatrix}
\]  

(B9)

After testing the involutivity of this distribution, the matrix \(g\) is augmented to a full rank \(n \times n\) matrix \(G\) by adding an appropriate unit vector:

\[
G = \begin{bmatrix}
2x_3 & -x_1 & 1 \\
-1 & -2x_2 & 0 \\
0 & x_3 & 0
\end{bmatrix}
\]  

(B10)

In an iteration process the flows of the columns of \(G\) and a composition of the mapping \(F\) are computed. Computing the flow of each column means to solve a system of \(n=3\) ordinary differential equations in the (temporary) flow variable \(z_i\). The flow of each column is computed by the procedure extdsolve.

As an illustration of the strategy which is performed by extdsolve, in an intermezzo the computation of the flow of the first column of \(G\) is discussed.

**intermezzo: procedure extdsolve:**

In order to compute the flow of the first column of \(G\), the following set of equations should be solved

\[
\begin{align*}
(1) \quad \frac{dx_1(z_1)}{dz_1} &= 2x_3 & x_1(0) &= x_{10} \\
(2) \quad \frac{dx_2(z_1)}{dz_1} &= -1 & x_2(0) &= x_{20} \\
(3) \quad \frac{dx_3(z_1)}{dz_1} &= 0 & x_3(0) &= x_{30}
\end{align*}
\]  

(B11)

In extdsolve this will be done by subsequently solving of the separate equations (and their initial conditions) in order of increasing complexity. The results of the already solved equations are substituted in the equation that is to be solved. The order in which the equations in this set are solved is 3, 2, 1. A simple selection criterium is the "length" of the expression in MAPLE.
It is clear that the solutions of the equations (3) and (2) are independent. These equations are solved by the standard MAPLE procedure to solve ordinary differential equations: `dsolve`. The solutions are respectively

\[
\begin{align*}
  x_3(z_1) &= x_{30} \\
  x_3(z_1) &= -z_1 + x_{20}
\end{align*}
\]

(B12)

The substitution of these results in the first equation gives the ordinary differential equation

\[
\frac{dx_i(z_1)}{dz_1} = 2x_{30}
\]

(B13)

with as solution (via `dsolve`)

\[
x_i(z_1) = 2x_{30}z_1 + x_{10}
\]

(B14)

These solutions are generalized to the flow (B19) by substituting \((x_1, x_2, x_3)\) for \((x_{10}, x_{30}, x_{30})\).

**end intermezzo**

In the first step the flow of the third column is computed and the mapping \(F\) is initialized. The set of three equations that should be solved for the first flow is:

\[
\begin{align*}
  \frac{dx_i(z_1)}{dz_3} &= 1 \\
  \frac{dx_i(z_1)}{dz_3} &= 0 \\
  \frac{dx_i(z_1)}{dz_3} &= 0
\end{align*}
\]

(B15)

The flow of this first column is

\[
\Phi_{z_1} = \begin{bmatrix} z_3 + x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \Rightarrow \quad F := \begin{cases} x_1 = z_3 + x_{10} \\ x_2 = x_{20} \\ x_3 = x_{30} \end{cases}
\]

(B16)

The mapping \(F(z_3)\) is the solution of

\[
x(z_3) = \Phi_{z_1}(x_{x_0}) = \Phi_{z_2}(x_{20}, x_{30})
\]

In the next step, the second flow is computed in which \(F(z_1)\) is substituted after which the mapping \(F(z_2, z_3)\) is created:

\[
\Phi_{z_2} = \begin{bmatrix} \exp(-z_2)x_1 \\ \exp(-2z_2)x_2 \\ \exp(z_2)x_3 \end{bmatrix} \quad \Rightarrow \quad F := \begin{cases} x_1 = \exp(-z_2)(z_3 + x_{10}) \\ x_2 = \exp(-2z_2)x_{20} \\ x_3 = \exp(z_2)x_{30} \end{cases}
\]

(B18)

Finally, the first flow is computed and the mapping \(F(z_1, z_2, z_3)\) is completed:

\[
\Phi_{z_3} = \begin{bmatrix} 2z_1x_3 + x_1 \\ -z_1 + x_2 \\ x_3 \end{bmatrix} \quad \Rightarrow \quad F := \begin{cases} x_1 = 2z_1\exp(z_2)x_{30} + \exp(-z_2)(z_3 + x_{10}) \\ x_2 = -z_1 + \exp(-2z_2)x_{20} \\ x_3 = \exp(z_2)x_{30} \end{cases}
\]

(B19)
Clearly, $F$ is a function of $(z_1, \ldots, z_n)$ and $(x_{10}, \ldots, x_{n0})$. Now it is necessary to consider the workpoint $x_{null} = [0,0,1]$. After substitution of $\{x_{10}=0, x_{20}=0, x_{n0}=1\}$ the mapping $F$ is just a function of $(z_1, \ldots, z_n)$ and the inverse mapping $F_{inv}$ can be computed (by MAPLE):

$$F_{inv} = \begin{cases} z_1 = -x_2 \\ z_2 = \ln(x_1) \\ z_3 = (x_1^2 + 2x_1x_j)x_j \end{cases}$$  \hspace{1cm} (B20)

In [1] it is proved that the last $n-m$ components of $F_{inv}$ solve the partial differential equation (4,1), so the solution in this example is:

$$\lambda(x_1, x_2, x_j) = (x_1^2 + 2x_1x_j)x_j$$  \hspace{1cm} (B21)
EXAMPLE 6

This example is particularly meant for explaining and illustrating the procedure outputfunc. The results of the procedure outputfunc for the model used in example 3 will be labourously traced in this example. For additional information refer to Chapter 3 section IV, Chapter 4, the flowchart in Appendix C, and if desirable the listing in Appendix D.

Consider the model from example 3

PART I creation and test of dimension and involutivity of the distributions $G_0$ till $G_n$.

step 0:
The distribution $G_0 = \text{span}\{g_1, g_2\}$ has dimension $2 = m$ in a neighbourhood of $x^0=0$. It’s easy to show that this distribution is involutive because

$$[g_1, g_2](x) = 0 \quad (B22)$$

step 1:
In order to define the distribution $G_1$, the distribution $G_{total}$ is considered:

$G_{total} = \text{span}\{g_1, g_2, adg_1, adg_2\}$

$$
\begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & -\cos(x_1-x_2) & -1 \\
\cos(x_1-x_2) & 1 & -x_2\sin(x_1-x_2) & -(x_1-x_2) \\
0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0
\end{bmatrix} \quad (B23)
$$

From this matrix the space that is spanned by its column vectors is simplified to the simplest matrix possible by the MAPLE algorithm extcolspe to the distribution spanned by the column vectors of the matrix

$$G_1 = \begin{bmatrix}
0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \quad (B24)
$$

It is not difficult to prove that this distribution is involutive, moreover, also the distribution spanned by the columns of $G_{total}$ is involutive because

$$[g_1, adg_1](x) = [g_2, adg_2](x) = [g_2, adg_1](x) = [g_2, adg_2](x) = 0 \quad (B25)$$

$$[adg_1, adg_2](x) = \sin(x_1-x_2)g_1(x)$$
When \( x_{null} \) is substituted in \( G_i \) and again the simplest representation is derived we get, not very surprisingly

\[
G_{1.null} = \begin{bmatrix}
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix}
\]

This distribution has maximal dimension 4 at \( x = x^0 \), therefore it's dimension is constant near \( x^0 \). So the tests of this step are passed and the algorithm is continued.

**step 2:**

Similar computations led us via \( G_{total} \) to the distribution \( G_2 \) and \( G_{2.null} \).

\[
G_{total} := \text{span}\{g_1, g_2, \text{adj}_1g_1, \text{adj}_2g_2, \text{adj}_1^2g_1, \text{adj}_2^2g_2\}
\]

\[
G_{total} = \begin{bmatrix}
0 & 1 & 0 & 0 & \cos(x_1-x_2) (1+2x_2) & 2x_2 \\
0 & 0 & -\cos(x_1-x_2) & -1 & 2\sin(x_1-x_2) x_2 & 0 \\
\cos(x_1-x_2) & 1 & -x_2\sin(x_1-x_2) & -(x_1-x_2) & \cos(x_1-x_2) (x_4-x_2^2) + \sin(x_1-x_2) (x_4x_3-x_2x_5-x_3) & x_4 \\
0 & 0 & 0 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 2\cos(x_1-x_2) x_2 & 2x_2
\end{bmatrix}
\]

\[
G_2 = G_{2.null} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

It is clear that the distribution \( G_2 \) is involutive and has maximal dimension 5 at \( x = x^0 \).

Since by definition \( G_{i+1} \subset G_i \) for any \( i \geq 1 \), and \( G_2 \) has a dimension equal to the dimension \( n \) of the state space, it is obviously that \( G_2 = G_3 = G_4 \) and \( G_2, G_3 \) are (trivially) involutive. So the system satisfies the hypotheses. In this case the value of the integer \( kfix \) is 2. In maximal two steps the output functions \( \lambda \) will be constructed.
PART II construction of \(m=2\) output functions

**step 1:**
In order to construct the functions \(\lambda\) first the distribution \(G_1^1\) is considered. The dimension of this (co-)distribution is 1, so there exists \((m_1=1)\) a real-valued function \(\lambda_1(x)\) such that

\[
\text{span}\{d\lambda_1\} = G_1^1 = \ker\{(G_1)^T\} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}
\]

So a possible output function \(\lambda(x)\) turns out to be \(x_1 - x_2\). At the end of the first step the matrix \(\text{GTOTAL}\) is initialized:

\[
\text{GTOTAL} = \text{span}\{d\lambda_1\} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}
\]

**step 2:**
Now the matrix \(\text{GTOTAL}\) is augmented with the differentials of the derivatives of the functions that are found in the previous step. The differentials of the new output function(s) that will be found in the present step should be independent from these differentials.

\[
\text{GTOTAL} := \text{span}\{d\lambda_1, dL_i\lambda_1(x)\} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}
\]

At this stage we move one step backwards and consider the distribution \(G_2^1\). Now we search for \(m_2 = 1\) function \(\lambda_2(x)\) whose differential is spanned by

\[
\text{span}\{d\lambda_2(x)\} = \ker\{(\text{GTOTAL} \cup G_2^1)^T\} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}
\]

So the second output function which gives the system a full relative degree is found by: \(\lambda_2(x) = x_4\).

At this stage the number of functions found is equal to the number of wanted outputs and the algorithm stops.
EXAMPLE 7

This example is to explain and illustrate the procedureInOutin. For additional information refer to Chapter 3 section V, Chapter 4, the flowchart in Appendix C, and if desirable the listing in Appendix D.

Consider the system ([example 5.4.1]): (It is important to know, that for this system no relative degree exists.)

\[
\begin{align*}
\dot{x} &= \begin{bmatrix} x_2 x_1^2 \\ x_3 x_1 \\ 0 \\ x_5 + x_3^2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u_2 \\
y &= \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}
\end{align*}
\]

For this system, the Structure Algorithm proceeds as follows.

step 1:
Construct the matrix

\[
T_0(x) = L_h(x) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}
\]

\[\text{rank}(T_0) < m\] so the algorithm may be continued. In the first step the following decomposition can be computed

\[
V T_0 = \begin{bmatrix} P \\ K \end{bmatrix} T_0 = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} T_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = S
\]

yielding (using the names of the implementation):

\[
S = \begin{bmatrix} 1 & 0 \end{bmatrix}
\]

\[\text{gamma} = x_3\]

\[\text{gammaln} = x_3 - x_4\]

The results are stored in the column \(\Gamma\), which is initialized as

\[
\Gamma = \begin{bmatrix} \text{gamma} \end{bmatrix} = \begin{bmatrix} x_3 \end{bmatrix}
\]

The values of \(S\), \(\text{gamma}\), \(\text{gammaln}\) are available in the next step as \(S_{\text{prev}}\), \(\text{gammaprev}\), and \(\text{gammalnprev}\).

step 2:
Consider the matrix

\[
T = \begin{bmatrix} S_{\text{prev}} \\ L_{\text{gammalnprev}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}
\]
Appendix B

Examples

B.26

Again \( \text{rank}(T) < m \). The following decomposition can be computed

\[
V T = \begin{bmatrix} 1 & 0 \\ K_1 & K_2 \end{bmatrix} T = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} T = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \Sigma \end{bmatrix}
\]

yielding

\[
S = \begin{bmatrix} 1 & 0 \\ \end{bmatrix}
\]

\[
\text{gammaln} = \text{gammaprev} - L_j \text{gammalnprev} = x_1
\]

Note that \textit{gamma} does not exist (and also the column \( \Gamma \) cannot be augmented), because \( P \) does not exist. In terms of the algorithm, the present step degenerates because \( \text{rank}(T_3) = \text{rank}(T) \). Again, the values of \textit{gammaln} and \( S \) are in the next step available as \( S_{\text{prev}} \) and \( \text{gammalnprev} \).

\textbf{step 3:}

Consider the matrix

\[
T = \begin{bmatrix} \text{Sprev} \\ L_j \text{gammalnprev} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]

At this stage the algorithm is terminated because \( \text{rank}(T) = m \) and \( \text{gamma} = L_j \text{gammalnprev} = x_2 + x_1^2 \).

The column with the results \( \Gamma \) is augmented with this new function

\[
\Gamma = \begin{bmatrix} x_3 \\ \text{gamma} \end{bmatrix} = \begin{bmatrix} x_3 \\ x_2 + x_1^2 \end{bmatrix}
\]

Then, the feedback \( u = \alpha(x) + \beta(x) \) can be computed by solving the equations (3.76)

\[
\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} x_3 - x_1 \\ 2x_1^3 + 2x_1x_2 + x_1x_3 \end{bmatrix}
\]

for \( \alpha(x) \) and \( \beta(x) \), with obvious solutions

\[
\alpha = \begin{bmatrix} x_3 - x_1 \\ 2x_1^3 + 2x_1x_2 + x_1x_3 \end{bmatrix}
\]

\[
\beta = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}
\]
EXAMPLE 8

This example is particularly meant for explaining and illustrating the procedure extnormform. It is rather interesting that the text of this example is created by the trace command of MAPLE. This command provides possibilities to trace a procedure and its interim results. Only a few edits are made. For additional information refer to Chapter 3 section III, Chapter 4, the flowchart in Appendix C, and if desirable the listing in Appendix D.

Consider the system (II)(example 6.1.2): (It is important to know, that this system has no relative degree at the working point $x^0 = 0$.)

\[
\begin{align*}
  f(x) &= \begin{bmatrix} x_2 \\ x_4 \\ x_4x_2 \\ x_3 \\ x_2 \\ x_2 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \\
  g(x) &= \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \\
  h(x) &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
\end{align*}
\]  
(B47)

On this system, the Zero Dynamics Algorithm will have the following results:

\[
> \text{extnormform}(f,g,h,x,xnull,'zerodyn','uzero');
\]

\[
--> \text{enter extnormform, args }= f, g, h, x, \{x[4] = 0, x[5] = 0, x[2] = c2, x[1] = 0, x[3] = 0\}, \text{zerodyn, uzero}
\]

\textit{initialization of parameters}

\begin{align*}
  n &:= 5 \\
  m &:= 2 \\
  p &:= 2 \\
  \text{regular} &:= \text{true} \\
  \text{MANI} &:= \{\} \\
  \text{constraints} &:= \text{true}
\end{align*}

\textit{step 1:}

\begin{align*}
  dh &:= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \\
  \text{locs} &:= 2 \\
  \text{stot} &:= 2 \\
  H &:= \begin{bmatrix} x[1] \\ x[2] \end{bmatrix} \\
  HNILL &:= \begin{bmatrix} x[1] \\ x[2] \end{bmatrix} \\
  \text{PHITOT} &:= \begin{bmatrix} x[1] \\ x[2] \end{bmatrix} \\
  dHxnull &:= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}
\end{align*}

\[
\]
\text{MANI} := \{x[2] = 0, x[1] = 0\}

1, -th step, output zeroing submanifold: \{x[2] = 0, x[1] = 0\}

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\text{dilaug} := \[
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\text{cond} := \{
\}

1, -th step, constraints on the neighbourhood:

no constraints on the chosen working point

\text{L}_{GH} := \[
\begin{bmatrix}
1 & 0 \\
\end{bmatrix}
\]

\text{Lkern} := \[
\begin{bmatrix}
1 & 0 \\
x[3] & 0
\end{bmatrix}
\]

\text{Rkern} := \{[-x[3], 1]\}

\text{R} := \{[-x[3], 1]\}


\text{PHITOT} := \[
\begin{bmatrix}
x[1] \\
x[2]
\end{bmatrix}
\]


\text{Hprev} := \[
\begin{bmatrix}
x[1] \\
x[2]
\end{bmatrix}
\]

\text{dHprev} := \[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix}
\]

\text{step 2:}

\text{Hnew} := \[
\begin{bmatrix}
x[1] \\
x[2] \\
\end{bmatrix}
\]

\text{rowh} := 3

\text{dh} := \[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\end{bmatrix}
\]

\text{locs} := 1

\text{stot} := 3

\text{H} := \[
\begin{bmatrix}
x[1] \\
x[2] \\
\end{bmatrix}
\]
Appendix B

Examples

B.29

\begin{align*}
\text{dH} & := \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix} \\
\text{MANI} & := \{x[2] = 0, x[1] = 0, x[4] = 0\} \\
2, \text{-th step, output zeroing submanifold: } & \{x[2] = 0, x[1] = 0, x[4] = 0\}
\end{align*}

\begin{align*}
\text{dHnull} & := \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & -x[3] & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix} \\
\text{dHaug} & := \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\end{align*}

\begin{align*}
\text{cond} & := \{}
\end{align*}

2, \text{-th step, constraints on the neighbourhood:}

\begin{align*}
\text{no constraints on the chosen working point}
\end{align*}

\begin{align*}
\text{LgH} & := \begin{bmatrix}
1 & 0 & 0 \\
1 & 0 & -x[3] \\
x[3] & 0 & 1 \\
\end{bmatrix} \\
\text{Kern} & := \begin{bmatrix}
\left(\frac{x[3]}{1+x[3]}\right)^2 \\
-\frac{x[3] - x[5]}{x[3]} \left(\frac{x[3]}{1+x[3]}\right)^2, \frac{-x[3] - x[5]}{x[3]} \left(\frac{x[3]}{1+x[3]}\right)^2, 1\}
\end{bmatrix} \\
\text{PHI} & :=
\begin{align*}
\end{align*}

\begin{align*}
\text{RH} & := \begin{bmatrix}
\left(\frac{x[3]}{1+x[3]}\right)^2 \\
-\frac{x[3] - x[5]}{x[3]} \left(\frac{x[3]}{1+x[3]}\right)^2, \frac{-x[3] - x[5]}{x[3]} \left(\frac{x[3]}{1+x[3]}\right)^2, 1\}
\end{bmatrix} \\
\text{L} & := \begin{bmatrix}
\left(\frac{x[3]}{1+x[3]}\right)^2 \\
\left(\frac{x[3]}{1+x[3]}\right)^2, \frac{-x[3] - x[5]}{x[3]} \left(\frac{x[3]}{1+x[3]}\right)^2, 1\}
\end{bmatrix} \\
\text{R} & := \begin{bmatrix}
\left(\frac{x[3]}{1+x[3]}\right)^2 \\
\left(\frac{x[3]}{1+x[3]}\right)^2, \frac{-x[3] - x[5]}{x[3]} \left(\frac{x[3]}{1+x[3]}\right)^2, 1\}
\end{bmatrix}
\end{align*}
Appendix B

Examples

B.30

PHITOT :=

\[
\begin{bmatrix}
  x[1] \\
  x[2] \\
\end{bmatrix}
\]

\[
\]

PHIPrev :=

\[
\begin{bmatrix}
  x[1] \\
  x[2] \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

dHprev :=

\[
\begin{bmatrix}
  0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}
\]

step 3:

Hnew :=

\[
\begin{bmatrix}
  x[1] \\
  x[2] \\
\end{bmatrix}
\]

\[
\]

rowh := 4

dh :=

\[
\begin{bmatrix}
  1, 0, 0, 0, 0 \\
  0, 1, 0, 0, 0 \\
  0, - x[3], - x[2], 1, 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
  1 + x[3], \\
  1 + x[3]
\end{bmatrix}
\]

\[
\]
Appendix B

Examples

B.31

\[
\begin{align*}
\text{locs} & := 1 \\
\text{stot} & := 4 \\
H & := \\
& \{ x[1] \} \\
& \{ x[2] \} \\
dH & := \\
& \{ 1, 0, 0, 0, 0 \} \\
& \{ 0, 1, 0, 0, 0 \} \\
& \{ 0, -x[3], -x[2], 1, 0 \} \\
& / (1 + x[3]) \} / (1 + x[3]^2) \\
\end{align*}
\]
\[ \text{MANI} := \{x[2] = 0, x[1] = 0, x[4] = 0, x[5] = 0\} \]
3, \(\text{-th step, output zeroing submanifold:}\)
\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & -x[3] & 0 & 1 \\
\end{bmatrix}
\]
\[ \text{dHaug} := \]
\[
\begin{bmatrix}
0 & \text{-}\frac{1}{2} & \text{-}\frac{1}{2} \\
0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]
\[ \text{cond} := \{\} \]
3, \(\text{-th step, constraints on the neighbourhood:}\)
\[ \text{no constraints on the chosen working point} \]
\[
\begin{bmatrix}
1 & 0 \\
\end{bmatrix}
\]
\[ L_{gH} \]
\[
\begin{bmatrix}
0 & \text{-}\frac{1}{2} & \text{-}\frac{1}{2} \\
0 & 0 & 1 & 0 & 0 \\
\end{bmatrix}
\]
\[ 3, \text{-th step, matrix } L_{gH} \text{ has full rank} \]
\[ \text{termination of the Zeroa'yn Algorithm, computation of the zero dynamics and the corresponding zeroing inputs:} \]
\[ + \begin{array}{c}
\end{array} \]

\[ L_g \text{uzero} := \{ \text{uzero}[1], x[3] \text{uzero}[1] + x[2] \text{uzero}[2], \]


\[ \text{zerodyn} := \]

\[ \{ x[1] \text{dot} = 0, x[4] \text{dot} = 0, x[5] \text{dot} = 0, x[3] \text{dot} = -x[3], x[2] \text{dot} = 0 \} \]

\[ \text{ <-- exit extnormform } = \{ x[1] \text{dot} = 0, x[4] \text{dot} = 0, x[5] \text{dot} = 0, x[3] \text{dot} = -x[3], x[2] \text{dot} = 0 \} \]
Appendix C  Flowcharts

In this appendix flowcharts of some important procedures are presented. These flowcharts provide a remarkable insight in the structure of the procedures. Note that the flowcharts do not form complete specifications of the procedures by themselves, rather they should be used as an aid in reading and understanding of the integral listings in Appendix D. The following flowcharts are provided:

- transform  C.2
- normform  C.4
- extnormform  C.6
- statelin  C.9
- outputfunc  C.10
- inoutlin  C.14
- psolve  C.17
procedure transform (tools package)

\[ f, g, h, x, \text{null} \]

\[
\begin{align*}
\text{procedure reldeg: rdeg, Adeg, rtot} \\
\text{Adeg non-singular?} \\
\text{no} \quad \text{ERROR 'no solution available for transformation to normal form'} \\
\text{yes} \\
\text{computation of the mapping phiini: the first rtot state transformations phi[1..rtot] are equal to h_1, L.h., ... , L^{rtot} hry} \\
\phi := \phi_{ini} \\
\text{rtot = n?} \\
\text{yes} \\
\text{try to solve the partial differential equation L_{q#} s for the deficient transformations} \\
\text{no} \\
\text{procedure psolve: s} \\
\text{test on s, does s fulfill the conditions?} \\
\text{no} \\
\text{selection in phidiff of n-rtot independent solutions out of n-m computed solutions of s in such a way that the jacobian locifla is nonsingular} \\
\text{yes}
\end{align*}
\]
\( \text{jacobian localfa non-singular?} \)

- yes: \( \phi := \phi_{\text{diff}} \)
  - create \( \phi_{\text{un}} \) by adding unitvectors to \( \phi_{\text{ini}} \) (or in some cases to \( \phi_{\text{diff}} \)) in such a way that the jacobian localfa is non-singular
  - \( \text{jacobian localfa non-singular?} \)
    - yes: \( \phi := \phi_{\text{un}} \)
      - computation of the inverse transformation \( \phi_{\text{inv}} \)
        - \( \phi_{\text{inv}} \) O.K.? 
          - yes: \( \alpha := \text{localfa} \)
            - \( \phi, \phi_{\text{inv}}, \alpha \)
          - no: \( \text{ERROR: 'no solution available for transformation to normal form'} \)
    - no: \( \text{ERROR: 'MAPLE failed in generating the inverse transformation'} \)

- no: \( \text{ERROR: 'MAPLE failed in generating the inverse transformation'} \)
procedure normform (zerodyn package)

\[ f, g, h, x, x_{null} \]

\[ \text{procedure reldeg: rdeg, Adeg, bdeg, rtot} \]

\[ \text{procedure transform: phi, phiinv, alfa, zeta, eta} \]

computation of the normal form parameters \( \alpha_zeta \) and \( \beta_zeta \).

\[ zeta[rdeg[i]] \cdot \dot zeta[i] = \beta_zeta[i] + I(i=1 \text{ to } m)(\alpha_zeta \cdot u[i]) \]

I : computation of normdyn

computation of the left-hand sides of the system dynamic equations in normal form by substitution of the state transformation \( \phi \) in the state vector and differentiation.

computation of the right-hand sides by applying the state transformation \( \phi \) to the original right hand sides. creation of the normal form equations by combining both sides.

test on this normal form.

\[ \text{ERROR 'the system dynamics in normal form do not compare with previous results on the parameters \( \alpha_zeta \) and \( \beta_zeta \).'} \]

this test is a comparison between \( \alpha_zeta \) and \( \beta_zeta \) and the corresponding new system equations in normdyn. the use of this test is to check the results of MAPLE.

II : computation of \( f_a \) and \( g_n \)
Appendix C

Flowcharts

computation of the normal form system dynamics $fn$ and $gn$

differentiation

$dz / dt = fn(z) + gn(z)$$2$

$fn = \alpha f$$2$

$gn = \alpha g$$2$

$x = \phi_{inv}(z)$$2$

test on this normal form?

O.K.

not O.K.

III: computation of zerodyn and usero

$r_{tot} < n$?

no

yes

computation of the zero dynamics in zerodyn and the zeroing input in usero. these are derived by setting all $zeta[i]$ states zero.

comparisons between $azeta$ and $bzeta$, and the equations in normdyn with the corresponding new system equations in $fn$ and $gn$. the use of this test is to check the results of MAPLE.

these tests are comparisons between $azeta$ and $bzeta$, and the equations in normdyn with the corresponding new system equations in $fn$ and $gn$. the use of this test is to check the results of MAPLE.

zerodyn := 'nozerodyn'

usero := 'no_usero'
procedure extnormform (zerodyn package)

\( f, q, h, x, \text{xnull} \)

\( i := 1 \)

step \( i \) of the Zero Dynamics Algorithm

\( i = 1 \)

yes

selection of \( \text{locs} = \text{rank}(dh) \) functions of \( h(x) \) in \( H \)

no

\( \text{Enew} := \text{col}( \text{Eprev}, \text{PHprev}) \)

\( \text{locs} := \text{rank}(d\text{Enew}) - \text{stot} \)

\( \text{stot} := \text{stot} + \text{locs} \)

\( \text{Enew} := \text{col}( \text{Eprev}, \text{PHprev}) \)

\( \text{locs} := \text{rank}(d\text{Enew}) - \text{stot} \)

\( \text{stot} := \text{stot} + \text{locs} \)

selection of \( \text{stot} \) functions of \( \text{Enew} \) in \( H \) so that \( \text{rank}(d\text{E}) = \text{stot} \)

\( \text{BNULL} := \text{subs(MANI,E)} \)

computation / augmentation of the output zeroing submanifold \( \text{MANI} \):
solve the equations \( \text{BNULL} = 0 \)

termination:

\( H := \text{Eprev} \)

\( \text{stot} := \text{stot} - \text{locs} \)

no additional constraints found in the present step

yes

no
computation of $\tilde{u}$, possible constraints in the neighborhood of the working point $x_{null}$ in which $\text{MANI}$ holds. This are conditions for which the rank of $\Delta B_{null}$ is less than $\text{stot}$.

$\text{PBI} := R \ast L_{null}$

where $R = L_{null} = 0$

$\text{rank}(L \ast g(x_{null}) = \text{rank}(\text{subs}(\text{MANI}, L \ast g(x))) = m$

end of step $i$: $\text{PBI}_{\text{TOT}} := \text{col}(\text{PBI}_{\text{TOT}}, \text{PBI})$

$\text{PBI}_{\text{prev}} := \text{PBI}$

$\text{x}_{\text{prev}} := \text{x}$

$\Delta B_{\text{prev}} := \Delta B$

ERROR('not possible to find the zero dynamics and zeroing input')

\[ i < n \]

\[ i := i + 1 \]

the output zeroing input $u_{\text{zero}}$ can be solved from the equation: $\text{subs}(\text{MANI}, L_{null} + L_{null} \ast u_{\text{zero}}) = 0$

the zero dynamics are easily found by substitution of $u_{\text{zero}}$ and $\text{MANI}$ in the original system equations:

$\text{zerodyn} := \text{subs}(\text{MANI}, f(x) + g(x) \ast u_{\text{zero}})$
Flowcharts

usero, serodyn
procedure statelin (zerodyn package)

f, g, h, x, xsal, w

procedure reldeg: rdeg, Adeg, bdeg, rtot

rtot = n

not possible to design an exact linearizing feedback, try

construction of a linearizing feedback
u = alphax + betaxv

where v is the new reference input

procedure transform: phi, phiinv, alfa

step 1: linearizing feedback

betax(x) = Adeg^(-1)(x)
alphax(x) = -Adeg^(-1)(x) * bdeg(x)

step 2: state transformation

Az = alfa * (f + g * alphax) * phiinv
Bz = alfa * (g * betax) * phiinv

a, Az, Bz
procedure outputfunc (zerodyn package)

1 FIRST PART creation and test for involutivity and
dimension of the
distributions locG.0 to
locG.ktel

step 0:
creation and
test of the
distribution
locG.0

distribution
locG.0 = g
involutive

no

ERROR
'searching for
outputfunctions
fails'

yes

dimension
locGnull.0 = m

no

ERROR
'searching for
outputfunctions
fails'

yes

ktel := 1

step ktel:
creation,
simplification and
test of the
distribution
locG.ktel

dimension
of the
distribution
locGnull.ktel
constant

no

ERROR
'searching for
outputfunctions
fails'

yes

dimension
locGnull.ktel
= m ?

no

yes

no
Appendix C

Flowcharts

C.11

II SECOND PART now the integer \( k_{tel} \) is known for which all distributions \( \text{locG}_i \) for \( 0 \leq i < k_{tel} \) have a constant dimension and dimension \( \text{locG}_{k_{tel}-1} = n-1 \). Under these conditions there exists a solution for \( m \) dummy outputs that will fulfill a full relative degree. The construction of these outputs is performed in \( k_{tel} \) steps. The algorithm is described in Isidori. pp 250-256.

For a SISO system an output function will be searched which gives the system a maximal relative degree, however strictly less than the dimension of the state space \( n \).

The differentials of \( \text{locG}_1 \) functions \( \lambda \) are spanned by the kernel of distribution \( \text{locG}\text{null}.(k_{tel}-1) \). The functions \( \lambda \) are found by integration.
initialization of the matrix GTOTAL which column vectors are the differentials of the functions lambda and the differentials of the differentials along f of these functions lambda.

step i

computation of new derivatives and augmentation of GTOTAL with these vectors

computation of locm.i, the number of new function(s) lambda in the present step

locm.i = 0

yes

i := i + 1

no

creation of GKKRN, the concatenation of the distribution matrix locGnull.(ktel-i) and the matrix GTOTAL

the kernel of GKKRN contains the differentials of locm.i functions lambda
Appendix C

Flowcharts

1. dimension kernel GKh = locm.l
   yes
   computation of the function(s) lambda by integration

2. i := i+1
   no
   if locm.i ≠ m
   yes
   define output lambda

3. clear garbage
   TEST: procedure reldeg: Adeg, rdeg, rtot

4. rtot <> n or det(Adeg) ≠ 0
   yes
   output lambda
   ERROR 'finding output function failed'
   no
procedure inoutlin (zerodyn package)

\[ f, g, h, x, xnull, v \]

\[ \text{deltatot} := 0, \text{rprev} := 0 \]

compute matrix \( T0 \)

\[ T0 = \begin{bmatrix} L_{ym1} & \ldots & L_{ymn} \\ \vdots & \ddots & \vdots \\ L_{yn1} & \ldots & L_{ynm} \end{bmatrix} \]

\[ i := 1 \]

STEP \( i, (i=1..2n) \) of the Structure Algorithm

create the matrix \( T \), in the first step \((i=1), T0 \) will be used

\[ T = \begin{bmatrix} Sprev(x) \\ \vdots \\ L_{ymgamma(x)} \end{bmatrix} \]

\[ r := \text{rank}(T), \delta := r - \text{rprev} \]

is \( xnull \) a regular point of \( T \)

no -> ERROR

yes ->

termination of the S.A. set \( S, \gamma, \loggamma \) and \( \text{deltatot} \)

yes ->

\[ \text{rank}(T) = m \]

no ->

\[ \delta = 0 \]

yes ->

the present step degenerates: \( \gamma \) does not exist, compute \( \gamma \) and proceed with the next step with \( Sprev=S, \gamma = \gamma, \text{rprev} = r \)

no ->
compute matrices
set gamma, gamma_main, locgamma, deltatot

define the nonsingular $m \times m$ matrix $V$ of real
test the
numbers such that the
decomposition
holds, with $S(x)$ is a $r \times m$
matrix and $\text{rank}(S(x_{\text{null}})) = r$, where $1.dot$ stands
for an unit matrix of
order deltatot

solve the equations:
$S'_{\text{localfa}} = -L_{\text{gamma}}$
and
$S'_{\text{localbeta}} = \text{Unit}$
either by computation
of the inverse $S_{\text{inv}}$
or in a conventional
way if the inverse
cannot be computed.
create the feedback
\( u = \text{localfa} + g \text{lochbeta} \)
and the linearized system dynamics
\( f_n = f + g \text{localfa}, \)
\( g_n := g + \text{lochbeta} \)

\( \alpha := \text{localfa} \)
\( \beta := \text{lochbeta} \)

\( u, f_n, g_n, \alpha, \beta \)
procedure psolve (tools package)

\begin{align*}
& \text{gg, x, xinit} \\
& \text{distribution gg involutive?} \\
& \text{return 'there exist no \( n \times m \) linear independent solutions for an optimal transformation to the normal form'} \\
& \text{rank(gg) = 0} \\
& \text{return ('no nontrivial solutions')} \\
& \text{rank(gg) < m (dependent columns)} \\
& \text{extract rank(gg) columns of gg in the matrix g so that this matrix gets maximal rank} \\
& \text{create the \( n \times n \) matrix G which has full rank \( n \) by augmenting the inputmatrix g with unitvectors} \\
& \text{rank(G) = n?} \\
& \text{return 'maple failed in creating an augmented inputmatrix by adding unitvectors'} \\
& \text{create the flow of each column g.i of G} \\
& \text{solution of the flow by means of the procedure extdsolve and creation of the mapping \( F \) by backward substitution of the successive flows}
\end{align*}
substitution of $x_{null}$ in $F$ and computation of the inverse mapping $F_{inv}$

$F_{inv} \text{ O.K.?}$

yes

selection (out of $F_{inv}$ into output $z$) and test of $m-x$ independent solutions of the partial differential equation $L \phi(x) = 0$

$\text{selection succeeded?}$

yes

$z$

no

ERROR 'MAPLE failed in solving the inverse mapping'

RETURN 'MAPLE failed in solving the partial differential equations although a solution exist'
Appendix D  Listings of the procedures

In this appendix the listings of all procedures of the zerodyn and tools packages are presented integrally. Please reserve some minor changes.

Contents:

zerodyn package
  reldeg : D.2
  normform : D.3
  extnormform : D.7
  statelin : D.12
  inoutlin : D.17
  outputfunc : D.19
  outputfunc : D.25

tools package
  ldiff : D.31
  vdiff : D.32
  liebrack : D.33
  extrank : D.34
  extgausselim : D.35
  extgaussjord : D.36
  extcolspace/ : D.38
  extrowspace : D.40
  transform : D.41
  psolve : D.45
  extdsolve : D.50
  involutive : D.52
  mkmatr/ : D.53
  mkmatc : D.54
  mklog : D.54
# zerdyn package version 1.0

The zerdyn package contains several procedures for analysis and design of nonlinear (multivariable) control systems in the affine (state space) form:

\[
dx/dt = f(x) + g(x) u(t) \\
y(x) = h(x)
\]

This package provides facilities to symbolically compute and analyze the relative degree, the normal form, the zero dynamics and corresponding zeroing inputs of nonlinear SISO and MIMO systems. Also two static state feedbacks which (locally) linearize the nonlinear systems, and (dummy) output functions for which the system gets a maximal order relative degree can be computed.

The underlying theory is developed by Alberto Isidori. In fact this package is a symbolic implementation of the analytic theories described in his book "Nonlinear Control Systems", Springer-Verlag, 1989.

The available procedures are:

- **reldeg**: computation and analysis of the relative degree in a working point near of the state space.
- **normform**: computation of a local state transformation to a normal form, explicit rendering of the system dynamics in normal form, computation of the zero dynamics and the zeroing inputs.
- **extnormform**: computation of the zero dynamics for a broader class of systems, also systems which do not have a (vector) relative degree are incorporated.
- **statelin**: computation of a static state feedback and a state transformation to derive (locally) exact linearized input-state equations; the closed-loop system in new coordinates can be described as linear and is fully controllable.
- **inoutlin**: computation of a static state feedback to derive (locally) exact linearized input-output behaviour.
- **outputfunc**: computation of (dummy) output function(s) for which the system gets a maximal order relative degree.

For further information on the use and theoretical backgrounds of the functions refer to the book of Isidori and the thesis that accompanies this software (Harm van Essen, "Symbols speak louder than Numbers, Analysis and Design of Nonlinear Control Systems with a Symbolic Algebra Program", TUE-WW 92061, June 1992). Information on a particular function can also be found in the header of the procedure in the following listings.

This package makes extensive use of procedures collected in the accompanying "tools package". These procedures include differential operators, some (ad-hoc) extended standard MAPLE procedures, some additional computational functions to compute transformations, to solve (sets of) partial differential equations and to test for properties of distributions, and finally a procedure to easily use the procedures in both packages.

Instead of standard MAPLE procedures to perform gaussian elimination (gausselim and gaussjord) and rank computation (rank) in some functions special procedures are used. These procedures (i.e.extgausselim, extgaussjord, extrank, and extcolsapce) provide some ad-hoc extensions, for example they are not limited to rational entries only. For further information see the listings and headers of these functions in the tools package.


Appendix D

Listings of the procedures

D.3

# reldeg(f,g,h,x,xnull,'rdeg','rtot','Adeg','bdeg','cond')
#
# reldeg computes the (vector) relative degree of SISO systems and MIMO systems
data number inputs larger than or equal to the number of outputs.
# (Optional) some useful matrices and initial state conditions for which the
# (local) relative degree is not valid because the matrix Adeg is singular are
# returned.
#
# inputs:
# # f : state vector function f(x), a representation of the smooth
# vector field f, defined in MAPLE as an n dimensional array.
# g : input matrix function g(x), a representation of the smooth
# vector fields g_i with i=1..m, defined in MAPLE as an
# n x m dimensional array.
# h : output vector function h(x), a representation of the smooth
# vector field h, defined in MAPLE as an p dimensional array.
# x : state space variable vector, defined in MAPLE as an
# unassigned n dimensional array.
# xnull : working point in the state space, defined in MAPLE as a n
# dimensional set: \{ x[1]=..., ..., x[n]=... \}
# dx/dt = f(x) + g(x) u
# y = h(x)
#
# output parameters:
# (optional in successive order, the returned function value is equal to the
# (vector) relative degree, rdeg)
# # rdeg : array which contains the (vector) relative degree
# rtot : sum of the (vector) relative degree
# Adeg : array which contains non-singularity matrix ((5.1.2)) from Isi.)
# bdeg : array which contains matrix ((5.1.8) from Isi.)
# cond : condition on the initial state for which the matrix Adeg turns
# out to be singular and where the relative degree is no longer valid
#
# It may occur that the relative degree is not well defined at the chosen
# working point xnull. In that case the function value returns a sequence of
# such conditions, which may be incomplete.
#
# reldeg calls ldiff, extrank, extgaussjord, and several standard and [linalg]
# functions
# author: H.v.E., TUE-WFW, 1992
# reldeg := proc(f,g,h,x,xnull,rdeg,rtot,Adeg,bdeg,cond)
# local n,m,p,locrdeg,locrtot,lgf,lfh,setsvar,i,j,k,ii,jj,kk,varseq,solseq,
# notdefseq,condhlp,def,condseq,hulpvarllghaugllghpivotllghranklrtel:
# if nargs<5 or nargs>10 then ERROR('invalid number of arguments') fi:
# if not type(f,'vector') then ERROR('1st argument must be defined as a vector,'\n# see your model description')
# else n := linalg[vectdim](f) fi:
# if not type(g,'matrix') then ERROR('2nd argument must be defined as a matrix,'\n# see your model description')
# else m := linalg[coldim](g) fi:
# if not type(h,'vector') then ERROR('3rd argument must be defined as a vector,'\n# see your model description')
# else p := linalg[vectdim](h) fi:
# if not m>=p then ERROR('the number of inputs should be larger than or equal\nto the number of outputs') fi:
# if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('4th argument is\nthe state space variable vector and must be defined as an unassigned vector,'\nwrong type or number of elements, see your model description'):
# if not type(xnull,'set') or not nops(xnull)=n then ERROR('5th argument must be\ndefined as a set, wrong type or number of elements, see your model\ndescription') fi:
# locrdeg := array(1..p,[[]]):
# lfh := array(1..p,[[]]):
# lgf := array(1..p,l.m,[[]]):
# notdefseq:=NULL:
# def := 'true';
# condseq:=NULL;
# varseq:=NULL;
# for i to n do varseq[i]:=varseq[i]=x[i] od:
#
# relative degree
Appendix D

Listings of the procedures

D.4

locrtot := 0;
for i to p do
    for k to n while setvar=0 do
        locrdeg[i]:=k: # relative degree of the i-th output channel
        if k=1 then lfh[i]:=h[i]
            else lfh[i]:=ldiff(f,lfh[i],x) fi:
        for j to m do
            lgh[i,j]:=simplify(normal(ldiff(linalg[col](g,j),lfh[i],x))):
        od:
        for j to m do
            if lgh[i,j]<>O then setvar:=l;
            fi:
        od:
        if k=n and setvar=0 then ERROR('The relative degree cannot be defined
            for this system, the ',i,'-th output is not at all affected by the input but is
            only depending on the initial state') fi:
    locrtot := locrtot + locrdeg[i]:
od:
# tests
if m=p then

Appendix D

Listings of the procedures

D.5

if linalg[det](lgh)=0 then lprint('re1deg: the matrix Adeg turns out to be singular!!');lprint('');fi;
traperror(map(simplify,map(normal,subs(xnull,op(lgh)))));
if linalg[det](")=0 then print('re1deg: the matrix Adeg turns out to be singular in the chosen working point xnull!! relative degree is not valid at this point.');?>lprint('')fi:
else # m>=p
if not extrank(lgh)=p then print('re1deg: the matrix Adeg do not have maximal rank (equal to the number of outputs)>);iprint('')fi:
 otherwise
if nargs>5 then rdeg:=op(locrdeg) fi:
if nargs>6 then rtot:=locrtot fi:
if nargs>7 then Adeg:=op(lgh) fi:
if nargs>8 then bdeg:=array(l..p,[]!;
for i to p do bdeg[i]:=ldiff(f,bdeg[i],x)
od:
fi:
if nargs>9 then
if m>p then
# augmentation of lgh with unit rowvectors to an mxm matrix lghaug
# of rank m
extgaussjord(lgh,'lghrank','lghpivot');
if not lghrank=p then ERROR('extgaussjord: rank mismatch') fi:
lghaug := copy(lgh);rtel:=p;
for k to m do
hulpvar := array(sparse,l..m);
setvar := 'true';
for j to p do
if lghpivot[j]=k then setvar:='false'; break fi:
od:
if setvar='true' then
rtel:=rtel+1;
hulpvar[k]:=1;
lghaug := linalg[stack](lghaug,hulpvar) fi:
if rtel=m then break fi:
od:
if not rtel=m then ERROR('extgaussjord: pivot mismatch') fi:
if not extrank(lghaug)=m then
ERROR('extgaussjord: rank mismatch') fi:
else #(m=p)
lghaug:=copy(lgh)
fi:
else 
# conditions
solseq := [traperror(solve(linalg[det](lghaug),(varseq))]);
if solseq=lasterror then print('re1deg: MAPLE failed in computing the initial state conditions for which the matrix Adeg turns out to be singular')
eelif op(solseq)=NULL then print('re1deg: no initial states are found for which the matrix Adeg turns out to be singular')
else
k:=nops(solseq); for j to k do condhlp:=NULL;
for i to n do
if not(op(1,op(j,solseq)[i])-op(2,op(j,solseq)[i])=0) then condhlp:=condhlp,op(j,solseq)[i]; fi:
od:
if condhlp=NULL then condhlp:='all' fi:
solseq:=condseq,condhlp); od:
fi:
# output
if nargs>5 then rdeg:=op(locrdeg) fi:
if nargs>6 then rtot:=locrtot fi:
if nargs>7 then Adeg:=op(lgh) fi:
if nargs>8 then bdeg:=array(l..p,[]!;
for i to p do bdeg[i]:=ldiff(f,bdeg[i],x)
od:
fi:
if nargs>9 then
if m>p then
# augmentation of lgh with unit rowvectors to an mxm matrix lghaug
# of rank m
extgaussjord(lgh,'lghrank','lghpivot');
if not lghrank=p then ERROR('extgaussjord: rank mismatch') fi:
lghaug := copy(lgh);rtel:=p;
for k to m do
hulpvar := array(sparse,l..m);
setvar := 'true';
for j to p do
if lghpivot[j]=k then setvar:='false'; break fi:
od:
if setvar='true' then
rtel:=rtel+1;
hulpvar[k]:=1;
lghaug := linalg[stack](lghaug,hulpvar) fi:
if rtel=m then break fi:
od:
if not rtel=m then ERROR('extgaussjord: pivot mismatch') fi:
if not extrank(lghaug)=m then
ERROR('extgaussjord: rank mismatch') fi:
else #(m=p)
lghaug:=copy(lgh)
fi:
else 
# conditions
solseq := [traperror(solve(linalg[det](lghaug),(varseq))]);
if solseq=lasterror then print('re1deg: MAPLE failed in computing the initial state conditions for which the matrix Adeg turns out to be singular')
eelif op(solseq)=NULL then print('re1deg: no initial states are found for which the matrix Adeg turns out to be singular')
else
k:=nops(solseq); for j to k do condhlp:=NULL;
for i to n do
if not(op(1,op(j,solseq)[i])-op(2,op(j,solseq)[i])=0) then condhlp:=condhlp,op(j,solseq)[i]; fi:
od:
if condhlp=NULL then condhlp:='all' fi:
solseq:=condseq,condhlp); od:
cond:={condseq};
print('reldeg: the initial state conditions for which the matrix Adeg turns\
out to be singular are: ', op(cond));
fi; #(solseq)
fi; #(nargs>9)
op(locrdeg);
end: # end of the procedure reldeg
Appendix D

Listings of the procedures D.7

--- normform(f,g,h,x,xnull,'normdyn')
--- normform(f,g,h,x,xnull,'fn','gn')
--- normform(f,g,h,x,xnull,'normdyn','zerodyn','uzero')
--- normform(f,g,h,x,xnull,'fn','gn','zerodyn','uzero')

# normform computes a state transformation to a normal form. This normal form # can be computed in two different ways. A normal form only exists if the # system has a (vector) relative degree. Also the zero dynamics and the zeroing # input(s) can be computed. normform is suitable for SISO systems and # MIMO systems with a number of inputs larger than or equal to the number of # outputs.

# In the first way to compute the normal form equations, the computation of # 'normdyn', the state transformation is successively substituted in the # left-hand and the right-hand sides of the original equations, after which the # necessary differentiation is performed and both sides are combined. The # second way, the computation of 'fn' and 'gn', is computationally extremely less # demanding and makes use of the Jacobian of the transformation mapping to # circumvent the differentiation. So the second way is preferable. However, if # both ways are computed the MAPLE results will be automatically compared and # checked so the results will have a better reliability.

# input: f, g, h, x, xnull (for further information see reldeg)

# output:

normdyn : n dimensional set which contains the system equations in normal form in the following way:
[ zeta[1]dot = ... , ..., eta[n-rtot]dot = ... ]

fn, gn : (arrays) n dimensional vector and n x m matrix which describe the dynamics of the system in normal form. ( z stands for the new state vector: [ zeta[1] ]

zerodyn : n-rtot dimensional set which contains the explicit zero dynamics of the system in the following way:
[ eta[1]dot = ... , ..., eta[n-rtot]dot = ... ]

uzero : (array) the corresponding unique input that keeps the output zero

# normform calls reldeg, transform, and several standard and [linalg] functions

# author H.v.E., TUE-WFW, 1992

normform := proc(f,g,h,x,xnull,normdyn,fn,gn,zerodyn,uzero)
local i,k,Addeg,bdeg,rddeg,rdeg,rtot,phi,phiinv,alfa,nlmlndynloclplazetazeroinvl
\varlseq,var2seq,var3seq,t,j,gg,expseq,u,varsseq,azetazero, \bzetazero,testl, \zetazeta,etaazeta,bzeta,varsseq,seql,tlocnormdyn,locgu, \locfn,locgloczerodyn,loczerodyn,loczerodyn,loczerodyn,loczerodyn,

if nargs<6 or nargs>10 then ERROR('invalid number of arguments') fi;
if not type(f,'vector') then ERROR('1st argument must be defined as a vector,' see your model description') fi;
if not type(g,'matrix') then ERROR('2nd argument must be defined as a matrix,' see your model description') fi;
if not type(h,'vector') then ERROR('3rd argument must be defined as a vector,' see your model description') fi;
if not type(x,'vector') then ERROR('4th argument must be defined as a vector,' see your model description') fi;
if not type(xnull,'set') then ERROR('5th argument must be defined as a set, wrong type or number of elements, see your model description') fi;

# symbolic initialization of the input u := array(l..m,[]);
reldeg(f,g,h,x,xnull,'rdeg','rtot','Addeg','bdeg');
now the local parameters rdeg, rtot, Addeg, bdeg are defined
transform(:g,h,x,xnull,'phi','phiinv','alfa','zeta','eta'):
# now the local parameters phi, phiinv, alfa and the unassigned arrays zeta and
# eta are defined

# now all phi[l..n] state transformations to the normal form are known. The new
# coordinates are called zeta and eta. zeta/eta = phi(x), also the inverse
# transformation phiinv is known. To compute the dynamics of the system in
# normal form these new coordinates are substituted in the system. The new
# dynamics of the system can be partly written in the form of zeta and bzet
# zeta[rdeg[i]]dot=bzeta[i](zeta,eta) + Sum(j=1..m)(azeta[i,j](zeta,eta)*u[j])
# explicit computation of azeta and bzeta
azeta := map(simplify,map(normal,subs(phiinv,op(Adeg))));
bzeta := map(simplify,map(normal,subs(phiinv,op(bdeg))));

if nargs=6 or nargs=8 or nargs=10 then
# start of computation of normdyn, the first way to compute the normal form
# the substitution computed in transform is performed. This happens in two
# steps: first the left and than the right-hand side of the system equations,
# after which both sides are combined and simplified to the normal form.

# computation of the left-hand side of the new system dynamic equations by
# the substitution of phiinv in the vector x and differentiation to xdot.
# further a substitution of zeta[i]=locz.i(t) and eta[i]=loce.i(t) is
# performed for computational reasons. (substitution and differentiation)

seqs := NULL;
for i to rtot do
seqs := seqs,zeta[i]=locz.i(t);
od;
for i from (rtot+1) to n do
seqs := seqs,eta[i-rtot]=loce.(i-rtot)(t);
od;

lefthand := array(l..n,[]);
for i to n do lefthand[i] := x[i] od;
lefthand := subs(phiinv,op(lefthand));
lefthand := subs({seqs},op(lefthand));

# differentiation
for i to n do
lefthand[i] := diff(lefthand[i],t);
lefthand[i] := simplify(normal(lefthand[i]));
od;

# substitution of d locz.j(t)/dt=zeta[j]dot (var1seq) and back
# substitution of locz.j(t)=zeta[j] (var2seq).
# also an additional substitution sequence is created which will be
# used later on (var3seq)
var1seq := NULL;
var2seq := NULL;
var3seq := NULL;
for j to rtot do
var1seq := var1seq,diff(locz.j(t),t)='zeta['.j.']dot',
var2seq := var2seq,locz.j(t)=zeta[j];
var3seq := var3seq,'zeta['.j.']dot'';
od;
for j from (rtot+1) to n do
var1seq := var1seq,diff(loce.(j-rtot)(t),t)='eta['.(j-rtot).']dot',
var2seq := var2seq,loce.(j-rtot)(t)=eta[j-rtot];
var3seq := var3seq,'eta['.(j-rtot).']dot'';
od;

# left-hand side of the system equations in normal form
lefthand := eval(subs({var1seq},{var2seq},op(lefthand)));

# clear garbage
for i to rtot do locz.i := evaln(locz.i) od;
for i from (rtot+1) to n do locz.(i-rtot) := evaln(loce.(i-rtot)) od;
Appendix D

Listings of the procedures

D.9

# substitution of phiinv in the right-hand sides and creating new system
# dynamics equations (left and right hand sides combined)

exprseq := NULL;
leffhand := array(1..n,[]);
for i to n do
  for j to m do
    if j=1 then gg[i] := 0 fi:
    gg[i] := gg[i] + g[i,j]*u[j];
  od;
  leffhand[i] := simplify(normal((f[i] + gg[i])));
exprseq := exprseq, leffhand[i] = righthand[i];

righthand := array(l..n,[]);
for i to n do
  for j to m do
    if j=l then gg[i] := 0 fi:
    sstil := sstil + sti,jl*u[Jl; od;
  righthand[i] := simplify(normal((f[i] + gg[i])));
exprseq := exprseq, leffhand[i] = righthand[i];

ndynloc := traperror(solve(exprseq, {var3seq}));
normdyn := map(simplify, map(normal, ndynloc));
if ndynloc=lasterror then
  ERROR('MAPLE failed in solving the system');
fi:

locnormdyn := NULL;
for i to rtot do
  for j to n do
    if has(ndynloc[j], 'zeta['.i.'dot') then
      locnormdyn := locnormdyn, ndynloc[j]; break fi:
  od:
  for i from (rtot+l) to n do
    if has(ndynloc[j], 'eta['.(i-rtot).']dot') then
      locnormdyn := locnormdyn, ndynloc[j]; break fi:
  od:
locnormdyn := [op(locnormdyn)];
normdyn := copy(locnormdyn);

# simple test on this normal form, substitution of azeta and bzeta in the
# corresponding system equations

test := array(1..p,[]);
k := 0;
for i to p do
  if i=1 then k := rdeg[i] else k := k + rdeg[i] fi;
  test[i] := op(2, normdyn[k]) - test[i];
  if simplify(normal(op(2, normdyn[k]) - test[i])) < 0 then
    ERROR('mismatch, the system dynamics in normal form computed by means of substitution of the state transformation and differentiation (normdyn), do not agree with previous results on azeta and bzeta, MAPLE did not succeed in deriving the correct equations in normal form');
  fi:
  test[i] := simplify(normal((test[i] + azeta[i,j] * u[j]));
if nargs=7 or nargs=9 or nargs=10 then
  # second way to compute the explicit normal form dynamics is to compute the
  # system equations:
  # dx / dt = fn(x) + gn(x) u
  # (start computation of fn and gn)

  fnz := alfa * f(x) with x=phiinv(z);
  locfn := map(simplify, map(normal, linalg[multiply](alfa, f)));
  locfn := map(simplify, map(normal, subs(phiinv, op(locfn))));

  gnz := alfa * g(x) with x=phiinv(z);
  locgn := map(simplify, map(normal, linalg[multiply](alfa, g)));
  locbn := map(simplify, map(normal, subs(phiinv, op(locgn))));
  # simple test on this normal form, substitution of azeta and bzeta in the corresponding system equations
  test1 := array(1..p,[]);
  for i to p do
    if i=1 then k := rdeg[i] else k := k + rdeg[i] fi:
    test1[i] := op(2, normdyn[k]) - test1[i];
    if simplify(normal(op(2, normdyn[k]) - test1[i])) < 0 then
      ERROR('mismatch, the system dynamics in normal form computed by means of substitution of the state transformation and differentiation (normdyn), do not agree with previous results on azeta and bzeta, MAPLE did not succeed in deriving the correct equations in normal form');
    fi:
  fi:
test[i] := 0; testl[i] := locfn[k];
for j to m do
  test[i] := test[i] + azeta[i,j] * u[j];
od:
  test[i] := simplify(normal(test[i] + bzeta[i]));
for j to m do
  testl[i] := testl[i] + locgn[k,j] * u[j];
od:
  testl[i] := simplify(normal(testl[i]));
if not simplify(normal(testl[i] - test[i])) = 0 then
  ERROR('mismatch, the system dynamics in normal form computed by
  means of multiplication with the jacobian of the transformation
  mapping (fn, gn), do not agree with previous results on azeta and
  bzeta, MAPLE did not succeed in deriving the correct equations in
  normal form') fi;
  od: # (i)
# end of test

if nargs=10 then
  # test on this normal form by a comparision with the former derived
  # equations in normdyn
  test := array(1..3,[]);
  for i to n do
    test[i] := locfn[i];
    for j to m do
      test[i] := test[i] + locgn[i,j] * u[j];
    od:
    test[i] := simplify(normal(test[i]));
  if not simplify(normal(op(2,normdyn[i]) - test[i])) = 0 then
    ERROR('mismatch, the system dynamics in normal form, computed
    by means of multiplication with the jacobian of the state transforxation
    mapping do not agree with previous results on the normal form equations
    computed in normdyn. MAPLE did not succeed in deriving the correct equations
    in normal form') fi;
  od: #(i)
  fi: # end of the test

# output
if nargs=8 or nargs=9 then
  normdyn:=copy(locfn); fn:=copy(locgn)
fi:
if nargs=10 then
  fn:=copy(locfn); gn:=copy(locgn)
fi:
fi:
# end of computation of fn and gn

if rtot<n then
  # computation of the unique input uzero that is capable to keep the output
  # zero by substitution of {zeta[i]=0 and zeta[i]dot=0}(var4seq) in azeta and
  # bzeta
  varlseq := NiKL;
  for j to rtot do
    var4seq := var4seqrzeta[j]=0,'zeta[j]dot'=0
  od:
  azetazero := map(simplify,map(normal,subs({var4seq},op(azeta))));
  bzetazero := map(simplify,map(normal,subs({var4seq},op(bzeta))));
  if m=p then
    azetazeroinv := traperror(linalg[Inverse](azetazero));
    if azetazeroinv=lasterror then ERROR(azetazeroinv,'MAPLE failed in
    solving for the input u that keeps the output identically zero') end;
    locuzero := evalm(-1*(linalg[Multiply](azetazeroinv,bzetazero)));
    locuzero := map(simplify,map(normal,locuzero))
  else #m>p
    # the inverse of azetazero cannot be computed. the equation
    # bzetazero + azetazero*locuzero = 0 will be solved in a
    # conventional way. However, in general there are m unknown
    # inputs and p<m equations. It may be impossible to solve the
    # unique zeroing input uzero.
    print('normform: It may be impossible to solve the unique zeroing input
    uzero, because in general there are m unknown inputs and p<m equations.').
  end: #m>p
Appendix D

Listings of the procedures

D.11

exprseq := NULL;
varseq := NULL;
Au := 0;
for i to p do
for j to m do
Au := Au + azetazero[i, j] * locuzero[j] od;
exprseq := exprseq, bzetazero[i] + Au;
od;
for j to m do varseq := varseq, locuzero[j] od;
solseq := traperror(solve({exprseq}, {varseq}));
if solseq = lasterror then ERROR('MAPLE failed in solving for the input');
else assign(solseq); map(simplify, map(normal, locuzero));
fi:
fi; #(m=p)

if nargs = 9 then # computation of locnormdyn with fn and gn
locnormdyn := NULL;
for i to rtot do
locgu := 0;
for j to m do
locgu := locgu + g[i, j] * u[j]
od;
locnormdyn := locnormdyn, 'zeta[ i ]dot' = locfn[i] + locgu
od;
for i from rtot + 1 to n do
locgu := 0;
for j to m do
locgu := locgu + g[i, j] * u[j]
od;
locnormdyn := locnormdyn, 'eta[ i - rtot ]dot' = locfn[i - rtot] + locgu
od;
locnormdyn := [locnormdyn];
fi;

if nargs = 8 then # calculation of the explicit zero dynamics by substitution of {zeta[i]=0 and # zeta[i]dot=0} (var4seq) and substitution of the zeroing output uzero (var5seq)
var5seq := NULL;
for i to m do var5seq := var5seq, u[i] = locuzero[i] od;
loczerodyn := NULL;
for i from rtot + 1 to n do
loczerodyn := loczerodyn, subs ({var4seq}, {var5seq}, locnormdyn[i]);
od;
loczerodyn := [loczerodyn];
loczerodyn := map(simplify, map(normal, loczerodyn));
else # rtot=n
loczerodyn := evaln(no_zerodyn); locuzero := evaln(no_uzero);
fi;

if nargs = 8 then fn := copy(loczerodyn); gn := copy(locuzero) fi;
if nargs = 9 then gn := copy(loczerodyn); zerodyn := copy(locuzero) fi;
if nargs = 10 then zerodyn := copy(loczerodyn); uzero := copy(locuzero) fi;
fi; # end of computation of zerodyn and uzero

op(normdyn); end: # end of procedure normform
extnormform computes the zero dynamics and the corresponding zeroing input(s).

The algorithm used is the Zero Dynamics Algorithm of Isidori, in which a
locally maximal output zeroing submanifold is computed. This algorithm can
compute the zero dynamics even if the system does not have a (vector)
relative degree. An additional result of this algorithm is that a coordinates
transformation to an extended normal form can be computed. This Extended
Normal Form is not yet implemented and it is stressed that also the
implementation of the computation of the zero dynamics in this algorithm is
still in a developing stage.

extnormform is suitable for SISO systems and MIMO systems with an equal number
of inputs and outputs.

input: f, g, h, x, xnull (for further information see reldeg)
output:
zerodyn : set which contains the explicit zero dynamics of the system
in the following way:
\[
\begin{align*}
\dot{x}[1] &= \ldots, \\
\ldots & \\
\dot{x}[n] &= \ldots
\end{align*}
\]
uzero : array which contains the corresponding unique input that
keeps the state evolving on the maximal locally output
zeroing submanifold and keeps the output identically zero.

extnormform calls ldiff, vdiff, extrank, extgaussjord, and several standard
and [linalg] functions

author H.v.E., TUE-WFW, 1992

\begin{verbatim}
extnormform := proc(f,g,h,x,xnull,zerodyn,uzero)
local n,m,p,i,j,k,rr,PHIprev,HNULL,PHITOT, \ 
    varseq,eqnseq,regular,MAN1, \ 
    varseq := NULL;
for i to n do
    varseq := varseq, x[i]
do;

if not nargs=7 then ERROR('invalid number of arguments') fi:
if not type(f,'vector') then ERROR('1st argument must be defined as a vector,' see your model description')
else n := linalg[vectdim](f) fi:
if not type(g,'matrix') then ERROR('2nd argument must be defined as a matrix,' see your model description')
else m := linalg[coldim](g) fi:
if not type(h,'vector') then ERROR('3rd argument must be defined as a vector,' see your model description')
else p := linalg[vectdim](h) fi:
if not m=p then ERROR('the number of inputs and outputs should be equal') fi:
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('4th argument is' the state space variable vector and must be defined as an unassigned vector,
wrong type or number of elements, see your model description') fi:
if not type(xnull,'set') or not nops(xnull)=n then ERROR('5th argument must be'
defined as a set, wrong type or number of elements, see your model \ 
description') fi:

if constraints:
if constraints = true then
    constraints := false;
else
    constraints := true;
fi:

\end{verbatim}
for i to n do
# step i of the Zero Dynamics Algorithm, maximal n steps.

# computation of the vector \( H \), its jacobian matrix \( \partial H \) and \( H\text{null} \), this is the
# vector \( H \) in which the output zeroing submanifold MANI is substituted.
# in connection with the initialization of \( H \) and PHI, a distinction is made
# between the first and the following steps of the procedure.
if i=1 then
    \( \partial h := \text{linalg[\partialian]}(h,x); \)
    locs := extrank(\( \partial h \));
    stot := locs;
    if locs = m then
        \( H := \text{copy}(h); \)
        \( H := \text{mkmatc}(\text{op}(H)); \)
        \( \partial H := \text{copy}(\partial h); \)
    else
        # the differentials of the entries of \( h \) are dependent.
        # selection of locs output functions in \( H \) whose differentials 
        # are independent
        # a consequence is that the chosen working point cannot be a
        # regular point of the algorithm
        regular := 'false';
        \( H := \text{array}(1..\text{locs},1..1,[]); \)
        for k to locs do
            if k=1 then
                # select the first output function
                for j to m do
                    \( \text{mkmatr}(\text{linalg[\row]}(\partial h,j)); \)
                    if extrank(")=1 then
                        \( \partial h := "; \)
                        \( H[k,1] := h[j]; \)
                        \( rr:=j; kk:=1; \)
                        break
                    fi:
                od:
            else
                for j from rr to m do
                    if
                        extrank(linalg[stack](\( \partial h,\text{linalg[\row]}(\partial h,j)\)));>kk then
                            \( \partial h := \text{linalg[stack]}(\( \partial h,\text{linalg[\row]}(\partial h,j)\)); \)
                            \( H[k,1] := h[j]; \)
                            \( rr:=j; kk:=kk+1; \)
                            break
                    fi:
                od:
            fi:
        od:
        if not kk=locs then ERROR('extnormform: rank mismatch') fi:
        \( \text{Hnull} := \text{copy}(H); \)
        \( \text{PHITOT} := \text{mkmatc}(\text{op}(h)); \)
        \( \partial h\text{null} := \text{map}(\text{simplify},\text{map}(\text{normal},\text{subs}(\text{xnull},\text{op}(\partial h)))); \)
        if not extrank(\( \partial h\text{null} \))=locs then
            ERROR('extnormform: no constant rank') fi:
    else
        \( Hnew := \text{linalg[stack]}(\text{Hprev},\text{PHIPrev}); \)
        \( \text{rowh} := \text{linalg[\rowdim]}(\text{Hnew}); \)
        \( \partial h := \text{linalg[\partialian]}(\text{linalg[\col]}(\text{Hnew},1),x); \)
        locs := extrank(\( \partial h \)) = stot;
        stot := stot + locs;
        if rowh=stot then \( H := \text{copy}(Hnew); \)
        \( \partial H := \text{copy}(\partial h) \) else
            # the differentials of the entries of \( H \) are dependent and this 
            # is due to dependencies in the differentials of \( \text{PHIPrev} \). now
            # perform a selection of stot functions in \( H \) whose 
            # differentials are independent
            # a consequence is that the chosen working point cannot be a
            # regular point of the algorithm
            regular := 'false';
        fi:
        \( H := \text{array}(1..\text{stot},1..1,[]); \)
        \( \partial h := \text{copy}(\text{Hprev}); \)
        for k to (\text{stot} - \text{locs}) do
            \( H[k,1] := Hnew[k,1]; \)
        od:
else
    \( Hnew := \text{linalg[stack]}(\text{Hprev},\text{PHIPrev}); \)
    \( \text{rowh} := \text{linalg[\rowdim]}(\text{Hnew}); \)
    \( \partial h := \text{linalg[\partialian]}(\text{linalg[\col]}(\text{Hnew},1),x); \)
    locs := extrank(\( \partial h \)) = stot;
    stot := stot + locs;
    if rowh=stot then \( H := \text{copy}(Hnew); \)
    \( \partial H := \text{copy}(\partial h) \) else
        # the differentials of the entries of \( H \) are dependent and this 
        # is due to dependencies in the differentials of \( \text{PHIPrev} \). now
        # perform a selection of stot functions in \( H \) whose 
        # differentials are independent
        # a consequence is that the chosen working point cannot be a
        # regular point of the algorithm
        regular := 'false';
    fi:
    \( H := \text{array}(1..\text{stot},1..1,[]); \)
    \( \partial h := \text{copy}(\text{Hprev}); \)
    for k to (\text{stot} - \text{locs}) do
        \( H[k,1] := Hnew[k,1]; \)
    od:
j:=stot-locs+l;
for k from j to rowh do
   if extrank(linalg[stack](dH,linalg[row](dh,k))) = j then
      dh := linalg[stack](dH,linalg[row](dh,k));
      H[3,j] := Hnew[k,3]; j := j+l; if j=stot then break fi;
   fi;
od;
if not extrank(dH)=stot then
   ERROR('extnormform: rank mismatch') fi:
fi:

HNull := map(simplify,map(normal,subs(op(MANI),op(H))));
dHnull := map(simplify,map(normal,subs(xnull,op(dH))));
if not extrank(dHnull)=stot then
   ERROR('extnormform: no constant rank') fi:
fi: #(i=l)

# computation in the first step and augmentation of the locally maximal output
# zeroing submanifold MANI in the successive steps: MANI = {x in U1 : H(x)=0}
# the Zero Dynamics Algorithm terminates if in the present step the submanifold
# eqnseq := NULL;
for k from stot-locs+l to stot do
   eqnseq:=eqnseq,Hnull[k,1]
od:
M:={traperror(soive({eqnseq},{Varseq}))};
if M=lasterror then ERROR(M,'MAPLE failed in computing a smooth submanifold')
else
   k:=nops(M); MANIhlp:=NULL;
   for ii to k do
      for jj to n do
         if not (op(1,op(ii,M)[jj])=op(2,op(ii,M)[jj])) then
            MANIhlp := MANIhlp,op(op(ii,M))
         fi:
      od:
   if MANIhlp=NULL then lprint(i,'-th step, no additional constraints\founded, termination of the Zero Dynamics Algorithm');
   else MANI:=[op(MANI),MANIhlp];
   fi:
print(i,'-th step, output zeroing submanifold: ',MANI);
if constraints then
   # computation of a neighbourhood U of xnull for which the present step holds:
   # investigation of possible conditions in the state space for which the rank of
   # dH is strictly less than stot.
   extgaussjord(linalg[transpose](dH),'dHrank','dHpivot');
   if not dHrank=stot then ERROR('extgaussjord: rank mismatch') fi:
else
   if stot<n then
      # augmentation of dH with unit rowvectors to an nxn matrix dHaug
      # of rank n
      dHaug := copy(dH);rtel:=stot;
      for k to n do
         hulpvar := array(sparse,l..n);
         setvar := 'true';
         for j to stot do
            if dHpivot[j]=k then setvar:='false'; break fi;
         od:
         if setvar='true' then rtel:=rtel+1;hulpvar[k]:=1;
         dHaug := linalg[stack](dHaug,hulpvar) fi:
      if rtel=n then break fi;
   od:
   if not rtel=n then ERROR('extgaussjord: pivot mismatch') fi:
   if not extrank(dHaug)=n then
      ERROR('extgaussjord: rank mismatch') fi:
fi: #(stot<n)

# substitution of MANI in order to avoid trivial conditions
   dHaug := map(simplify,map(normal,subs(op(MANI),op(dHaug))));
# searching for constraints
cond := traperror(solve(linalg[det](dHaug), {varseq}));
if cond=lasterror then ERROR(cond,’MAPLE failed in computing conditions in the state space for which the rank of’,op(dH),’is less than’,locs)
elif op(cond)=NULL then U:=NULL
else
k:=nops(cond);
for j to k do condhlp:=NULL;
for jj to n do
if not (op(1,op(j,cond)[jj])=op(2,op(j,cond)[jj])) then
dcondhlp:=condhlp,op(j,cond)[jj]
of;
if condhlp=NULL then condhlp := ‘all’ fi;
U := U,{condhlp};
of;
fi:
print(i,’-th step, constraints on the neighbourhood:’,U);
if U=NULL then print(’no constraints on the chosen working point’) fi;
lprint(‘ ');lprint(’1
fi: #(constraints)

# computation of the vector PHI (via computation of L_gH and R)
if i=1 then
L_gH := array(1..stot,1..m,[]);
else
L_gH := linalg[extend](L_gH,locs,0);
fi:
for k from stot-locs+1 to stot do
for j to m do
L_gH[k,j] := ldiff(linalg[col](g,j),H[k,l],x);
od;
of;

# if the matrix L_gH has rank equal to m at the working point xnull
# for all x in U and MANI, then the Zero Dynamics Algorithm is terminated
# because no matrix R can be found no more.
if extrank(map(simplify,map(normal,subs(op(MANI),op(L~gH)))))=m and extrank(map(simplify,map(normal,subs(xnull,op(L~gH)))))=m then
print(i,’-th step, matrix L_gH has full rank m’); break fi:

if i=1 then Lkern:=copy(L_gH)
else Lkern := linalg[extend](L_gH,0,locs,0);
for k to locs do
for j to stot-locs do
Lkern[j,k+m]:=R[k,j]
of;
of;
fi:

Lkern := map(simplify,map(normal,subs(op(MANI),op(Lkern))));
Rkern := traperror(linalg[kernel](linalg[transpose](Lkern),’nulldim’));
if Rkern=lasterror then ERROR(’MAPLE failed in solving the kernel of the\ matrix’,op(L_gH),’although a solution exists’) fi;
R := op(1,Rkern);
for j from 2 to nulldim do
R := linalg[stack](R,op(j,Rkern))
of;
if nulldim=1 then R:=mkmatr(op(R)) fi:
if stot=1 then PHI:=array(1..1,[]); PHI[1]:=map(simplify,map(normal,\ R[1,1]*ldiff(f,H[1,1],x)))
else
PHI := map(simplify,map(normal,
linalg[multiply](R, vdiff(f,linalg[col](H,1),x))));
fi:
PHI := mkmatr(op(PHI));
PHITOT := linalg[stack](PHITOT,PHI);
PHIprev:=copy(PHI);
Hprev:=copy(H);
dHprev:=copy(dH);

# end of step i
od:
# after this zero dynamics algorithm we have constructed a locally maximal
# output zeroing submanifold MANI. with this submanifold it is possible to
# compute the zero dynamics and the unique input u that keeps the state evolving
# on this submanifold.
# uzero can be solved from the following equations: L_fH + L_gH*uzero = 0
uzero:=array(1..m,[]); varseq:=NULL;
for i to m do varseq:=varseq,uzero[i]; od;
L_fH := vdiff(f,linalg[col](H,l),x);
L_gHuzero := linalg[multiply](L_gH,uzero);
eqnsq:=NULL;
for i to m do eqnseq:=eqnseq,simplify(normal(subs(op(MANI),(L_fH[i]+L_gHuzero[i])))) od;
locuzero := {} (traperror(solve({eqnseq},{varseq}));
if locuzero=lasterror then ERROR('MAPLE failed in solving the unique output\
zeroing input uzero') fi:
def := 'true';
for k to m do if (op(1,op(k,op(locuzero))) = op(2,op(k,op(locuzero)))) then def:='false' fi:
locuzero:=evaln(no-uzero);zerodyn:=evaln(no-zerodyn); RETURN(); else assign(locuzero) fi;
uzero := map(simplify,map(normal,uzero));

# application of uzero and substitution of the output zeroing submanifold MANI
# in the system equations f(x) + g(x)*uzero(x) lead to the zero dynamics of the
# system.
zerohlp:=map(simplify,map(normal,\
subs(op(MANI),linalg[add](f,linalg[multiply](g,uzero)))));
for i to n do
if not zerohlp[i]=0 then loczerodyn:=loczerodyn,
'x'['i,'dot'=zerohlp[i] fi:
loczerodyn:=loczerodyn,'x'['i,'dot'=zerohlp[i] od;
zerodyn := {loczerodyn};

# with the functions PHI that are stored in the vector PHITOT it is possible
# to compute a coordinates transformation to the so called extended normal form.
# This is only possible if the chosen working point xnull is a regular point
# of the above zero dynamics algorithm. in that case the differentials of the
# entries of PHITOT are linearly independent at xnull. xnull is a regular point
# if the invertibility hypotheses are assumed. these mean that in the first
# step
# \[ H(x) = h(x) \]
# holds and in the following steps
# \[ H(x) = [ Hprev ] \]
# \[ PHIprev \]
# holds.
# in this implementation this hypotheses is easily tested with the boolean
# variable regular.
statelin computes the exact linearizing static state feedback $u$ and the corresponding state transformation which linearize the input-state equations of a nonlinear SISO or square MIMO system under the condition of a full order relative degree: there must exist a(n) (set of) output function(s) for which the sum of the (vector) relative degree is equal to the dimension of the state space of the system, i.e. $rtot=n$.

The state transformation that is used is exactly equal to the transformation to the normal form and is (internally) computed by the procedure transform. The new state coordinates are called $zeta$. The closed loop system in the new coordinates is linear and controllable.

input $f$, $g$, $h$, $x$, $xnull$ (for further information see reldeg)

output (arrays)

$u$ : the linearizing feedback $u(x) = alfa(x) + beta(x) \cdot v$
$Az$ : the linear system matrix multiplied with the new state vector ($zeta$) of the closed-loop system
$Bz$ : input matrix of the linearized closed-loop system
$dzeta/dt = Az(zeta) + Bz \cdot v$

(also):

$alfa$ : the linearizing feedback $alfa(x)$ and $beta(x)$

statelin calls reldeg, transform and several standard and [linalg] functions

author H.v.E., TUE-WFW, 1992

---

# reldeg(f,g,h,x,xnull,'rdeg','rtot','Adeg','bdeg);
# test for existence of a linearizing feedback:
# if not rtot=n then ERROR('for the present choice of output function(s), this system does not have a full-order (vector) relative degree. It is not possible to design an exact input-state linearizing feedback. You may try to define a(n) (set of) dummy output function(s) h(x) for which rtot=n by running -outputfunc- first or else to define an input-output linearizing feedback by running -inoutlin-') fi;

# the first step to determine an exact linearization is to compute the feedback $u(x)$ (in the original state coordinates)
$u(x) = alfa(x) + beta(x) \cdot v$

betax := linalg[Inverse](Adeg);
alfax := linalg[Multiply]((-betax),bdeg);

$u := map(simplify,map(normal,linalg[add]([linalg[multiply](beta(x),v)],alfa(x))));$
Appendix D

Listings of the procedures

# in transform, in the closed-loop system, this can be done easily, by
# multiplication with the jacobian of the mapping and substitution of the
# inverse mapping

# computation of the state transformation, the jacobian and the inverse
# transform(f,g,h,x,xnull,'phi','phiinv','jacobia','zeta','eta'):

# Ax = jacobia(f + g*alfax)
Ax := linalg[multiply]([jacobia, linalg[add](f, linalg[multiply](g, alfax))]);
Az := map(simplify, map(normal, subs(phiinv, op(Ax))));
if not type(Az, 'vector') then Az := linalg[col](Az, 1) fi:

# Bx = jacobia(g*betax)
Bx := linalg[multiply]([jacobia, linalg[multiply](g, betax)]);
Bz := map(simplify, map(normal, subs(phiinv, op(Bx))));
if not type(Bz, 'matrix') then Bz := evalm(Bz) fi:

if nargs > 9 then alfa := alfax; beta := betax fi:
op(u);
end: # end of procedure statelin
Appendix D

Listings of the procedures

## D.19

### D.

**Listings of the procedures**

#### D.19

```plaintext
#---> inoutlin(f,g,h,x,xnull,v,'u','f1','gl')
#---> inoutlin(f,g,h,x,xnull,v,'u','f1','gl','alfa','beta')

# inoutlin computes a static state feedback so that the closed loop system
# obtains a linear input-output behaviour. the feedback functions alfa(x) and
# beta(x) are constructed by the so called Structure Algorithm. Also the system
# equations of the linearized closed-loop system are returned. inoutlin is
# suitable for nonlinear SISO and square MIMO systems.

# A linear input-output behaviour means that the response of the closed loop
# system is given by the sum of the response under zero input, which is a
# (generally nonlinear) function of the time and initial state condition only,
# and of a response depending on the input and not on the initial state, which
# is linear in the input itself.

# input: f, g, h, x, xnull {for further information see reldeg}
#       v : the new reference input v. the dimensions of the array
#       v should equal those of the original input u, v may be
#       unassigned.
# output: (arrays)
#       u : the linearizing feedback u(x) = alfa(x) + beta(x) * v
#       fl, gl : the dynamics of the (linearized) closed loop system.
#       dx/dt = fl(x) + gl(x) v(x)
#       y(x) = h(x)
#       (optional)
#       alfa : the linearizing feedback alfa(x) and beta(x)

# inoutlin calls idiff, vdiff, extrank, and several standard and [linalg]
# functions
# author H.v.E., TUE-WFW, 1992

inoutlin := proc(f,g,h,x,xnull,v,u,fl,gl,alfa,beta)
local i,j,k,ii,jj,p,n,m,V,P,K,Khlp,Sinv,T,TO,Lfgamma,LgLf,Unit,c,
     dim,locgamma,deltatot,gammahlp,piv,Lggamma,localfa,locbeta,testl,test2,
     gamma,gammaln,gammalnprev,r,rprev,S,Sprev,delta,eqnbeta,expralfa,exprbeta,
     var alf a
     varbeta:

# tests on input
if not type(f,'vector') then ERROR('1st argument must be defined as a vector,
see your model description') fi:
if not type(g,'matrix') then ERROR('2nd argument must be defined as a matrix,
see your model description') fi:
if not type(h,'vector') or not linalg[vectdim](h)=n then ERROR('3rd argument must be defined as a vector,
see your model description') fi:
if not m=p then ERROR('the number of inputs and outputs should be equal') fi:
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('4th argument is
the state space variable vector and must be defined as a unassigned vector,
wrong type or number of elements, see your model description') fi:
if not type(xnull,'set') or not nops(xnull)=n then ERROR('5th argument must be
defined as a set, wrong type or number of elements, see your model
description') fi:
if not type(v,'vector') or not linalg[vectdim](v)=m then ERROR('the new
reference input, the 6th argument, must be defined as a vector which may
be unassigned') fi:

# initializations
deltatot := 0;
rprev := 0;
TO := array(1..m,1..m,[1]):
for r to m do
  for c to m do
    TO[r,c] := idiff(linalg[col](g,c),h[r],x);
  od:
  TO[r,c] := simplify(normal(TO[r,c]));
od:
```

The code provides a function `inoutlin` that computes a static state feedback to ensure the closed-loop system has a linear input-output behavior. The function constructs linearizing feedback functions $\alpha(x)$ and $\beta(x)$ based on the so-called Structure Algorithm. It also returns the system equations of the linearized closed-loop system. The function is suitable for nonlinear SISO and square MIMO systems.

The function takes inputs like $f$, $g$, $h$, $x$, $xnull$, and an output $v$, which represents the new reference input. The output includes the linearizing feedback $u(x) = \alpha(x) + \beta(x) \cdot v$ and the dynamics of the linearized closed-loop system.

The code snippet provides the necessary checks and initializations for the function to work correctly, ensuring that the inputs and outputs are of the expected types and dimensions.
Appendix D

Listings of the procedures

D.20

# start of the Structure Algorithm, recursive, maximal 2n steps
# step i of the Structure Algorithm
# from the previous step r.i-1 = rprev, gammaln.i-1 = gammalnprev,
# S.i-1 = Sprev, locgamma, and deltatot are known, occasionally some necessary
# exceptions in the case i=1 are made.

# create the matrix T
if i=1 then T := copy(T0) else
  for j to m do
    if j=1 then
      LgLf := vdiff(linalg[vec](g,1),vdiff(f,gammalnprev,x),x)
    else
      LgLf := linalg[vec](LgLf,vdiff(linalg[vec](g,j),vdiff(f,gammalnprev,x),x))
    fi:
  od:
fi: #if(i=1)

# set the rank of T (r), and the parameter delta
if not type(T,'matrix') then T := mkmatc(T) fi:
r := extrank(T):
delta := r - rprev;

# test whether xnull is a regular point of the matrix T
if not r = extrank(map(simplify,subs(xnull,op(T)))) then ERROR('the chosen working point xnull is not a regular point of the matrix T. step:',i) fi:

# test whether rR(T) = rK(T)
# ????
if r = m then
  # if this test is passed the Structure Algorithm terminates
  # and is concluded by setting gamma and S and updating locgamma and
  # deltatot
  if i=1 then S := copy(T0); locgamma := copy(h); deltatot := delta else
    gamma := map(simplify,map(normal,vdiff(f,gammalnprev,x))); for j to m do
      if j=1 then Lggamma := vdiff(linalg[vec](g,1),gamma,x)
    else
      Lggamma := linalg[vec](Lggamma,vdiff(linalg[vec](g,j),gamma,x))
    fi:
  od:
  if deltatot = 0 then locgamma := copy(gamma);
    S := Lggamma;
    deltatot := delta;
  else
    locgamma := linalg[stack](locgamma,gamma);
    S := linalg[stack](Sprev,Lggamma);
    deltatot := deltatot + delta;
  fi:
fi: #(i>1)

break; # end of the Structure Algorithm
fi: #(r=m)

# test whether the present step degenerates
if delta = 0 then
  # proceed with the next step and set the vector gammaln as follows
  # the matrix S stays Sprev
  if i=1 then gammaln := copy(h) else
    Khl := linalg[vec](linalg[transpose](T),dim);
    if not dim = (m-r) then ERROR('K, step:',i,'kernel mismatch') fi:
    for k to dim do
      if k=1 then K := op(1,Khl) else
        K := linalg[stack](K,op(k,Khl))
      fi:
      if dim = 1 then K := mkmatr(K) fi:
    od:
    if r = 0 then gammaln := vdiff(f,gammalnprev,x) else
if deltatot=0 then gammahlp:=vdiff(f,gammalnprev,x) else
gammahlp := linalg[stack]([locgamma,vdiff(f,gammalnprev,x)]) fi;
gammahlp := map(simplify,map(normal,linalg[multiply]([K,gammahlp])))
gammaln := array(1..r,[]);
for k to r do
  if type(gammahlp,'matrix') then gammaln[k] := gammahlp[k,1]
  else gammaln[k] := gammahlp[k] fi:
  od:
fi:

# note that the permutaion matrix P and so the matrix gamma do not
# exist in this case.
fi: #i=1
gammalnprev := copy(gamma1n);
rprev := r;
next; # next step i
fi: #(delta=0)

# creation of the matrices V, S, P, K which fulfil the partition conditions:
# (with I.d.1 stands for an identity matrix of dimensions corresponding to
# delta.1 etc.)
#
# V T =
# [ I.d.1 .. 0 0 ]
# [ .. .. .. ]
# [ 0 .. I.d(i-1) .. ]
# [ .. .. .. ]
# [ 0 .. 0 .. ]
# [ 0 .. 0 .. ]
# [ K.1 .. K.(i-1) K.i ]
#
# where V is a nonsingular matrix of real numbers, P performs row permutations
# and S(x) is a r x m matrix which has rank S(xnull)=r
piv := array(sparse,1..delta);
for k to delta do
  if k=1 then
    if deltatot=0 then
      # select the first nonzero row from T
      for j to m do
        if extrank(linalg[row](T,j))=1 then
          piv[k]]:=j;
          S := linalg[row](T,j); break; fi:
        od:
      else #(deltatot>0)
        # select the first occurring row from the added part of
        # T that raises the rank of Sprev and add this row to Sprev
        for j from (rprev+l) to m do
          if extrank(linalg[stack](Sprev,linalg[row](T,j)))=rprev+l
            then
              piv[k]:=j-rprev;
              S := linalg[stack](Sprev,linalg[row](T,j));
              break;
            fi:
        od:
      fi: #(deltatot=0)
    else #(k>1)
      # augment S with the next row that raises the rank
      for j from (piv[k]+1) to m do
        if extrank(linalg[stack](S,linalg[row](T,j)))=piv[k]+1
          then
            piv[k]:=j-piv[k]-1;
            S := linalg[stack](S,linalg[row](T,j));
            break;
          fi:
        od:
    fi:
  od: # now S and the pivots of P are known

# test on S
if r=1 then if not extrank(mkmatr(S))=r or not
  extrank(mkmatr(subs(xnull,S)))=r then
  ERROR('S:1, step:','1, 'rank of matrix S does not equal previous\results') fi:
else
  if not extrank(S)=r or not extrank(subs(xnull,S))=r then
    ERROR('S:1, step:','1, 'rank of matrix S does not equal previous\results')
else
  fi:
fi:
Appendix D

Listings of the procedures

D.22

results') fi;
fi:

# explicit creation of P
P := array(sparse,1..delta,1..m);
for k to delta do P[k,piv[k]] := 1 od:

# creation of K
Khlp := linalg[kernel](linalg[transpose](T),'dim');
if not dim = (m-r) then ERROR('K, step:',i,'the dimension of the null space
of matrix T does not compare with expected value') fi:
for k to dim do
  if k=1 then K := Khlp[1]
  else K := linalg[stack](K,Khlp[k]) fi:
od;
if dim=1 then K := mkmatr(op(K)) fi:

# creation (as above) and test of V and test of the partition
V := array(sparse,1..m,1..m);
for k to deltatot do V[k,k]:=1 od:
for ii from (deltatot+l) to r do
  for jj to m do
    V[ii,jj] := P[ii-deltatot,jj];
  od;
  od;
for ii from (r+l) to m do
  for jj to m do
    V[ii,jj] = K[ii-r,jj];
  od;
  od:
if not type(V,'matrix'(realcons)) then ERROR('V:1, step:',i,'the matrix V
computed by MAPLE does not entirely contain real numbers, V should be a
permutation matrix, this system does nor fulfil the demands for exact
IO linearization') fi:
if linalg[det](V) = 0 then ERROR('V:2, step:',i,'the matrix V computed
by MAPLE is singular, this system does nor fulfil the demands for exact
IO linearization') fi:
test1 := map(simplify,map(normal,linalg[multiply](V,T)));
test2 := linalg[stack](S,array(sparse,1..(m-r),1..m));
for ii to m do
  for jj to m do
    if not test1[ii,jj] - test2[ii,jj] = 0 then ERROR('V:3, step:',i,'the partition
computed by MAPLE seems to be not correct, the test V*T = [S 0] failed') fi:
  od;
  od:

# if all these tests are passed then step i is concluded by setting
# gamma and gammaln
if i=1 then gammahlp:=map(simplify,map(normal,linalg[multiply])(P,h))
else gammahlp := map(simplify,map(normal,
linalg[multiply])(P,vdiff(f,gammalnprev,x))));
fi: #(i=1)
gamma := array(1..delta,[]);
for k to delta do
  if type(gammahlp,'matrix') then gamma[k]:=gammahlp[k,1]
  else gamma[k]:=[gammahlp[k,1] fi:
od:

if i=1 then gammahlp:=map(simplify,map(normal,linalg[multiply])(K,h)))
else if deltatot=0 then gammahlp:=vdiff(f,gammalnprev,x) else
  gammahlp := linalg[stack](locgamm, vdiff(f,gammalnprev,x)) fi:
  gammaln := map(simplify,map(normal,linalg[multiply])(K,gammahlp)));
fi: #(i=1)
gammaln := array(1..r,[]);
for k to r do
  if type(gammahlp,'matrix') then gammaln[k]:=gammahlp[k,1]
  else gammaln[k] := gammahlp[k,1] fi:
  od:
Appendix D

Listings of the procedures D.23

# update the matrix which contains the functions gamma from the previous steps
# and the integer which stores the total of all delta's:
if deltatot = 0 then locgamma := copy(gamma) else
locgamma := linalg[stack](locgamma, gamma) fi;
deltatot := deltatot + delta;

gammalnprev := copy(gammaln);
rprev := r;
Sprev := copy(S);

od: # next step i

# the functions gamma that are created by the previous Structure Algorithm are
# stored in the vector locgamma. From these functions a linearizing feedback
# can be constructed in the following way:
# (recall that the matrix S = Lg-gamma)

# create the matrix Lf-gamma
if not type(locgamma, 'vector') then
locgamma := linalg[col](locgamma, 1) fi:
Lf gamma := vdiff(f, locgamma, x);
for i to r do Lfgamma[i] := -L * Lfgamma[i] od:

# create the matrix [ 1.r 0 ]
Unit := array(sparse, 1..r, 1..m);
for i to r do Unit[i, i] := 1 od:

if r = m then
    # create the matrix Sinv = inv(Lg-gamma)
    if n > 1 then Sinv := linalg[inverse](S) elif m = 1 then
        S := evalm(op(S)) fi:
        Sinv := array(1..1, 1..1, [(1, 1) = 1/S[1, 1]])
    fi:

    # compute the feedback alfa and beta
    localfa := map(simplify, map(normal, linalg[multiply](Sinv, Lfgamma)));
    locbeta := map(simplify, map(normal, linalg[multiply](Sinv, Unit)));
else #(rem)
    # it may occur that r < m, this is the case if the last (up to 2*n) steps
    # degenerates. S is not square and the inverse Sinv cannot be solved. the
    # equations: S*alfa=Lf-gamma and S*beta=Unit are solved in a convenient way

    # initialization
    localfa := array(1..m, []);
    locbeta := array(1..m, 1..m, []);
    varalfa := NULL;
    varbeta := NULL;
    for i to m do
        varalfa := varalfa, localfa[i];
    for j to m do
        varbeta := varbeta, locbeta[i, j];
    od;

    # alfa
    expralfa := NULL;
    for i to r do
        expralfa := expralfa, S[i] * localfa[i] - Lfgamma[i];
    od;
    traperror(solve({expralfa}, {varalfa}));
    if = lasterror then print("ERROR('MAPLE failed in computing the linearizing feedback')") else assign("map(simplify, map(normal, localfa))") fi:

    # beta
    exprbeta := NULL;
    for i to r do
        for j to m do
            eqnbeta := NULL;
            for ii to r do
                for jj to m do
                    eqnbeta := eqnbeta + S[ii, jj] * locbeta[ii, jj];
                od;
            od;
        od;
    exprbeta := exprbeta, eqnbeta = Unit[i, j];
    od;

traperror(solve({exprbeta},{varbeta}));
if "=lasterror then print(");ERROR('MAPLE failed in computing the"
linearizing feedback') else assign(");map(simplify,map(normal,locbeta)) fi:
fi: #(r=m)

# linearizing feedback u (output)
u:=map(simplify,map(normal,linalg[add](linalg[multiply](locbeta,v),localfa))):

# linearized system dynamics (output)
fl := map(simplify,map(normal,linalg[add](f,linalg[multiply](g,localfa))));
gl := map(simplify,map(normal,linalg[multiply](g,locbeta)));

# tests
if not type(fl,'vector') or not linalg[vectdim](fl)=n then ERROR('fl') fi:
if type(gl,'vector') then gl := mkmatrix(op(gl)) fi:
if not type(gl,'matrix') or not linalg[rowdim](gl)=n or
not linalg[coldim](gl)=m then ERROR('gl') fi:

# extra output
if nargs>9 then alfa := copy(localfa); beta := copy(locbeta) fi:

end: # end of the procedure inoutlin
Appendix D

Listings of the procedures

D.25

#---> outputfunc(f,g,x,xnull,'lambda')
# outputfunc tries to find a(n) (set of) dummy output function(s) lambda(i) so
# that the system gets a full order relative degree. outputfunc is suitable for
# SISO systems and square MIMO systems. If such a function does not exist, for
# SISO systems only, an output function (in some cases several functions) is
# searched for that maximizes the relative degree, which will be, however,
# strictly less than the dimension of the state space.
# input
# f,g,x,xnull (for further information see reldeg)
# output
# lambda : array containing output functions
# outputfunc calls reldeg, ldiff, liebrack, psolve, involutive, extrank,
# extcolsolve, and several standard and [linalg] functions
# author H.v.E., TUE-WFW, 1992

outputfunc:=proc(f,g,x,xnull, lambda)
local n,m,i,j,k,p,r,rr,argseq,ktel,tl,G,lf,kern,nulldim,mm,rdim,Adeg,rtot,
GREEN,GTOTAL,dlambda,tt,Gtotal,Ghlp,loclambda,var:
if not type(f,'vector') then ERROR('1st argument must be defined as a vector,')
see your model description') fi:
if not type(g,'matrix') then ERROR('2nd argument must be defined as a matrix,')
see your model description')
else m := linalg[vectdim](f) fi:
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('3th argument is
the state space variable vector and must be defined as a unassigned vector,')
wrong type or number of elements, see your model description') fi:
if not type(xnull,'set') or not nops(xnull)=n then ERROR('4th argument must be
defined as a set, wrong type or number of elements, see your model
description') fi:

FIRST PART
creation and test of involutivity and dimension of the distributions locG.0
# to logG.ktel

# step 0:
# distribution locG0 is created and tested on involutivity and constant
# dimension near xnull
locG.0 := copy(g);
# test on dimension of distribution locG0 near xnull
locGnull.0 := map(simplify,map(normal,subs(xnull,op(locG.0))));
lockdim.0:=extrank(locGnull.0);
if not lockdim.0=0 then #print('the dimension of the distribution,"
spanned by the columns of the inputmatrix',op(g),' is not equal to ',m,' near"
',op(xnull));
if the columns of locGnull.0 are dependent then we try to
# extract a matrix locG.0 which has maximal rank Grank before
# we proceed. select Grank columns of locG.0
for i from rr+l to m do
if i=1 then
# select the first nonzero column from g in locG.0
for j from rr+1 to m do
if extrank(mkmatc(linalg[col](g,j)))=1 then
locG.0 := linalg[col](g,j); rr := j; break;
ofi:
else
# augment locG.0 with the next column of g which raises the rank
for j from rr+1 to m do
if extrank(mkmatc(linalg[col](g,j)))=i then
locG.0 := linalg[augment](locG.0,linalg[col](g,j));
rr := j; break;
fi:
ofi:
else
locGnull.0 := map(simplify,map(normal,subs(xnull,op(locG.0))));
m:=extrank(locGnull.0);
if not m=lockdim.0 then ERROR('rank-mismatch') else print('the \ninput matrix used instead is:',op(locG.0)) fi:
fi:

# test on involutivity of distribution locG0=g
if m>1 then
argseq := [linalg[col](locG.0,1..m)];
if not involutive(argseq,x) then ERROR('the columns of the inputmatrix\n',op(g),'do not form an involutive distribution, searching for output functions\which give the system a full relative degree fails') fi:
fi:

# step ktel:
# creation and testing of distributions locG1 .. locGktel where ktel<=n-1,
# computing of an integer ktel for which dim(locGktel)=n (Isi. p.250). The
# matrices locG1 .. locGktel that correspond to the distributions, are
# simplified by replacing the original column vectors by vectors which are
# computed by the procedure extcolspace, i.e. vectors which form a basis for
# the spaces that is spanned by the distribution. In order to compute the right
# derivatives in the following steps the original columns are stored in the
# matrix Gtotal.
Gtotal := copy(g);
for ktel to n-1 do

  # compute the m new columns in the present step out of Gtotal, augment the
  # distribution locG.ktel-1 (G) with these vectors, create from this matrix
  # the distribution G.ktel by simplification via extcolspace and define the
  # distribution locGnull.ktel by substituting xnull
  r := (ktel+1)*m;
  G := copy(locG.(ktel-1));
  for j from (r-m+1) to r do
    Gtotal:=linalg[concat](Gtotal,1~ebrack(f,1ina1g[co1](Gtotal,~-m)),x));
  G := linalg[concat](G,linalg[col](Gtotal,~));
  od;
  Gelp := extcolspace(G);
  locG.ktel := Gelp[1];
  for i from 2 to nops(Gelp) do
    locG.ktel := linalg[augment](locG.ktel,Gelp[i]);
  od:
  # occasionally this "simplification" may lead to ERRORS because
  # for example state variables will be placed as denominator and
  # substitution of the working point xnull may cause division by
  # zero . if an error occurs, proceed with the "old" matrix.
  # what is needed is a "fraction free" version of gaussjord,
  # which is not available.
  locG.ktel := map(simplify,map(normal,locG.ktel));
  traperror(subs(xnull,op(locG.ktel)));
  if '='lasterror then locG.ktel:=copy(G) fi:
  locGnull.ktel := map(simplify,map(normal,subs(xnull,op(locG.ktel))));

  # test on the constant dimension of the simplified distribution
  # locGnull.ktel near xnull
  # extcolspace(locGnull.ktel,'lockdim.ktel');
  if not type(lockdim.ktel,realcons) then ERROR('the system does not\n  meet the constant dimension condition, Isi. Ch.5.2 p.250, the',ktel,'-th\ndistribution failed, searching for output\functions which give the system a relative degree fails') fi:

  # endcondition of the loop
  if lockdim.ktel=n then break
  elif ktel=n-1 and not lockdim.n-1 = n then
    ERROR('it is not possible to find an integer ktel for\
which all distributions locG1, 0<=i<=ktel-1<=n-2 are involutive, all\distributions locG1, 0<=i<=ktel<n-1 have a constant dimension, and\dim(locGktel)=n.') fi;

end
Appendix D

Listings of the procedures

D.27

# test on the involutivity of distribution locG.ktel
argseq := [linalg[col](locG.ktel, l..r)];
if not involutive(argseq, x) then
    if m>1 then ERROR('the system does not meet
    the involutive distribution condition, Isa. Ch.5.2 p.250, the ', ktel, ',th\n    distribution failed, searching for output functions which give the system a\n    relative degree fails')
else #(m=1)

# in the SISO case it is possible to compute an output function for
# which a feedback can be designed which will transform the system
# to a closed loop system which contains a linear subsystem of maximal
# dimensions
lprint('for this siso system an output function will be searched for,
which give the system a maximal relative degree, however strictly less than\nthe dimension of the state space');

# locG.ktel-1 = inv{ g, ..., ad_f `ktel-1 g} is the involutive closure
# of locG.ktel-1 (or at least involutive) and
# locG.ktel = colspace( g, ..., ad_f `ktel g) is not involutive

# By means of an iterative procedure the distribution locG.ktel is
# expanded to its involutive closure. Further, this distribution is
# expanded with the next derivative to locG.ktel+1 and these steps are
# repeated until the rank of the distribution which is obtained is
# equal to the dimension n of the state space.

r := extrank(locG.ktel);
var := ktel; # new count variable
while r<n do
    # locG.var is expanded to its involutive closure
tel := linalg[coldim](locG.var);
    for i to tel while r<n do
        for j from i+1 to tel while r<n do
            if not i=j then linalg[augment](locG.var,liebrack(linalg[col](locG.var,i),linalg[col](locG.var,j),x));
            if extrank(")=r+1 then
                lprint("r-1 then
                locG.var:=map(simplify,map(normal,")); r:=r+1;
                fi:
            fi:
        od:
    fi:
    while r<n do
        #add the next Lie derivative: create distribution locG.var+1
        Gtotal := linalg[augment](Gtotal, liebrack (f, linalg[col](Gtotal), var), x));
        tel := linalg[coldim](Gtotal);
        var := var + 1;
        locG.var := linalg[augment](locG.(var-1), linalg[col](Gtotal, tel));
    fi:
    Ghlp := extcolspace(locG.var);
    locG.var := Ghlp[1];
    for i from 2 to nops(Ghlp) do
        locG.var := linalg[augment](locG.var, Ghlp[i]);
    od:
    locG.var := map(simplify,map(normal,locG.var));
    r := extrank(locG.var)
    od: # repeat until r=n

    # the output function labda which enables us to create a linear
    # subsystem of maximal dimensions is now found:
    loclambda := traperror(psolve(locG.(var-1), x, xnull));
    if loclambda = lasterror then lprint(loclambda);ERROR('MAPLE failed in\nfinding involutive distribution condition, Isa. Ch.5.2 p.250, the ',
    order to find the output function');
    else
        tel := linalg[vectdim](loclambda);
        lambda := copy(loclambda);
        # test on this result(s):
Appendix D  Listings of the procedures  D.28

print('the output function(s) lambda computed by MAPLE do(es) give the\n  system a relative degree of ');  
  for i to tel do  
    loclambda:=array([1..m],[([l]=lambda[i])]);  
    reldeg(f,g,loclambda,x,xnull,'rdeg','rtot');  
    print(rdeg);  
  od:  
  RETURN(lambda);  
fi:  
fi:  
# next step ktel of the first part  

# SECOND PART:  
# now ktel is known for which all distributions locGi, 0<=i<=ktel-l<=n-2 are  
# involutive and all distributions locGi, 0<=i<=ktel<=n-1 have a constant  
# dimension near xnull, and finally dim(locGlotel)=n.  
# with this information we compute the first step in the iteration process to  
# define all m output functions lambda. in this first step the values of locm.1,  
# loclambda and GTOTAL are initialised.  
# step 1  
# there exist locm.1 functions lambda1 which fulfill the demands. they can be  
# constructed with the kernel of locGnull.(ktei-l)  
locm.1 := n - lockdim.(ktei-l);  
kern := linalg[kernel](linalg[transpose](locGnull.(ktei-l)),'nulldim');  
if not nulldim=locm.1 then ERROR('MAPLE failed in solving the kernel of the\n  distribution',op(locGnull.(ktei-l)),'although a ',locm.1,' dimensional\n  solution exists') fi:  
loclambda.1:=array([1..locm.1],[[j]=0]);  
  dlambda := op(1,kern);  
  if j=1 then  
    GTOTAL := linalg[concat](GTOTAL,dlambda)  
  else      
    for k to n do  
      loclambda.1[j] := loclambda.1[j] + int(dlambda[k],x[k]);  
    od;  
  od:  
  loclambda.1[j] := simplify(normal(loclambda.1[j]));  
  tt := locm.1;  
  mm := locm.1;  

# step i  
# if locm1<locm then there exist locm2 additional functions loclambda which can be  
# constructed in a similar way. if locm1+locm2 still turns out to be strictly  
# less than m there exist locm3 additional functions etc. every step i,  
# j=locm.i functions are defined  
for i from 2 to ktel while mm<m do  
  # i number of iteration steps, maximal ktel steps necessary  

# compute new derivatives and augment GTOTAL with these derivatives  
# at the last step i the columns of GTOTAL form a basis for locGlotel-orthogonal  
for p to i-1 do  
  # p indicates the category of the derivatives: cat. p=1 indicates  
  # loclambda.p ([1..locm1), category p=2 indicates (locm1+1..locm1+locm2)  
  # etc.  
  for j to locm.p do  
    # j indicates the separate functions loclambda within a category
if i=p+1 then 
lclf.p.j := ldiff(f, loclambda.p[j], x)
else 
lclf.p.j := ldiff(f, locLf.p.j, x)
fi:
if := array(1..n,[]);
for k to n do
if[k] := diff(locLf.p.j, x[k]);
end;
GTOTAL := linalg[concat](GTOTAL, lf);
end;
# next function j
end;
# next categorie p
if i=p+l then 
locLf.p.j := ldiff(f, loclambda.p[j], x)
fi:
if := array(l..n,[]);
for k to n do
if[k] := diff(locLf.p.j, x[k]);
end;
GTOTAL := linalg[concat](GTOTAL, lf);
end;
# next categorie p

# calculate locm.i the number of new functions loclambda in the present step:
locm.i := n - lockdim.(ktel-i) - (linalg[coldim](GTOTAL));
if locm.i > tt then tt:=locm.i fi;
if locm.i=0 then next fi;
# if no functions are found in the present step then proceed with the next step

# create GKERN and compute its kernel, test for the dimension of this
# nullspace:
GKERN := linalg[concat](locGnull.(ktel-i),GTOTAL);
kern := linalg[kernel]((linalg[transpose](GKERN)),'nulldim');
if not nulldim=locm.i then ERROR('MAPLE failed in solving the kernel of the
# distribution',op(GKERN),'although a ',locm.i,' dimensional solution exists.') fi;

# define j functions loclambda in the present step from the differentials which
# span kern by integration. GTOTAL is augmented with these differentials:
loclambda.i :=array(1..locm.i,[]);
for j to locm.i do
loclambda.i[j] := 0;
dlamb := op(j,kern);
GTOTAL := linalg[concat](GTOTAL,dlamb);
for k to n do
loclambda.i[j] := loclambda.i[j] + int(dlamb[k],x[k]);
end;
loclambda.i[j] := simplify(normal(loclambda.i[j]));
end;
# define mm= locm1+locm2+....
mm := mm + locm.i;
end;
# next step i of the iteration proces

# output lambda:
lambda := array(1..m,[]);
if m=1 then lambda[1]:=loclambda.1[1] else
j := 0;
for k to ktel while j<mm do
if locm.k=0 then next fi;
for i to locm.k do
j := j+1;
lambda[j] := loclambda.k[i]
end;
end;
fi;

# clear garbage
for i from 0 to ktel do
locG.i := evaln(locG.i);
locGnull.i := evaln(locGnull.i);
lockdim.i := evaln(lockdim.i);
end;
for i to ktel do
locm.i := evaln(locm.i);
loclambda.i := evaln(loclambda.i);
for j to tt do
locLf.i.j := evaln(locLf.i.j)
end;
Appendix D

Listings of the procedures D.30

# test on this result:
reldeq(f,g,lambda,x,xnull,'rdeg','rtot','Adeg','bdeg','cond');
if rtot<>n or linalg[det](Adeg)=0 then RETURN('the output function(s)'
lambda computed by MAPLE do not give the system a full order relative
degree. searching for such output function(s) has failed') fi:
op(lambda);
end:  # end of procedure outputfunc
tools package version 1.0
The tools package version 1.0 belongs to the zerodyn package version 1.0 and contains some useful and necessary tools for the procedures in zerodyn.
The tools package contains the following procedures:

ldiff : computes the derivative of a scalar valued function along a vector field
vdiff : computes the derivative of a vector valued function along a vector field
liebrack: computes the Lie-derivative or Liebracket of two vector fields
extrank : extension of the standard MAPLE procedure 'rank'
extgausselim : extension of the standard MAPLE procedure 'gausselim'
extgaussjord : extension of the standard MAPLE procedure 'gaussjord'
extcolspace/ extrowspace : extension of the standard MAPLE procedures 'colspace'/ 'rowspace'
transform : computes a state space transformation to a normal form
psolve : computes a solution to a (set of) partial differential equations.
extdsolve : this procedure provides a strategy to solve a set of ordinary differential equations on the basis of the standard MAPLE procedure dsolve.
involutive : test for an involutive distribution
mkmatr/
   mkmatc : rather elementary functions which perform a transformation from a vector sized array to a matrix sized array for computational reasons only.
mklog : provides facility to easily use the analysis and design tools in the zerodyn package
For further information see the "zerodyn package".
Information on a particular function can also be found in the header of the concerning procedure in the following listing.
Appendix D  

Listings of the procedures  

D.32

#-->
ldiff(f,h,x)
#
ldiff computes the derivative of a scalar valued function h(x) along a vector
field f(x) and is often written as Lfh.
#
# input
# f : n- dimensional array, vector valued function f(x)
# h : l- dimensional array, scalar function h(x)
# x : differentiation variable, defined as an unassigned
# n-dimensional vector
# result : scalar-valued function Lfh(x) = \frac{\partial h}{\partial x}
#
# author H.v.E., TUE-WFW, 1992
ldiff := proc(f,h,x)
local n,j,ld;
if not type(f,'vector') then ERROR('1st argument must be a vector') fi:
if type(h,'table') then if not linalg[vectdim](h)=l then ERROR('2nd argument
must be a scalar valued function') fi:
#else
# if not type(h,'scalar') then ERROR('2nd argument must be a scalar valued
# function') fi:
# else
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('3rd argument is
the variable vector and must be defined as an unassigned vector,
wrong type or number of elements, see your model description') fi:
ld:=0;
for j to n do
   ld := ld + diff(h,x[j])*f[j];
od:
simplify(normal(ld));
end:
# end of the procedure ldiff
### Appendix D

#### Listings of the procedures

D.33

---

### D.33

```plaintext

vdiff(f,h,x)

vdiff is equal to ldiff with a slight difference that the second argument is
not of the type 'scalar' but can be defined as a 'vector', i.e. a
1-dimensional array. So the result of vdiff is an array.

- **Input**
  - `f`: n-dimensional vector valued function `f(x)`
  - `h`: 1-dimensional vector valued function `h(x)`
  - `x`: Differentiation variable, defined as an unassigned
    n-dimensional vector

- **Result**
  - l-dimensional vector valued function `Lfh`

- **Author**
  - H.v.E., TUE-WFW, 1992

vdiff := proc(f,h,x)
local n,i,j,ld;

if not type(f,'vector') then ERROR('1st argument must be a vector')
  else n := linalg[vectdim](f) fi:

if not type(h,'vector') then ERROR('2nd argument must be a vector')
  else 1 := linalg[vectdim](h) fi:

if not type(x,'vector') or not linalg[vectdin](x)=n then ERROR('3rd argument is
the variable vector and must be defined as an unassigned vector,
wrong type or number of elements, see your model description') fi:

ld := array(1..l,[]);

for i to n do
  ld[i] := 0:
  for j to l do
    ld[i] := ld[i] + diff(h[i],x[j])*f[j];
  od;

ld[i] := simplify(normal(ld[i]));
od;

op(ld);
end: # end of the procedure vdiff
```

---

# Appendix D

## Listings of the procedures

D.33

---

### D.33

```plaintext

vdiff(f,h,x)

vdiff is equal to ldiff with a slight difference that the second argument is
not of the type 'scalar' but can be defined as a 'vector', i.e. a
1-dimensional array. So the result of vdiff is an array.

- **Input**
  - `f`: n-dimensional vector valued function `f(x)`
  - `h`: 1-dimensional vector valued function `h(x)`
  - `x`: Differentiation variable, defined as an unassigned
    n-dimensional vector

- **Result**
  - l-dimensional vector valued function `Lfh`

- **Author**
  - H.v.E., TUE-WFW, 1992

vdiff := proc(f,h,x)
local n,i,j,ld;

if not type(f,'vector') then ERROR('1st argument must be a vector')
  else n := linalg[vectdim](f) fi:

if not type(h,'vector') then ERROR('2nd argument must be a vector')
  else 1 := linalg[vectdim](h) fi:

if not type(x,'vector') or not linalg[vectdin](x)=n then ERROR('3rd argument is
the variable vector and must be defined as an unassigned vector,
wrong type or number of elements, see your model description') fi:

ld := array(1..l,[]);

for i to n do
  ld[i] := 0:
  for j to l do
    ld[i] := ld[i] + diff(h[i],x[j])*f[j];
  od;

ld[i] := simplify(normal(ld[i]));
od;

op(ld);
end: # end of the procedure vdiff
```
#---> liebrack(a,b,x)
# liebrack computes the liebracket of the two input vectors
# liebrack[a,b] = vdiff(a,b,x) - vdiff(b,a,x)
# input
# a : n-dimensional vector valued function a(x)
# b : n-dimensional vector valued function b(x)
# x : differentiation variable, defined as an unassigned
# n-dimensional vector
# result : n-dimensional vector valued function [a,b](x)
# author H.v.E., TUE-WFW, 1992

liebrack := proc(a,b,x)
local lievect,n,nn;
if not type (a,'vector') or not type (b,'vector') then ERROR('invalid arguments, should be vectors') else
n:=linalg[vectdim](a);nn:=linalg[vectdim](b) fi;
if not n=nn then ERROR('invalid arguments, vector dimensions should be equal') fi;
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('3th argument is the variable vector and must be defined as an unassigned vector, wrong type or number of elements, see your model description') fi;
lievect := array(1..n,[]);
lievect := linalg[add](vdiff(a,b,x),vdiff(b,a,x),1,-1);
lievect := map(simplify,map(normal,lievect));
op(lievect);
end: # end of the procedure liebrack
extrank(A) := proc(AA) local r, A;
    A := AA;
    if not type(A, 'matrix') then
        A := evalm(A);
        if A = 0 then RETURN(0) fi;
    fi;
    if nargs<>1 or not type(A, 'matrix')
        then ERROR(`wrong number or type of arguments`) fi;
    if type(A, 'matrix'(polynom(rationaì)))
        then linalg[ffgausselim](A, 'r')
        else extgausselim(A, 'r'); fi;
    fi;
    r
end:
# end of the procedure extrank
# Appendix D
# Listings of the procedures

## D.36 Gaussian Elimination:

This algorithm is based on the standard MAPLE 'gausselim' procedure. Few
ad-hoc changes are made to handle non-rational entries and to store which
element of each column is used as a pivot. These changes are marked with
the initials HvE.

### Purpose:
Reduce the matrix $A$ to triangular form.

### Optional argument:
- `rmar`: elimination to stop at column $rmar$

### Output:
- Function value: upper triangular matrix
- 'extrank': rank of $A$
- 'pivot': vector which contains the pivot elements of $A$
- 'det': determinant of $A$

### Original version 'gausselim':
Copyright 1990 by the University of Waterloo
See also 'gausselim' (HvE, may 1992)

```maple
extgausselim := proc(AA,extrank,pivot,det)
local A,B,d,n,m,i,j,c,r,rmar,pv,pvswap,setvar;
  # HvE
  A := AA;
  if not type(A,'matrix') then A := evalm(A) fi;
  if not type(A,'matrix') then ERROR('1st argument must be a matrix') fi;
  n := linalg['rowdim'](A);
  m := linalg['coldim'](A);
  B := array(l..n,l..m);
  setvar:=0; # HvE
  for i to n while setvar=0 do
    for j to m do
      if not type(A[i,j],ratpoly(rational)) then setvar := 1;
      B := map(normal,A);
      B := map(simplify,A);
      break
    fi; # HvE
    B[i,j] := normal(A[i,j]);
  od;
  if nargs>i and type(args[2],'integer') then
    rmar := args[2]; if rmar<0 or nargs<2 then ERROR('invalid arguments') fi;
    # HvE
    pv:=array(1..n,[]);
    pvs := array(1..m,[]);
    for i to n do
      pvs[i]:=i
    od;
    r := 1;
    d := 1;
    for c to min(m,rmar) while r <= n do
      if printlevel>2 then lprint('extgausselim: elimination at row',c) fi;
      # Search for a pivot element
      for i from r to n while B[i,c] = 0 do od;
      for j from i+1 to n do
        if B[j,c] = 0 then next fi;
        if length(B[j,c]) < length(B[i,c]) then i := j fi;
      od;
      if i <= n then
        pv[c] := pvs[i]; # HvE
        # interchange row i with row r if necessary
        if i <> r then
          # ...
        fi;
      fi;
    od;
  fi;
end proc;
```

# Copyright 1990 by the University of Waterloo
# See also 'gausselim' (HvE, may 1992)
if nargs>2 and not type(args[2],integer) then
  t:=pvswap[i];pvswap[i]:=pvswap[r];pvswap[r]:=t;
  fi: #HvE
  d := -d;
  for j from c to m do t := B[i,j]; B[i,j] := B[r,j]; B[r,j] := t od
  fi;
if nargs>3 and not type(args[2],integer) then d := d*B[r,c] fi: #HvE
for i from r+1 to n do
  t := simplify(normal(B[i,c]/B[r,c])); #HvE
  for j from c+1 to m do
    B[i,j] := simplify(normal(B[i,j]-t*B[r,j])) #HvE
  od;
  B[i,c] := 0
od;
  r := r + 1          # go to next row
fi;
  # go to next column

B := subs('A'=A,op(B)); d := subs('A'=A,d);
if has(op(B),'A') or has(d,'A') then ERROR('undefined matrix elements') fi;
if nargs>1 and not type(args[2],integer) then extrank := r-1 fi;
if nargs>2 and not type(args[2],integer) then
  pivot:=array(1..(r-1),[]);
  for i to (r-1) do
    pivot[i] := pv[i]
  od; #HvE
  op(B);
  fi;
if nargs>3 then if n = r-1 then det := simplify(normal(d)) else det := 0 fi fi:
  #HvE
  op(B);
end:          # end of procedure extgausselim
Gaussian Elimination:
This algorithm is based on the standard MAPLE 'gaussjord' procedure. Few
ad-hoc changes are made to handle non-rational entries and to store which
element of each column is used as a pivot. These changes are marked with the
initial HvE.

Purpose: Reduce the matrix A to diagonal (Gauss Jordan) form.
optional argument:
rmar : elimination to stop at column rmar

Output: Function value: reduced matrix
optional parameters:
'rank' : rank of A
'pivot' : vector which contains the pivot elements of A
'det' : determinant of A

original version 'gaussjord' : Copyright 1990 by the University of Waterloo
See also 'gaussjord','extgausselim'

---

extgaussjord(A,rmar)
#-->
extgaussjord(A,'rank','pivot','det')

# Gaussian Elimination:
# This algorithm is based on the standard MAPLE 'gaussjord' procedure. Few
# ad-hoc changes are made to handle non-rational entries and to store which
# element of each column is used as a pivot. These changes are marked with the
# initial HvE.
#
# Purpose: Reduce the matrix A to diagonal (Gauss Jordan) form.
#optional argument:
# rmar : elimination to stop at column rmar
#
# Output: Function value: reduced matrix
#optional parameters:
# 'rank' : rank of A
# 'pivot' : vector which contains the pivot elements of A
# 'det' : determinant of A
#
# original version 'gaussjord' : Copyright 1990 by the University of Waterloo
# See also 'gaussjord','extgausselim'
#
# (HvE, may 1992)

extgaussjord := proc(AA,rank,pivot,det)
local A,B,d,n,m,i,j,c,r,t,rmar,pv,pvswap,setvar;
#HVE
A:=AA;
if not type(A,'matrix') then
A:=traperror(evalm(A));
if A=lasterror then ERROR('invalid arguments') fi;
if A=O then ERROR('first argument is zero, need zero matrix') fi;
fi;
if not type(A,'matrix') then ERROR('1st argument must be a matrix') fi;

n := linalg['rowdim'](A);
m := linalg['coldim'](A);
B := array(l..n,l..m);
setvar:=0; #HVE
for i to n while setvar=0 do
for j to m do
if not type(A[i,j],ratpoly(rational)) then setvar := 1;
B := map(norma1,A);
B := map(simp1ify,A);
break
fi; #HVE
B[i,j] := normal(A[i,j]);
end;

if nargs>1 and type(args[2],'integer') then
rmar := args[2]; if rmar<0 or nargs>2 then ERROR('invalid arguments') fi;
else rmar := m fi;

pv:=array(sparse,1..rmar,[]); #HVE
pvswap:=array(1..n,[]);
for i to n do
pvswap[i]:=i;
od;

r := 1;
d := 1;
for c to min(m,rmar) while r <= n do

# Search for a pivot element i in col c
for i from r to n while B[i,c] = 0 do od;
for j from i+1 to n do
if B[j,c] = 0 then next fi;
if length(B[j,c]) < length(B[i,c]) then i := j fi
od;

```
Appendix D

Listings of the procedures

D.39

if i <= n then
  pv[c] := pvswap[i];
  # interchange row i with row r if necessary
  if i <> r then
    if nargs>2 and not type(args[2], 'integer') then
      r := pvswap[i]; pvswap[i] := pvswap[r]; pvswap[r] := t;
      fi; #HVE
      d := -d;
      for j from c to m do t := B[i,j]; B[i,j] := B[r,j]; B[r,j] := t od;
    fi;
    for j from c+1 to m do B[r,j] := simplify(normal(B[r,j]/B[r,c])) od; #HVE
    if nargs>3 then d := d*B[r,c] fi; #HVE
    B[r,c] := 1;
    for i to n do
      if nargs>2 and not type(args[2], 'integer') then
        pivot := array[1..(r-l),[]];
      fi;
      j := 1;
      for i to n while j <= (r-l) do
        if pv[i] = 0 then next fi;
        pivot[j] := pv[i];
        if not type(pivot[j], 'integer') then ERROR('pivot mismatch') fi;
        j := j + 1;
      od;
    fi;
  fi;
  if nargs>3 then if n = r-1 then det := simplify(normal(d)) else det := 0 fi; fi;
  #HVE
  op(B) := subs('A'=A, op(B));
end:
  # end of procedure extgaussjord
#---> extrowspace(A,'dim')
#---> extcolspace(A,'dim')
#
# Computes the row (column) space of the matrix A and optionally
# assigns dim, the dimension of the row (column) space = rank(A).
# The row (column) space returned is a set of vectors.
#
# original version linalg[colspace]/[rowspace]
# Copyright 1990 by the University of Waterloo

extrowspace := proc(A,dim) local B,rank;
    B := extgaussjord(A,'rank'); #ReE
    if nargs = 2 then dim := rank fi;
    { linalg['row'](B,1..rank) }
end:

extcolspace := proc(A,dim)
    extrowspace( linalg[transpose](A), args[2..nargs] )
end:
Appendix D
Listings of the procedures

D.41

# transform computes a state space transformation to the normal form. optional
# the inverse transformation and the jacobian matrix of the transformation are
# computed and tested. also the new state variables zeta and eta are then
# available in the output. transform is used as a tool inside procedures
# from the zero-dyn package (i.e. normform and statelin). The transformation is
# built up in the following way. The first rtot elements of the mapping are
# defined by the output functions h. The last n-rtot transformations may be
# constructed in two manners. First, MAPLE tries to solve the partial differential
# equation \( L_D = 0 \). If this succeeds, the transformation is completed and a
# message is returned. Else, if MAPLE fails to solve the differential equation
# (or cannot find enough solutions), the transformation is completed by adding
# unit vectors to the jacobian of the transformation. In this case no message is
# returned.

# input f, g, h, x, xnull (for further information see reldeg)
# output
# phi : array which contains the state transformation to a normal form
# (optional output:)
# philinv : set which contains the inverse transformation
# alfa : jacobian matrix of the transformation phi(x)
# zeta : unassigned rtot dimensional vector representing the first rtot
# new state variables
# eta : unassigned n-rtot dimensional vector representing the last
# n-rtot new state variables
#
# transform calls reldeg, psolve and extgaussjord
#
# author H.v.E., TUE-WFW, 1992

transform := proc(f, g, h, x, xnull, phi, philinv, alfa, zeta, eta)
local i, j, k, ralfa, palfa, setvar, r, eqnseq, varseq, n, m, p, Adeg, rdeg, rtot, \n    z, solvar, rtest, t, localfa, phiini, phidiff, phiun, rr;
if not nargs=6 and not nargs=10 then
    ERROR(‘not enough arguments, should be 6 or 10’): fi;
if not type(f, ‘vector’) then ERROR(‘1st argument must be defined as a vector, \n    see your model description’): fi;
if not type(g, ‘matrix’) then ERROR(‘2nd argument must be defined as a matrix, \n    see your model description’): fi;
if not type(h, ‘vector’) then ERROR(‘3rd argument must be defined as a vector, \n    see your model description’): fi;
if not m>=p then ERROR(‘the number of inputs should be larger than or equal to \n    the number of outputs’): fi;
if not type(x, ‘vector’) or not linalg[vectdim](x)=n then ERROR(‘4th argument is \n    the state space variable vector and must be defined as an unassigned vector, \n    wrong type or number of elements, see your model description’): fi;
if not type(xnull, ‘set’) or not nops(xnull)=n then ERROR(‘5th argument must be \n    defined as a set, wrong type or number of elements, see your model \n    description’): fi;

phiini := array(1..n,[]):
localfa := array(1..n,1..n,[]):
rtest := 0;
reldeg(f, g, h, x, xnull, ‘rdeg’, ‘rtot’, ‘Adeg’):
# now the local parameters rdeg, Adeg and rtot are defined

# test for non-singularity of Adeg
if e=p then if linalg[det](Adeg)=0 then ERROR(‘non-singularity matrix to test \n    the relative degree of the system is singular, it is not possible to find a \n    state transformation to the normal form.’): fi;
else if extrank(Adeg)=p then ERROR(‘non-singularity matrix to test \n    the relative degree of the system is singular, it is not possible to find a \n    state transformation to the normal form.’): fi;
fi:
# the first rtot=rl+..+rp state transforms are equal to hi, lfhi, .. ,
# lfh(ri-1)hi . (see isi prop(5.1.3)).
j := 0:
for i to p do
  j := j + 1;
  phiini[j] := h[i];
  for k from 2 to rdeg[i] do
    j := j + 1;
    phiini[j] := ldiff(f, phiini[j - 1], x);
  od;
od;

# the last n-rtot transformations phi have to satisfy the condition that the # jacobian matrix of the mapping phi has full rank (i.e. is non-singular). a # better but much stronger condition is that the mapping fulfills the set of # differential equations ldiff(g, phi, x) = 0. if this is the case then the input # u will not appear in the remaining equations.
# first we try to find solutions to this set of equations with aid of the # procedure psolve. in case such solutions do not exist or MAPLE is not able to # find them, then the transformation is augmented with unit vectors.

# start adding transformations (if necessary)
if rtot=n then phi:=copy(phiini);
  localfa:=linalg[transpose][linalg[jacobian](phi, x)];
else # start calculation of additional transformations
  # start solving differential equations, call of psolve
  z := traperror(psolve(g, x, xnull));
  # in cases where MAPLE succeeds in solving these equations the solution is # stored in the array z with n-m independent solutions. in general we need # n-rtot of these solutions in such a way that the jacobian matrix of # the transformation phi is non-singular.
  # note that always: rtot >= p and m >= p
  # test on z
  solvar:=false;
  if z=lasterror or traperror(op(z))=lasterror then print(z)
  elif (type(op(z), 'array')) and (linalg[vectdim](z)=(n-m)) then
    # solution z fulfills the basic conditions, now create the transformation # phi out of z. selection of n-rtot independent solutions in phi.
    phidiff := copy(phiini);
    localfa:=linalg[transpose][linalg[jacobian](phi, x)];
    ralfa:=extrank(localfa);
    # sort all n-m solutions in array z in order of increasing complexity
    for i from 1 to (n-m) do
      for j from i to (n-m) do
        if length(z[j]) < length(z[i]) then
          t := z[i]; z[i] := z[j]; z[j] := t;
        fi;
      od:
    od:
    # add successively a solution to phi and test if the rank of the jacobian # increases.
    k:=1;
    for i from rtot+1 to n do
      setvar:=false; j:=0;
      while not setvar and j<(n-m+1) do
        j := j + 1;
        phidiff[i] := z[k];
        localfa:=linalg[transpose][linalg[jacobian](phi, x)];
        test := extrank(localfa);
        if test > ralfa then
          ralfa := test; setvar := true fi;
          k := k + 1;
        if k>(n-m) then break fi;
        od:
        if k>(n-m) then break fi:
      if not setvar then phidiff[i] := 0; break; fi;
      od:
    fi:
  fi:
# testing phidiff, non-singularity of localfa, an essential condition.
  if not linalg[det](localfa)=0 then solvar:=true fi:
else
  # solution z does not fulfill the basis conditions
  print(z)
fi:
# end of test on z end of solving the differential equations
if solvar then phi := copy(phidiff); print('MAPLE succeeded in solving the
differential equations ldiff(g, phi, x)') else
  # If no or not enough (i.e. this may be the case where the number of
  # inputs is larger than the number of outputs) solutions of the
  # differential equations can be found, the mapping is completed with
  # unit vectors in such a way that the jacobian has full rank.
  # start of adding unitvectors.
  if rtest > rtot then phiun := copy(phiini); rr := rtest
  else phiun := copy(phidiff); rr := rtot fi;
  localfa := linalg[transpose](linalg[jacobian](phiun, x));
  # the last n-rtot (or, in case a part of phidiff can be used, n-rtest)
  # transformations phi are now computed by adding one unitvector at the
  # time according to the pivots that will be found in extgaussjord
  extgaussjord(localfa, 'ralfa', 'palfa');
  if not ralfa = rr then
    ERROR('rank-mismatch, rank from extgaussjord does not agree
  with previous results from transform') fi;
  r := rr;
  for i to n do
    setvar := 'true';
    for j to rr do
      if palfa[j] = i then setvar := 'false'; break; fi;
    od;
    if setvar = 'true' then r := r+1; phiun[r] := x[i] fi;
  od;
  if not r = n then ERROR('pivot-mismatch, number of pivots from
  extgaussjord does not compare with the number of unknown transformations') fi;
  localfa := linalg[transpose](linalg[jacobian](phiun, x));
  # test for non-singularity of localfa
  if linalg[det](localfa) = 0 then ERROR('jacobianmatrix of the mapping phi,
  computed by adding unitvectors, is singular. no solution computed for
  transformation to normal form') else phi := copy(phiun) fi:
fi:  # end of adding unitvectors

fi:  # end of calculation of additional transformations phi

# now all phi[1..n] state transformations are known. the new coordinates are
# called zeta and if rtot<n also eta. zeta/eta = phi(x). inverse
# transformation is called phiinv, calculation of phiinv:
if nargs > 6 then
  zeta := array(1..rtot, []);
  eqnseq := NULL;
  varseq := NULL;
  for k to rtot do
    eqnseq := eqnseq, zeta[k] = phi[k]:
    varseq := varseq, x[k]:
  od:
  if rtot < n then
    eta := array(1..(n-rtot), []);
    for k from (rtot+1) to n do
      eqnseq := eqnseq, eta[k-rtot] = phi[k]:
      varseq := varseq, x[k]:
    od:
  fi:
  phiinv := traperror(solve({eqnseq}, {varseq}));
fi:  # end of calculation of additional transformations phi

if phiinv = lasterror then print(phiinv); RETURN('MAPLE failed
in solving the inverse state transformation. this inverse is necessary for
computation of the normal form.') else
  for i to n do
    if not simplify(eval(subs(op(phiinv), eqnseq[i]))) then
      RETURN('MAPLE failed in solving the inverse state transformation. this inverse
is necessary for computation of the normal form.') fi:
  od:
fi:
# assign output alfa
localfa := linalg[transpose](localfa);
alfa := op(localfa);
fi;

end:  # end of procedure transform
# Lists of the procedures

D.45

--->
\texttt{psolve(gg,x)}

# psolve tries to solve the (set of) partial differential equation(s) \( L gg \phi(x) = 0 \) for all \( x \) near \( x_{\text{init}} \). The algorithm is based on a constructive proof of the Frobenius theorem. This theorem proves that a sufficient and necessary condition for the existence of \( n-m \) independent solutions is the involutivity or complete integrability of the distribution spanned by the column vectors of the input matrix \( gg \).

# \textbf{Input} \( gg \) : \( nxm \) array whose entries are functions of \( x \) in such a way that the partial differential equation
\[
\begin{bmatrix}
gg[1,1] & \cdots & gg[1,m] \\
gg[2,1] & \cdots & gg[2,m] \\
\cdots & \cdots & \cdots \\
gg[n,1] & \cdots & gg[n,m] 
\end{bmatrix}
\]
will be solved around the point \( x = x_{\text{init}} \).

# \textbf{x} : unassigned \( n \) dimensional variable array (optional:)

# \textbf{xinit} : set which contains the initial conditions of the variables \( x \) in the form \( \{x[1]=\ldots, x[n]=\ldots\} \) if this initial condition is not set then the symbolic values \( \text{locx0.}_i \) will be used in the internal computations.

# \textbf{Output} (function result:)

# string with message: 'negative testresult' or 'can't find solutions'

# array with solutions

# The way this procedure must be called (i.e. by mean of an array and vectors of variables and initial conditions) differs from the general way in which the \textsc{Maple} "solve procedures" are called (i.e. by mean of sequences of equations and variables respectively).

# This difference is due to properties of the constructive proof that is used in the algorithm and also to the main purpose of this procedure in the zerodyn package where it is used to compute state transformations to the normal form.

# However, the general use of this procedure in order to solve a (set of) partial differential equation(s) is stressed.

# \texttt{psolve calls involutive, dsolve,extdsolve}

# author H.v.E., TUE-WFW, 1992

psolve := proc(gg,x,xinit)
local i,F,Finv,G,Grank,argseq,eqnseq,varseq,exprseq,locxinit,\flow,g,hulpvar,j,allz,m,n,r,rr,s,setvar,z,gtestrFtest:
if nargs<2 or nargs>3 then ERROR('invalid number of arguments') fi;
if not type(gg,'matrix') then ERROR('1st argument must be defined as a matrix, see your model description') fi:
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('2nd argument is must be defined as an unassigned vector, wrong type or number of elements, see your model description') fi:
if nops(xinit)>2 then
if not type(xinit,'set') or not linalg[vectdim](x)=n then ERROR('3rd argument must be defined as a set, wrong type or number of elements, see your model description') fi:
locxinit := NULL;
for i to n do
locxinit := locxinit,x[i]=locx0.i
od;
locxinit := {locxinit};
fi;

# test for the existence of a solution to the (set of) partial differential equation(s):
# a solution does only exists if and only if the distribution spanned by the column vectors of \( gg \) is involutive. In case of a single column this condition is always satisfied
if not m = 1 then
argseq := [linalg[col](g,1..m)];
if not involutive(args, x) then RETURN('the distribution spanned by \nthe columns of the input matrix is not involutive, there exist no n-m linear \nindependent solutions for the partial differential equations.'); fi:

# create G, the nxn augmented input matrix which has rank n, the matrix gg is # augmented with appropriate unit vectors
G:=g;
extgaussjord(G, 'Grank', 'Gpivot');
r:=Grank;
if Grank=0 then RETURN('there is no nontrivial solutions'); fi:

for i to m do
    if i=1 then
        select the first nonzero column from gg in g
    else
        augment g with the next column of gg which raises the rank
    fi:
    od:

G := g;
extgaussjord(G, 'Grank', 'Gpivot');
if not Grank=r then ERROR('rank mismatch'); fi:

for i to n do
    for j to m do
        hulpvar := array(sparse, 1..n);
        setvar := 'true';
        for j to m do
            if Gpivot[j]=i then setvar := 'false'; break; fi:
        od:
        if setvar = 'true' then
            if r=n then break fi:
        fi:
    done:
    if not r=n then ERROR('pivot-mismatch, invalid number of pivots'); fi;
    if not extrank(G)=n then ERROR('MAPLE failed in creating an augmented \ninput matrix which has full rank. MAPLE tried to do this by adding \nunitvectors.'); fi:

for i to n by -1 to 1 do
    g:=linalg[concat](G, hulpvar[i]);
    # start with the last columnvector of G in connection with backward substitution of the successive flows that are computed in the mapping F
    g:=map(simplify, map(normal, subs({exprseq}, op(g))));

    exprseq:=NULL;
    for j to n do
        locx[j]:=evaln(locx[j]);
        locx0[j]:=evaln(locx0[j]);
        exprseq:=exprseq, x[j]=locx[j](locz[i]);
    od:
    g:=map(simplify, map(normal, subs({exprseq}, op(g))));

    exprseq:=exprseq, exprseq, x[j]=locx[j](locz[i]);
    od:

Appendix D  
Listings of the procedures  

D.47

# solve the flow

# at the time two procedures to solve the set of ordinary differential
# equations that together form the flow are available.
# in the first place we can try to do so with the standard MAPLE procedure
# dsolve. However, this procedure turns out to be not very strong and has
# serious difficulties in solving sets of equations.
# in the second place a strategy is available to solve the n differential
# equations successively in order of increasing complexity while
# substituting the solutions of the previous equations. this is
# implemented in the procedure extdsolve. in some cases this procedure
# returns better solutions.

flow := traperror(extdsolve(g,x,locz.i));
if flow=lasterror then
   print('extdsolve:',flow,'MAPLE procedure
dsolve failed in solving the flow of ',op(g));
fi:

# solution of the i-th flow with the standard procedure dsolve

eqnseq := NULL;
varseq := NULL;
for j to n do
   eqnseq:=eqnseq,diff(locx.j(locz.i),locz.i)=q[j],locx.j(0)=locx0.j;
   varseq:=varseq,locx.j(locz.i);
od:
flow := traperror(dsolve({eqnseq},{varseq}));
if flow=lasterror then
   ERROR('extdsolve:',flow,'MAPLE procedure
dsolve failed in solving the flow of: ',op(g))
fi:

# now the i-th flow is known and can be simplified and tested
flow:=map(simplify,flow);

# test this solution
qtest := array(1..n,[j]);
for j to n do qtest[j]:=x[j] od:
qtest := map(simplify,map(normal,subs({exprseq},op(qtest))));
qtest := map(simplify,map(normal,subs(flow,op(qtest))));
for j to n do
   if not simplify(normal(diff(qtest[j],locz.i)) - q[j])=0 then print('MAPLE
failed in solving the flow of the ',i,'-th column of ',op(G),' although\nthis flow exists.');ERROR(); fi;
   od:

# creation of the mapping F from the computed flow(s) by mean of backward
# substitution
# store results of the i-th flow in the array flowhelp.i
assign(flow);
flowhelp.i:=array(1..n,[i]);
for j to n do
   flowhelp.i[j]:=locx.j(locz.i);
   od:
if not i=n then
   exprseq:= NULL;
   for j to n do
      exprseq:=exprseq,locx0.j=flowhelp.(i+1)[j];
      od:
   flowhelp.i := subs({exprseq},op(flowhelp.i));
fi:
F:=flowhelp.i;

# END LOOP, next flow i
# now the mapping F is known as a function of loczl..loczn and locx01..locx0n.
# when xinit is substituted in F then F is just a function of loczl..loczn.

# substitution of locx0.j=x[j] in F for substitution of xinit, which is of

exprseq:=NULL;
for j to n do
  exprseq:=exprseq,locx0.j=x[j];
od:
F:=subs({exprseq},op(F));
F:=map(simplify,map(normal,subs(xinit,op(F))));

# calculation of Finv
eqnseq:= NULL;
varseq:= NULL;
for j to n do
  eqnseq:=eqnseq,locx.j=F[j];
  varseq:=varseq,locz.j;
od:
Finv:=traperror(solve({eqnseq},{varseq}));

# test this solution
if Finv=lasterror then
  print(Finv);ERROR('MAPLE failed in\nsolving the inverse mapping of F',op(F),' although this inverse exists.')
else
  Ftest := copy(F);
  Ftest := map(simplify,map(normal,subs(Finv,op(Ftest))));
  for j to n do
    if not simplify(normal(Ftest[j]-locx.j))=0 then
      ERROR('MAPLE failed in\nsolving the inverse mapping of F',op(F),' although this inverse exists.')
    fi:
  od:
  fi;  #(Finv=lasterror)

assign(Finv);

# (the last) n-m functions of the n functions z (in Finv) are independent
# solutions of the partial differential equations Lg(phi)(x)=0.
# create output z

# create locz, with all n elements
allz:=array(1..n,[]);
for j to n do
  allz[j]:=locz.j;
  allz[j]:=simplify(normal(allz[j]));
od:

# back substitution, locx.j = x[j]
exprseq:=NULL;
for j to n do
  exprseq:=exprseq,locx.j=x[j];
od:
allz:=subs({exprseq},op(allz));

# test and selection for output z
z:=array(1..(n-m),[]); s:=0;
for i from n by -1 to 1 do
  setvar:=0;
  for j to m do
    if simplify(normal(linalg[col][gg,j],allz[i],x))) <> 0 then
      setvar:=1
    fi:
  od:
  if setvar=0 then s:=s+1; z[s]:=allz[i]; fi:
  if s=n-m then break fi:
  od:
if not(s=n-m) then RETURN('MAPLE failed in (completely) solving the partial\ndifferential equations Lg(phi)(x), although n-m independent solutions exist."
However, possible solutions that already are found are,';op(z)) fi:
Appendix D

Listings of the procedures

# clear garbage
for i to n do
  locz_i := evaln(locz.i);
  locx_i := evaln(locx.i);
  locx0_i := evaln(locx0.i);
  flowhelp_i := evaln(flowhelp.i);
  od;

op(\%);
end: # end of the procedure psolve
Appendix D  Listings of the procedures  D.50

# --> extdsolve(gg,x)
# --> extdsolve(gg,x,flowvar)
# --> extdsolve(gg,x,flowvar,xinit)

if the MAPLE procedure -dsolve- failed in solving a set of n ordinary
differential equations, it may be possible to solve, instead of all equations
simultaneously, these n differential equations successively in order of
increasing complexity while substituting the solutions of the previous
equations. This is done in the procedure extdsolve.

input gg : nx1 array whose entries are functions of x in such a way
that the n equations

\[
\begin{align*}
\frac{dx}{flowvar} &= gg[1](x) \\
\frac{dx}{flowvar} &= gg[2](x) \\
\cdots &= gg[n](x)
\end{align*}
\]

will be solved.

x : unassigned n dimensional variable array

flowvar : symbolic name of the temporary flow variable. if this\argument isn't used then 'z' will be used.
xinit : set which contains the initial conditions of the variables
in the form \{x[1]=..., x[n]=...\} if this argument is not
set then the symbolic names locx0.i will be used.

output (function value):
n dimensional array which contains the flow of array gg, a\smooth function of flowvar and x with the property that the\functions flow(xinit) = x(flowvar) solves the set of ordinary\differential equations stated above.

the way this procedure is written differs from the way the MAPLE procedure
"dsolve" is written. This departure is due to the ad-hoc use of extdsolve in
the procedure psolve. Note that also this procedure may easily fail to solve
a set of ordinary differential equations.

extdsolve calls dsolve


extdsolve := proc(gg,x,flowvar,xinit)
local piv,kk,ll,tt,eseq,vseq,flow,flowex,j,n,z,locxinit,i,exprseq:
if not type(gg,'vector') then ERROR('1st argument must be defined as a vector,\see your model description')
ext else n:=linalg[vectdim](gg) fi:
if not type(x,'vector') or not linalg[vectdim](x)=n then ERROR('2nd argument is\must be defined as an unassigned vector, wrong type or number of elements, see\your model description') fi:
if nargs>3 then locxinit:=copy(xinit);# substitution of x[j]=locx.j(z) for calculation reasons (for\# differentiation explicit functions of z are necessary)
else locxinit:={locxinit}
if nargs>3 then locxinit:=NULL; for i to n do
locx.j:=evaln(locx.j);# substitution of x[j]=locx.j(z) for calculation reasons (for\# differentiation explicit functions of z are necessary)
else locxinit:=NULL; for i to n do
locx.j:=evaln(locx.j);# substitution of x[j]=locx.j(z) for calculation reasons (for\# differentiation explicit functions of z are necessary)
else locxinit:=evaln(flowvar) fi:
if nargs>2 then z := evaln(flowvar) else z := evaln(z) fi:

# --> extdsolve(gg,x)
# --> extdsolve(gg,x,flowvar)
# --> extdsolve(gg,x,flowvar,xinit)

# --> extdsolve(gg,x,flowvar)
# --> extdsolve(gg,x,flowvar,xinit)
Appendix D  Listings of the procedures  D.51

\begin{verbatim}
tt:=piv[kk];piv[kk]:=piv[ll];piv[ll]:=tt
od:

# creating and solving the j-th differential equation while substituting
# the solutions of previous equations of gg. The solutions are stored in
# the array flow
flow := NULL;
flowex := NULL;
for kk to n do
  j := piv[kk];
  gg[j] := map(simplify,map(normal,subs(flow,gg[j])));
  eseq := diff(locx.j(z),z)=gg[j],locx.j(0)=subs(locxinit,x[j]);
  vseq := locx.j(z);
  flowex := traperror(dsolve({eseq},{vseq}));
  if flowex=lasterror then
    # now it turns out that at least one of the equations is not
    # solvable by MAPLE -->
    RETURN(flowex,'MAPLE failed in solving the ordinary differential
    equation',eseq,'although a solution exists.')
  else
    flow := flow,flowex
  fi:
od:

# next kk, separated calculation of elements of flow i
flow := {flow};

# clear garbage
for i to n do
  locx.i:=evaln(locx.i);
  locx0.i:=evaln(locx0.i);
od:

# output
RETURN(flow);
end: # end of the procedure extdsolve
\end{verbatim}
# --> involutive(argseq,x)
# involutive tests whether the sequence of input vectors are vector fields that
# span an involutive distribution.
# a distribution is called involutive if the Lie bracket of any pair of
# vector fields belonging to the distribution belongs to the distribution itself
# i.e. the distribution is closed under the Lie bracket operation.
# input: argseq : set {} of n-dimensional vectors, this set may exist of names
# or vector-structures
# x : unassigned vector containing differential variables
# output: 'true' or 'false'
# involutive calls the procedure liebrack and extrank
# author H.v.E., TUE-WFW, 1992

involutive:=proc(argseq,x)
local i,k,qq,G,loclv,r,n ;
# tests on the input and creation of the matrix G which columns are the vectors
# of argseq
if not nargs=2 then ERROR('wrong type or number of arguments'); fi:
if not type(argseq,'set') then ERROR('invalid 1st argument, should be a
set {} of vectors') else
qq := nops(argseq); if qq=1 then RETURN('true') fi:
fi:
if not type(x,'vector') then ERROR('invalid 2nd argument, should be an
unassigned array containing the differential variables') fi:
for i to qq do
loclv := op(i,argseq):
if not type (loclv,'vector') then loclv := op(op(i,argseq)):
if not type (loclv,'vector') then ERROR('invalid 1st
argument, should be (set of) vectors')
fi:
if i=1 then n:=linalg[vectdim](loclv); G:=copy(loclv);
elif not linalg[vectdim](loclv)=n then ERROR ('invalid 1st argument,\vectors dimensions should be equal') else G:= linalg[augment](G,loclv)
fi:
od: #i)
r := extrank(map(simplify,map(normal,G)))
for i to qq-1 do
for k from i+1 to qq do
loclv := liebrack[linalg[col](G,i),linalg[col](G,k),x];
if extrank[linalg[augment](G,loclv)]>r then RETURN('false') fi:
od: #k)
od: #i)

'true';
end:  # end of procedure involutive
#---> mkmat(r)
#---> mkmat(c)
#
# mkmat creates a (r x 1) or (1 x c) matrix structure of a r dimensional row or a
# c dimensional column array. the purpose of this matrix is to be able to apply
# (linear algebra) matrix computations on the original array
#
# input
#   r : r dimensional array
#   c : c dimensional array
#
# output
#   function value : the corresponding matrix structure
#

mkmat := proc(r)
local n, i, locmat:
    if not type(r, 'vector') then
      ERROR( 'argument must be defined as a vector,' )
    else
      n := linalg[vectdim](r)
    fi:
    locmat := array(1..1,1..n,[]);
    for i to n do
      locmat[1,i] := r[i]
    od;
    op(locmat);
end:

mkmatc := proc(c)
local n, i, locmat:
    if not type(c, 'vector') then
      ERROR( 'argument must be defined as a vector,' )
    else
      n := linalg[vectdim](c)
    fi:
    locmat := array(1..n,1..1,[]);
    for i to n do
      locmat[i,1] := c[i]
    od;
    op(locmat);
end:
# --> mklog()
# mklog creates a log - on the screen- of a full analysis of a system. mklog
# includes reldeg, normform, extnormform, statelin, outputfunc, inoutlin and
# transform. as much information as possible is returned
#
mklog := proc()
# creation of an example log

# used model
print('output function(s): ', h);print('initial states: ', xnull);

# relative degree
print('');
print('***** the relative degree, (reldeg)*****');
print('** possible conditions on the initial states:');
traperror(reldeg(f,g,h,xnull, 'rdeg', 'rtot', 'Adeg', 'bdeg', 'cond')):
if "lasterror then print('Error in reldeg: ', " fi;
print(rtot);
print(rdeg);
print('***** the total sum of the (vector) relative degree and this (vector)\nrelative degree:');
print(rtot);
print(rdeg);
print('Adeg');print(bdeg);

# the normal form
normdyn:= 'normdyn';fn:='fn';gn:='gn';zerodyn:= 'zerodyn';uzero:= 'uzero';
print('');
print('****** the normal form, (normform)*****');
traperror(normform(f,g,h,xnull, 'normdyn', 'fn', 'gn', 'zerodyn', 'uzero'));
if "lasterror then print('Error in normform: ', " fi;
print('***** the normal form system equations: (normdyn)*****';
print(normdyn);
print('***** the normal form system dynamics: (fn and gn)*****');
print(fn);print(gn);
print('***** the zero dynamic equations and the unique zeroing\input: (zerodyn, uzero)*****');
print(zerodyn);print(uzero);

# the zero dynamics from extnormform
zerodyn:='zerodyn';uzero:='uzero';
print('');
print('****** the zero dynamics from extnormform, (extnormform)*****');
traperror(extnormform(f,g,h,xnull, 'zerodyn', 'uzero')):
if "lasterror then print('Error in extnormform: ', " fi;
print('***** the zero dynamics: (zerodyn)*****');
print(zerodyn);
print('***** the zeroing input: (uzero)*****');
print(uzero);

# exact linearization of the state input equations
u:='u';Az:='Az';Bz:='Bz';alfa:='alfa';beta:='beta';
print('');
print('****** exact linearization of the state input equations,\(statelin)*****');
traperror(statelin(f,g,h,xnull,v,u,Az,Bz,alfa,beta)):
if "lasterror then print('Error in statelin: ', " fi;
print('***** the exact linearizing feedback and linearized system dynamics: \(u, Az, Bz)*****';
print(u);print(Az);print(Bz);
print('***** the explicit feedback: (alfa, beta)*****');
print(alfa);print(beta);
Appendix D

Listings of the procedures

# finding output functions which give the system a full order relative degree
lambda:=\texttt{lambda};
print('\');
print('***** finding output functions which give the system a full order\relative degree, (outputfunc)*****');
traperror(outputfunc(f,g,xnull,lambda));
if \\texttt{''} =\texttt{lasterror} \texttt{then} print('Error in outputfunc:',') fi;
if type(lambda,\texttt{array}) \texttt{then} print('the output function(s) which fulfil the\demands are: ') else print('the output function(s) that are computed are: ') fi;
print(lambda);

# exact linearization the input output mapping
u:='u';fl:='fl';gl:='gl';alfa:='alfa';beta:='beta';
print(' ');
print('***** exact linearization the input output mapping, (inoutlin)*****');
inoutlin(f,g,h,xnull,v,u,fl,gl,alfa,beta));
if \\texttt{''} =\texttt{lasterror} \texttt{then} print('Error in inoutlin:',') fi;
print('* the exact linearizing feedback and linearized system dynamics:\
\{u, fl, gl\}');
print(u);print(fl);print(gl);
print('* the explicit feedback (alfa, beta)');
print(alfa);print(beta);

# transformation to the normal form
phi:='phi';phiinv:='phiinv';jacobia:='jacobia';zeta:='zeta';eta:='eta';
print(' ');
print('***** transformation to the normal form, (transform)*****');
transform(f,g,h,xnull,'phi','phiinv','jacobia','zeta','eta');
if \\texttt{''} =\texttt{lasterror} \texttt{then} print('transform: ',') fi;
print('* transformation, inverse transformation and jacobian');
print(phi);print(phiinv);print(jacobia);
end: