Analysis of nonlinear control systems with "Nonlincon"

van de Ven, R.J.M.

Published: 01/01/2000

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 18. Dec. 2018
Analysis of nonlinear control systems with "Nonlincon"

R.J.M.v.d.Ven

WFW 2000.32

Research project report

Supervisor: dr.ir.H.A.v.Essen

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF MECHANICAL ENGINEERING
DYNAMICS AND CONTROL GROUP

Eindhoven, November 2000
Contents

1 Introduction ........................................... 3

2 Theory .................................................. 5
   2.1 Introduction ....................................... 5
   2.2 System definition ................................... 5
      2.2.1 Transforming an affine system into a nonaffine system .... 6
   2.3 Relative degree ..................................... 6
      2.3.1 Relative degree for affine systems ................. 7
      2.3.2 Relative degree for nonaffine systems ............ 7
   2.4 Normal forms ....................................... 8
      2.4.1 Coordinate transformation of the state variables ... 8
      2.4.2 System dynamics in the new coordinates .......... 9
   2.5 Exact linearizations ............................... 9
      2.5.1 Exact input-state linearization .................. 10
      2.5.2 Exact input-output linearization ................. 10
      2.5.3 Zerodynamics .................................. 10

3 MAPLE package: "Nonlincon" .......................... 13
   3.1 Introduction and history ............................ 13
   3.2 The structure of the "Nonlincon"-package ............ 13
Contents

4 Adaptations and improvements to the package 15
4.1 The procedure "sysclass" ............................. 15
4.2 Reldeg ............................................. 16
4.3 Comparison between affine making and dynamic extension ....... 16
4.4 Standard MAPLE procedures ........................ 17

5 Conclusions and recommendations 19
5.1 Conclusions .......................................... 19
5.2 Recommendations ..................................... 19

Bibliography 21

A Basic code of the "Nonlincon"-package 23

B Procedures 25
B.1 Procedure "sysclass" .................................. 25
B.2 Procedure "reldeg" ..................................... 32

C Examples 39
C.1 Example 1 .............................................. 40
C.2 Example 2 .............................................. 42
C.3 Example 3 .............................................. 44
Chapter 1

Introduction

In this project the MAPLE package "Nonlincon" is studied. MAPLE is a symbolic mathematical computational program. The "Nonlincon"-package contains procedures to compute control problems, related to exact linearizations, in a symbolic way. The most basic procedures are used to compute the relative degree, normal forms, zerodynamics, exact input-state linearizations and input-output linearizations. The "Nonlincon"-package is based on an earlier package, called the "Zerodyn"-package, which was developed by H.v.Essen. The procedures in the "Zerodyn"-package were only suitable for the analysis of affine systems. Two years later that package is adapted and extended and for some procedures it has become possible to analyze nonaffine systems (more information about the "Nonlincon"-package can be found in chapter 3 and in [4] and [5]). In this project the "Nonlincon"-package is investigated with the most recent version of MAPLE: MAPLE version 6.

The problem definition of the project:

Is it possible to make the "Nonlincon"-package more user friendly and what is the added value of MAPLE version 6 for the "Nonlincon"-package?

The main goal of the project:

Make the package more user friendly and look for possibilities in MAPLE version 6 which can simplify or replace parts of the computations or programming code.

The way this goal can be reached is by:

- Adapting the output messages so that much relevant information about the system or the error message is given which can be easily interpreted by the user of the "Nonlincon"-package.

- Looking for (new) standard procedures in MAPLE which can simplify or replace the computations or parts of the programming code.
Chapter 1. Introduction

• Looking for strange or wrong constructions in the programming code

This report is organized as follows:

In chapter 2 theory about exact linearizations is reviewed. A good understanding of the theory is necessary because the implementation of the theory has to cover the whole range of the subject: from computation to error messages. In chapter 3 a brief introduction to the "Nonlincon"-package is given. In chapter 4 the main results of this project are given. The conclusions and recommendations for further work on the "Nonlincon"-package are given in chapter 5. In the appendices the structure of the "Nonlincon"-package and the relation between the procedures is visualized. Also the programming code of the new procedure "sysclass" and the adapted procedure "reldeg" is given. The last appendix contains examples with solutions generated by the "Nonlincon"-package. These examples concentrate on the new procedure "sysclass" and the adapted procedure "reldeg".
Chapter 2

Theory

2.1 Introduction

In this chapter a short preview of the underlying theory for nonlinear control systems is given. It is not pretended to give a complete treatment of the theory. Only a quick survey of the theory, which is used in this project, is given. A more elaborate treatment of the theory can be found in e.g. Isidori [1], Nijmeijer and van der Schaft [2].

The concept of exact linearizing methods of control systems is to find a direct relation between the input(s) and output(s). This direct relation is found when the relative degree is defined. When this relation exists and the system is transformed into a normal form then the closed-loop dynamics can be linearized. When the closed-loop dynamics are linearized then the powerful linear control methods can be applied. One of the main drawbacks of exact linearization methods is the strong model dependency of the method. A very slight difference between the model and the real system can degrade the performance of the controller very fast if not enough precautions are taken.

2.2 System definition

Throughout this paper a state space notation is used to describe a control system. The notation of a general control system in state space formulation is given by:

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

(2.1)

Where \(x\) is a n-dimensional state space variable vector, \(u\) is a m-dimensional input vector and \(y\) is a p-dimensional output vector. The vector \(f\) is called the state vector function and \(h\) is called the output vector function. A system which is affine in control
(a system where the input appears only linear in the state vector function) is described by:

\[
\begin{align*}
\dot{x} &= f(x) + g(x)u \\
y &= h(x)
\end{align*}
\] (2.2)

Where \( g \) is the input matrix function.

A system which cannot be described by 2.2 is called a nonaffine system. Most theory aims at a local analysis. In this context the working points \( x = x^0 \) and \( u = u^0 \) are defined which do not necessarily need to be equilibrium points. The analysis is in most cases only applicable around these working points.

### 2.2.1 Transforming an affine system into a nonaffine system

It is very important to make a distinction between a system which is affine in control (in this report this will simply be called an affine system) and a nonaffine system because most of the theory developed for control systems is different for each of these systems. An important part of the theory is yet only applicable to affine systems, e.g. the normal form of a control system if the relative degree is not defined. Therefore it can be of great value if a nonaffine system can be transformed into an affine system so that the theory for affine systems is applicable. Transformation of a nonaffine system into an affine system can be done by adding input(s) to the state:

\[
x = \begin{pmatrix} x \\ u \end{pmatrix}
\]

A new input is then defined:

\[
v = \dot{u}
\] (2.4)

The new input adds an integrator to the system and this has to be handled with care. The extra integrator gives the system a phase lag and this can endanger the stability of the system.

### 2.3 Relative degree

A fundamental property of a system is the relative degree. All succeeding subjects, like normal forms, exact linearizations and zerodynamics, are based on the notion of relative degree. A way to describe the relative degree is: "The relative degree of a control system is the number of times an output has to be differentiated until an input appears explicitly in the output". The relative degree is treated in more detail in the next sections and formal descriptions are given for MIMO affine systems and MIMO nonaffine systems. The relative degree for SISO systems follows out these definitions by setting \( m = p = 1 \).
Chapter 2. Theory

2.3.1 Relative degree for affine systems

The formal description of the relative degree \( r \) for affine systems (MIMO) is given by:

\[
L_{g_j} L_{f_i}^k h_i(x) = 0 \quad \forall \ k < r_i - 1, \ 1 \leq i \leq p, \ 1 \leq j \leq m \text{ and } x \text{ near } x^0 \quad (2.5)
\]

\[
A_{\text{deg}}(x) = \begin{pmatrix}
L_{g_1} L_{f_1}^{r_1-1} h_1(x) & \cdots & L_{g_m} L_{f_1}^{r_1-1} h_1(x) \\
\vdots & \ddots & \vdots \\
L_{g_1} L_{f_p}^{r_p-1} h_p(x) & \cdots & L_{g_m} L_{f_p}^{r_p-1} h_p(x)
\end{pmatrix}
\]

condition: \( \text{rank}(A_{\text{deg}}) = p \) at \( x = x^0 \)

When the condition is satisfied then the (vector) relative degree is: \( r = [r_1, \ldots, r_p] \). The total relative degree is given by:

\[
r_{\text{tot}} = r_1 + \cdots + r_p \quad (2.6)
\]

The relative degree cannot be defined if:

- There exists an output for which holds: \( L_{g_j} L_{f_i}^k h_i(x) = 0 \quad \forall \ k > 0, \ 1 \leq j \leq m \) and \( x \) near \( x^0 \). In this case this output only depends on the initial state.
  The matrix \( A_{\text{deg}} \) cannot be formulated.

- Each output has a well defined relative degree but: \( \text{rank}(A_{\text{deg}}) < p \) at \( x = x^0 \)

2.3.2 Relative degree for nonaffine systems

The formal description for nonaffine systems (MIMO) is comparable to the one given for affine systems. The main difference is due to the fact that for nonaffine systems no input matrix \( g \) exists. The definition of the relative degree for nonaffine systems:

\[
\frac{\partial}{\partial u_{ij}} L_{f_i}^k h_i(x, u) = 0 \quad \forall \ k < r_i, \ 1 \leq i \leq p, \ 1 \leq j \leq m \text{ and } x \text{ near } x^0, \ u \text{ near } u^0 (2.7)
\]

\[
A_{\text{deg}}(x, u) = \begin{pmatrix}
\frac{\partial}{\partial u_1} L_{f_1}^{r_1} h_1(x, u) & \cdots & \frac{\partial}{\partial u_m} L_{f_1}^{r_1} h_1(x, u) \\
\vdots & \ddots & \vdots \\
\frac{\partial}{\partial u_1} L_{f_p}^{r_p} h_p(x, u) & \cdots & \frac{\partial}{\partial u_m} L_{f_p}^{r_p} h_p(x, u)
\end{pmatrix}
\]

condition: \( \text{rank}(A_{\text{deg}}) = p \) at \( x = x^0 \) and \( u = u^0 \)

Also, as in the case of affine systems, the relative degree cannot be defined if:

- There exists an output for which holds: \( \frac{\partial}{\partial u_{ij}} L_{f_i}^k h_i(x, u) = 0 \quad \forall \ k < r_i, \ 1 \leq i \leq p, \ 1 \leq j \leq m \) and \( x \) near \( x^0 \), \( u \) near \( u^0 \)

- Each output has a well defined relative degree but: \( \text{rank}(A_{\text{deg}}) < p \) at \( x = x^0 \) and \( u = u^0 \).
2.4 Normal forms

2.4.1 Coordinate transformation of the state variables

SISO systems

By putting the system into a "normal form" the structure becomes very simple and transparent for the further analysis of exact linearizations and zerodynamics. A normal form can be achieved via a suitable coordinate transformation. This mapping relates the new coordinates, \( \phi \), to the state variables, \( x \). The normal form will first be derived for SISO affine systems.

If the relative degree is well defined at the working point \( x = x^0 \) then the first \( r \) new coordinates are defined as:

\[
\begin{align*}
\phi_1(x) &= h(x) \\
\phi_2(x) &= L_f h(x) \\
&\vdots \\
\phi_r(x) &= L_f^{r-1} h(x)
\end{align*}
\]

When the relative degree is equal to the number of states then the coordinate transformation is already complete. If \( r < n \) then it is possible to find an additional \( n - r \) functions so that a local coordinate transformation in the neighborhood of \( x^0 \) is reached (the transformation is one to one and onto, so the additional coordinates are independent). The total mapping \( \Phi(x) \):

\[
\Phi(x) = \begin{pmatrix}
\phi_1 & \cdots & \phi_r & \phi_{r+1} & \cdots & \phi_n
\end{pmatrix}^T
\]

In the SISO case it is always possible to choose the additional \( \phi_{r+1}, \ldots, \phi_n \) coordinates in such a way that the following property holds:

\[
L_g \phi_i(x) = 0 \quad \forall \, r + 1 \leq i \leq n, \, x \text{ near } x^0
\]

The advantage of this choice is that the input doesn't appear explicitly in these functions. Later on, while treating the zerodynamics of a control system, this advantage becomes clear.

MIMO systems

The coordinate transformation for MIMO affine systems is analogous to the one given for SISO systems with a slight difference. The total mapping is now defined as:

\[
\Phi(x) = \begin{pmatrix}
\phi_1^1 & \cdots & \phi_1^{r_1} & \cdots & \phi_1^m & \cdots & \phi_2^m & \cdots & \phi_{r_{tot}+1} & \cdots & \phi_n
\end{pmatrix}^T
\]
For MIMO systems it is only possible to choose the last $\phi_{r+1}, \ldots, \phi_n$ coordinates such that property 2.10 holds when the distribution: $G = \text{span}\{g_1, \ldots, g_m\}$ is involutive at $x = x^0$.

### 2.4.2 System dynamics in the new coordinates

After the coordinate transformation the system dynamics can be displayed in these new coordinates:

$$z_i = \phi_i(x) \quad \forall 1 \leq i \leq n \quad (2.12)$$

and the system dynamics:

$$\frac{dz_i}{dt} = \frac{\partial \phi_i}{\partial x} \frac{dx}{dt} = L_f^i h(x) = \phi_{i+1} = z_{i+1} \quad \forall 1 \leq i \leq r-1 \quad (2.13)$$

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

$$\frac{dz_j}{dt} = \frac{\partial \phi_j}{\partial x} \frac{dx}{dt} = \frac{\partial \phi_j}{\partial x} (f(x) + g(x)u(t)) = L_f \phi_j(x) + L_g \phi_j(x)u(t) \quad \forall r+1 \leq j \leq n$$

The system dynamics in a more compact formulation:

$$\begin{pmatrix}
\dot{z}_1 \\
\vdots \\
\dot{z}_{r-1} \\
\dot{z}_r \\
\dot{z}_{r+1} \\
\vdots \\
\dot{z}_n
\end{pmatrix} =
\begin{pmatrix}
z_2 \\
\vdots \\
z_r \\
q_{r+1}(z) + p_{r+1}(z,u) \\
\vdots \\
q_n(z) + p_n(z,u)
\end{pmatrix} \quad (2.14)$$

If the relative degree is equal to the number of states then a very simple structure remains:

$$\begin{pmatrix}
\dot{z}_1 \\
\vdots \\
\dot{z}_{r-1} \\
\dot{z}_r
\end{pmatrix} =
\begin{pmatrix}
z_2 \\
\vdots \\
z_r \\
b(z) + a(z)u
\end{pmatrix} \quad (2.15)$$

### 2.5 Exact linearizations

Two types of linearization exist:
Chapter 2. Theory

- Exact input-state linearization: the relative degree of the system is equal to the number of states. The closed-loop system dynamics can be, theoretically, fully linearized by choosing suitable input(s).

- Exact input-output linearization: the relative degree of the system is less than the number of states. The closed-loop system can now only be partially linearized. Another system, with dimension $n - r_{tot}$, remains. For this system the stability has to be investigated. This is done by analyzing the zerodynamics.

2.5.1 Exact input-state linearization

Exact input-state linearization is done by cancelling the nonlinearities in a nonlinear system so that the closed-loop dynamics becomes linear. The closed-loop dynamics for a SISO affine system can be linearized by looking for a suitable input which linearizes the last system equation from formula 2.15:

$$\dot{z} = b(z) + a(z)u$$

The linearizing input is then defined as:

$$u = \frac{v - b(z)}{a(z)}$$  \hspace{1cm} (2.16)

where $v$ is the new input. After setting the input equal to the one described by the above formula, a linear closed-loop system remains of dimension: $n$. Because the remaining system can be treated as an $n$-dimensional integrator, this closed-loop system can always be stabilized by applying linear control techniques.

2.5.2 Exact input-output linearization

For systems with a relative degree which is less than the number of states, a closed-loop system can only be partially linearized by choosing a linearizing input as suggested in formula 2.16. The remaining $n - r_{tot}$ system equations can be treated as a different system. The linearizing input is only of practical importance if the $n - r_{tot}$ system equations define a stable system. Investigation of the stability of this additional system is done by studying the zerodynamics.

2.5.3 Zerodynamics

When input-output linearization is applied to a system then $n - r_{tot}$ system equations (internal dynamics) remain which are not input-output linearized. Will a controller be useful in practice then these equations have to define a stable system. A way to study
the stability of these equations is by looking at the zerodynamics. The zerodynamics are defined to be the internal dynamics of a system when the system output(s) are kept at zero by the input(s). A nonlinear system whose zerodynamics is stable is called a minimum phase system. Especially for the analysis of the tracking dynamics of a system it can be very useful if the input(s) don't appear in these last equations. Then there are no restrictions on the input(s) for the stability of the zerodynamics.
Chapter 3

MAPLE package: "Nonlincon"

3.1 Introduction and history

In 1992 at the Technical University of Eindhoven a MAPLE package is written for the symbolic analysis of nonlinear control systems. The package is mainly aimed at the exact feedback linearizing methods and all subjects closely related to this. The implemented theory at that time was only applicable to affine systems. The viability of the use of symbolic computation in nonlinear control was investigated by B. de Jager [8]. In 1994 the contents of the package was further extended. It has become possible to analyze some nonaffine systems also. Also a menu-structure was made to ease the use of the package. Later on some programming revisions were made. In this chapter some basic properties of the package are explained. For some more detailed description of the package and the procedures is referred to van Essen [4] and Fischer [5].

3.2 The structure of the "Nonlincon"-package

The "Nonlincon"-package consists of two basic packages: nlctools and nlcmain. The nlctools package consists of mathematical functions and some extended MAPLE procedures. The nlcmain package contains all the procedures for the symbolic analysis of nonlinear control systems. In this project most attention is paid to the basic code of the nlcmain package. The basic code consists of the following functions:

- Reldeg: computation of the relative degree of a system
- Normform: computation of a local state transformation to the normal form
- Zerodyna1: "The zerodynamics algorithm" of Isidori [1].
Chapter 3. MAPLE package: "Nonlincon"

- Statelin: computation of a static state feedback and a state transformation to derive (locally) exact linearized input-state equations
- Inoutlin: computation of a static state feedback to derive (locally) exact linearized input-output behaviour

In appendix A a survey of the procedures in the basic code of the "nlcmain"-package is given. Also the relation between the nlcmain- and the nlctools-package is visualized.
Chapter 4

Adaptations and improvements to the package

4.1 The procedure "sysclass"

As already mentioned in chapter 2 it is very important to distinguish between an affine and a nonaffine system because parts of the theory are different for these two classes of systems. A new procedure is written to determine the kind of system. This procedure checks the input vectors and matrices and classifies the system as being affine or non-affine. This output is given to the user. If a system is nonaffine then the reason why the system is nonaffine is given. It is also possible to transform a nonaffine system into an affine system by putting the input(s) in the matrix g instead of the vector f. This is only possible when the input(s) appear only linear in f and not in "sticked" form, this is when at least two linear appearing inputs are multiplied with each other (e.g. \( u_1 u_2 \)). For a system where the input(s) appear in "sticked" form or appear nonlinear in the vector f it is also possible to transform this system into an affine one. This can be done by adding input(s) to the state (as treated in chapter 2). Note that it is not known in advance that transforming a nonaffine system into an affine one will give better results in analyzing the system because in the classifying stage nothing is known yet about the properties of the system which are important for the computation of the linearizing input, like the relative degree or the normal form. The procedure ends by displaying the input vectors f and h, the matrix g and the working point \( x^0 \). If the system is nonaffine then also the working point \( u^0 \) is given. In Appendix B the programming code of the procedure "sysclass" is given.
4.2 Reldeg

The procedure "reldeg" was originally built to compute the relative degree for affine systems. Later this procedure was adapted to be able to compute the relative degree for nonaffine systems also. In the procedure still some programming faults existed and these are removed. The following faults are removed:

- The relative degree for a nonaffine system can be in the range: \(0 - n\) (degree of the system). In the old procedure the for-loop for computing the relative degree of a nonaffine system was only in the range \(0 - (n - 1)\). So for a nonaffine system with a relative degree equal to the number of states the calculated relative degree was wrong.

- If an output doesn't depend on the input explicitely then an error message has to appear.

The procedure "reldeg" is also adapted with regard to output messages. If the relative degree is not defined then an explicit message is given. The possible relative degree, if the working point is chosen such that a relative degree can be defined for that particular output, is displayed as additional information. Furthermore the reasons why a relative degree is not defined are given and then the numbers of the output(s), for which the relative degree is not defined, are given. In appendix C a few examples are given which demonstrate the output of the procedures "sysclass" and "reldeg".

4.3 Comparison between affine making and dynamic extension

In the procedure "sysclass" an option is available for the transformation of a nonaffine system into an affine one. As a consequence it can occur that the relative degree changes. Both in affine making and dynamic extension input(s) are added to the state and new input(s) are defined which are the integration(s) of the old input(s). So these two algorithms seem to be quite similar but they are essentially different. If the relative degree for the system is not defined because one or more input(s) appear always later than other(s) and \(m = p\) then the matrix \(A_{\text{deg}}\) does not have a rank equal to the number of output(s). By adding the early input to the state, the appearance of this input is "delayed" to higher orders of the outputs and it is then possible that the late input shows up while determining the relative degree. It is then possible that the matrix \(A_{\text{deg}}\) has a rank equal to the number of output(s) and the relative degree becomes defined. Thus the dynamic extension algorithm is only applicable if each separate output has a well defined relative degree but \(\text{rank}(A_{\text{deg}})\) is less than the number of outputs because of "late" inputs. By applying the "dynamic extension algorithm" the relative degree for the total system can become defined.
The main motive for transforming a nonaffine system into an affine one is that the theory for affine systems becomes applicable. As a consequence the relative degree can change (it can become defined but also undefined).

The idea of adding an input to the state can also be used to delay an input if the relative degree is already defined. In this case the relative degree increases.

4.4 Standard MAPLE procedures

It is possible to write procedures in MAPLE with the aid of standard procedures of MAPLE. These standard procedures quicken and ease the programming work and the computations. In the new procedure "sysclass" a few new standard procedures are used like the "has"- and "degree"-procedure. Compared to earlier versions of MAPLE some standard procedures have become much more powerfull, but some of the standard procedures still have serious limitations.

As already mentioned in chapter 2, the analysis of nonlinear control systems becomes much easier if a proper coordinate transformation can be found which transforms the system description into a transparent and simple structure. When dealing with a system where the relative degree is less than the number of states, the remaining \( n - r_{\text{tot}} \) coordinates have to be chosen such that the transformation is complete (one to one and onto). For simplicity of the normal form and the zerodynamics these coordinates have to be chosen such that, provided the distribution is involutive, condition 2.10 is satisfied. In the case of MIMO systems this condition represents a set of partial differential equations. In the current MAPLE 6 there is still no standard procedure available to solve these sets of partial differential equations. This is a limitation because the analysis can become unnecessarily complicated.

In the mathematical toolbox of the "Nonlincon"-package (see appendix A) also a lot of extended MAPLE procedures exist: extrank, extgausselim, extgaussjord, extcolspace, extrowspace, and extdsolve. These procedures were written to replace the weak standard procedures in the previous MAPLE versions. In MAPLE 6 the standard procedures have become much more powerfull. It is not investigated if it is possible to replace the extended MAPLE procedures in the "Nonlincon"-package with the standard MAPLE procedures.
Chapter 4. Adaptations and improvements to the package
Chapter 5

Conclusions and recommendations

5.1 Conclusions

- The "Nonlincon"-package is extended with the procedure "sysclass". This procedure classifies each system as being affine or nonaffine. Furthermore it is possible to transform a nonaffine system into an affine one by rearranging inputs or by adding input(s) to the state.

- The procedure "reldeg" is improved, among other things with regard to the relative degree of nonaffine systems.

- Some output messages are elaborated for better interpretability of the results from the computations or the error messages.

- A shortcoming of MAPLE version 6 is the lack of a procedure for solving sets of partial differential equations.

- MAPLE 6 contains procedures which are more powerful than the same procedures in earlier versions of MAPLE.

5.2 Recommendations

- The package can be enlarged by adding new procedures. Especially in the analysis of nonaffine systems some new theory can be applied. Also a new area of analysis tools can be added, e.g. stability analysis.

- Solutions have to be found for mathematical difficulties like finding the solution to a set of partial differential equations.
• The strength of the latest MAPLE versions as well as the strength of the "Nonlincon"-package has to be investigated by using complex examples

• Investigation of the possibility to replace the extended MAPLE procedures by the standard MAPLE 6 procedures

• The package "Nonlincon" can be made more user friendly by:
  – making a help library
  – adapting the menu environment such that only the analysis functions become available which are useful in the analysis of the particular system
  – adapting the output messages, which are difficult to understand, of all procedures
  – avoiding large amounts of output messages on the screen
Bibliography


Appendix A

Basic code of the "Nonlincon"-package

Figure A.1: Basic procedures in the "Nonlincon"-package
Appendix A. Basic code of the "Nonlincon"-package

Figure A.2: A simplified model of the structure of the "Nonlincon"-package
Appendix B

Procedures

B.1 Procedure "sysclass"

The procedure sysclass classifies each system as being affine or nonaffine in control.

The following definitions hold for affine and nonaffine systems:

affine system:

\[ xdot = f(x) \]

\[ y = h(x) \]

nonaffine system:

\[ xdot = f(x,u) \]

\[ y = h(x,u) \]

In some cases it is possible to transform a nonaffine system into an affine one:

1. if \( u \) appears only linear and does not appear in 'sticked' form then an affine
   system description can be obtained by reordering elements in the vector \( f \) and the
   matrix \( g \).

2. by adding input(s) to the state
   
For autonomous systems \( g \) has to be defined as \( g := 0 \)

Output: the vectors \( f \) and \( h \) and the matrix \( g \). The sets \( \text{xnull} \) and \( \text{unull} \). Boolean expression
   nonaffine. These output parameters are all global.

```plaintext
sysclass:=proc(fin,gin,hin,xnullin,unullin)
# The procedure sysclass classifies each system as being affine or nonaffine in control.
# The following definitions hold for affine and nonaffine systems:
# affine system:
# nonaffine system:
# In some cases it is possible to transform a nonaffine system into an affine one:
# 1. if \( u \) appears only linear and does not appear in 'sticked' form then an affine
# system description can be obtained by reordering elements in the vector \( f \) and the
# matrix \( g \).
# 2. by adding input(s) to the state
# For autonomous systems \( g \) has to be defined as \( g := 0 \)
# Output: the vectors \( f \) and \( h \) and the matrix \( g \). The sets \( \text{xnull} \) and \( \text{unull} \). Boolean expression
# nonaffine. These output parameters are all global.

local i,j,t,l,m,k,us,ud1,ud2,us1,us2,uset,count,count1,cl,vs,seq1,seq2,ma,ma1,ma2,
max2,fn,fn1,fg,gg,fg1,gg1,fg2,gg2,fn1,fn2,gn1,gn2,gn3,
all_u,fg_i,cout1,cout2,cout3,coun1,coun2,coun3,coun4,coun5,p,pplus,q,r,set,nona,
fadd,gadd,gad,up1,up2,up3,
opt,xadd,xnull,unull,opz,loef,locg,loch,locnull,locnull1,locnull2,locnull3,locnull4,
print1,print2,print3,print4,print5,print6,print7,print8,print9,print10,print11,
v,n,nacs,
global f,g,h,xnull,unull,nonaffine,x,u;

xnull:=NULL; unull:=NULL; opt:=0; n:=0; m:=0; p:=0; unassign('x','u');

fe:=copy(fin); # fe is equivalent with fin
ge:=copy(gin); # ge is equivalent with gin
he:=copy(hin); # he is equivalent with hin

if not type(fe,vector) then lprint('f has to be of type vector'); fi;
if not type(he,vector) then lprint('h has to be of type vector'); fi;
if (ge<>0 and type(ge,matrix)=false) then lprint('g has to be of type matrix or 0');
error('see last comment'); fi;
```

```plaintext
fe:=copy(fin); # fe is equivalent with fin
ge:=copy(gin); # ge is equivalent with gin
he:=copy(hin); # he is equivalent with hin
```

```plaintext
if not type(fe.vector) then lprint("f has to be of type vector"); error("see last comment"); fi;
if not type(he.vector) then lprint("h has to be of type vector"); error("see last comment"); fi;
if (ge<>0 and type(ge.matrix)=false) then lprint("g has to be of type matrix or 0");
error("see last comment"); fi;
```
Appendix B. Procedures

### DETERMINING DIMENSION (N,M,P) ###

```plaintext
n := linalg[vectdim](fe);
q := linalg[vectdim](fin);
p := linalg[vectdim](he);
if gec > 0 then
  r := linalg[coldim](ge);
m := linalg[coldim](gin);
else
  m := 0;
fi;
```

### TESTING ON AFFINE OR NON AFFINE ###

```plaintext
coun1 := 0; coun2 := 0; coun3 := 0; coun4 := 0; coun5 := 0;
for j to m do
  for i to n do
    if has(fin[i], u[j]) and degree(fin[i], u[j]) <> 1 then coun2 := 1;
    elif has(fin[i], u[j]) then coun1 := 1; fi;
  od;
for i to p do
  for j to m do
    if has(fin[i], u[j]) then coun1 := 1; fi;
  od;
if coun1 = 0 and coun2 = 0 and coun4 = 0 and gec = 0 then
  lprint('The system has no input(s). The system is completely autonomous');
  nacs := true; # nacs = not a control system
  fi;
if gec > 0 then
  for j to m do
    for i to n do
      for l to m do
        if has(gin[i], u[l]) then
          handle';
          lprint('The input matrix g contains input variables. The program cannot handle this input. In this case this nonaffine system has to be written in the form:');
          lprint('xdot = f(x,u), y = h(x,u) and setting g as a sparse array');
          error('See last comment');
        od;
      od;
    od;
  od;
fi;
```

# combining of f(x,u) and g(x)
if (coun2 = 1 or coun1 = 1) and gec > 0 then
  fadd := array(1..n, sparse); gadd := array(1..n, sparse);
  for i to n do
    for j to m do
      gad := gin[i, j]*u[j];
      gadd[i] := gadd[i] + gad;
    od;
    fadd[i] := fe[i] + gadd[i];
  od;
ge := 0; fe := copy(fadd);
fi;
if nacsc=true then
# GENERAL NONAFFINE SYSTEM AND MAKING IT AFFINE OR EXPlicit AFFINE SYSTEM AND PUTTING IT IN THE RIGHT FORM

# DETECTING DEGREE(U[I])<>1
if (count2=1 or count4<>1) and count4<>1 then
us1:=NULL; udl:=NULL;
else if udl<>NULL then
for i to n do
    for j to m do
        if has(fe[i],u[j]) and degree(fe[i],u[j])<>1 then
            udl:=udl,j;
            od;
    od;
usl:=sort({udl});
fi;

# SUBSTITUTION OF U[I] WITH DEGREE<>1 IN F(X,U)
varseq1:=NULL;
if udl<>NULL then
    for i to nops(usl) do
        varseq1:=varseq1,u[usl[i]]*x[n+i];
    od;
    fn1:=array(1..n+nops(usl),sparse);
    for i to n do
        fn1[i]:=normal(subs(varseq1,fe[i]));
    od;
    for i to nops(usl) do
        fn1[n+i]:=u(usl[i]);
    od;
else
    fn1:=array(1..n, sparse);
    for i to n do
        fn1[i]:=normal(subs(varseq1,fe[i]));
    od;
fi;

# ALL U[I] WITH DEGREE<>1 ARE SUBSTITUTED IN F(X,U)
# LOOKING FOR U[I] WITH U[I]*U[0] (= U[I] IN G)
us2:=NULL; ud2:=NULL; all_u:=0; count1:=0; k:=1;
while (count1<>1 and count4<>1) do
    gn2:=array(1..n,1..m,sparse);
    for i to n do
        gn2[j]:=0;
        for j to m do
            if has(fk[i],u[j]) then
                fnk:=simplify(-coeff(fk[i],u[j])*u[j]);
            else
                fnk:=0;
            fi;
            gn2[i,j]:=fnk;
        od;
    fnk:=simplify(fk);
    gn2:=subs(fk,fnk);
    fi;
    fn1:=fn1[1..n,1..m, sparse];
    for i to n do
        fn1[i]:=simplify(fn1[i]);
    od;
    n:=n+1;
    fi;
while (count1<>1 and count4<>1) do
    gn2:=array(1..n,1..m,sparse);
    for i to n do
        gn2[j]:=0;
        for j to m do
            if has(fk[i],u[j]) then
                fnk:=simplify(-coeff(fk[i],u[j])*u[j]);
            else
                fnk:=0;
            fi;
            gn2[i,j]:=fnk;
        od;
    fnk:=simplify(fk);
    gn2:=subs(fk,fnk);
    fi;
    fn1:=fn1[1..n,1..m, sparse];
    for i to n do
        fn1[i]:=simplify(fn1[i]);
    od;
    n:=n+1;
    fi;
while (count1<>1 and count4<>1) do
    gn2:=array(1..n,1..m,sparse);
    for i to n do
        gn2[j]:=0;
        for j to m do
            if has(fk[i],u[j]) then
                fnk:=simplify(-coeff(fk[i],u[j])*u[j]);
            else
                fnk:=0;
            fi;
            gn2[i,j]:=fnk;
        od;
    fnk:=simplify(fk);
    gn2:=subs(fk,fnk);
    fi;
    fn1:=fn1[1..n,1..m, sparse];
    for i to n do
        fn1[i]:=simplify(fn1[i]);
    od;
    n:=n+1;
    fi;
# LOOKING FOR A U[I] IN G
# DETERMINING WHICH U[I] WILL BECOME A STATE VARIABLE

```plaintext
cl:=l; max3:=NULL;
for j to m do
  count[j]:=0;
  for i to n do
    for t to m do
      if (has(gn2[i,t], u[j]) or has(gn2[i,j], u[t])) then
        count[j]:=count[j]+1; coun5:=1; fi;
      od;
    if j=1 then max1:=count[j];
    else max1:=max(count[2], count[1]);
    fi;
    max2:=max(max1, count[j]); max3:=max3, max2;
  od;
if max2<>0 then
  while max3[cl]<>max2 do cl:=cl+1; od;
  us:=cl; ud2:=ud2, us; us2:=sort({ud2});
else us:=0; coun5:=1;
fi;
if us<>0 then
  n:=n+1;
  fn[k+1]:=array(1..n, sparse);
  for i to n-1 do
    fn[k+1][i]:=fn[k][i];
  od;
  fn[k+1][n]:=subs(u[us]=x[n], fn[k][i]);
  od;
  fn[k+1][n]:=u[us];
  k:=k+1;
else all_u:=1;
  uset:=sort(uset union us2);
fi;
opt2:=true;
if coun5<>1 and coun2<>1 and ge=0 and opt2<>true then
  opt:=yes; opt2:=true;
fi;
```

# ALL U'S WHICH ARE SUBSTITUTED (DEGRE(U[I])<>1 AND U[I]*U[J]) ARE STORED IN US
# THESE U'S WILL NOW BE SUBSTITUTED IN F(X,U)

```plaintext
varseq2:=NULL;
if all_u=1 then
  for i to nops(uset) do
    varseq2:=varseq2, u[uset[i]]:=x[i+1];
  od;
  fg:=array(1..q+nops(uset), sparse);
  for i to q do
    fg[i]:=simplify(normal(subs(varseq2, fe[i][i])));
  od;
  for i to nops(uset) do
    fg[q+i]:=u[uset[i]];
  od;
  n:=q+nops(uset);
fi;
```
Appendix B. Procedures

# NEXT FROM F(X,U) THE MATRICES F(X) AND G(X) ARE DETERMINED

fg1:=array(1..n,sparse);
gg1:=array(1..n,1..m,sparse);
for i to n do
  fg_j:=0;
  gg_j:=0;
  for j to m do
    if has(fg[l],u[j]) then
      fg_c:=simplify(coeff(fg[l],u[j])*u[j]);
      gg_c:=coeff(fg[l],u[j]);
    else fg_c:=0; gg_c:=0;
    fi;
    fg_j:=fg_j+fg_c;
    gg_j[j]:=gg_c;
  od;
  fg_i:=fg_j+fg[l];
fgl[l]:=[simplify(fg_i)];
od;

### OUTPUT

# combining of f(x,u) and g(x)
optadd:=0;
if coun1=l and coun2<>1 and coun5<>1 and ginc>0 and coun4<>1 then
  lprint('The given model is, f, g, h');
  printf2:=NULL;
  for i to linalg[vectdim](f) do
    printf2:=printf2,eval(f[i]);
  od;
  printf3:=Vector(l..linalg[vectdim](f),printf3);
  printf2:=printf2,eval(h[i]);
  od;
  printf3:=Vector(l..linalg[vectdim](h),printf3);
  printf(printf1,op(g),printf3);
  lprint('The system is now written in the nonaffine form:');
  lprint('x=fdot=f(x,u)+g(x)*u, y=h(x)');
  lprint('but because the input appears only linear in f(x,u) the');
  lprint('system can be rewritten in an affine form. ');
  while (optadd<>yes and optadd<>no) do
    optadd:=readstat('Write the system in explicit affine form (yes or no)?:');
    od;
    locf:=copy(fadd);
    locg:=array(1..q,1..r,sparse);
    locxnull:=copy(xnullin);
    locunull:=copy(unullin);
  fi;
  locnonaffine:=false;
  if coun1=1 or (coun1=1 or coun5=1) or optadd=0 then
    if (coun2<>1 and coun5<>1) and optadd<>0 then
      lprint('The system is nonaffine because only the output vector contains input(s): h(x,u)');
      fi;
    if coun4=0 and optadd=0 then
      lprint('The system is nonaffine because only the output vector contains input(s): h(x,u)');
      fi;
  # printing of model because of the question about affine making
Appendix B. Procedures

lprint('The given model is, f, g, h');
printf2:=NULL;
for i to linalg[vectdim](f) do
  printf2:=printf2,eval(f[i]);
od;
printf 3 :=lprintf21;
printf 1 :=Vector(l..linalg[vectdim](f), printf 3);
printh2 :=NULL;
for i to linalg[vectdim](h) do
  printh2 :=printh2,eval(h[i]);
od;
printh3 :=[printh21;
printh :=vector(1..linalg[vectdim](h),printh3);
print (printf 1, op(g), printh);
if coun5=0 and coun2=1 then
  lprint('The system is nonaffine because the state vector contains input(s) which');
  lprint('appear nonlinear in the state vector: f(x,u)');
elseif coun5=1 and coun2=0 then
  lprint('The system is nonaffine because the state vector contains inputs which');
  lprint('appear in 'sticked form' (u[i]*u[j]) in the state vector: f(x,u)');
elseif coun5=1 and coun2=1 then
  lprint('The system is nonaffine because the state vector contains inputs which');
  lprint('appear nonlinear in the state vector and inputs appear in 'sticked form');
  lprint('in the state vector: f(x,u)');
fi;
elif coun4=1 and optadd=0 then
  if coun5=0 and coun2=1 then
    lprint('The system is nonaffine because the state vector contains input(s) which');
    lprint('appear nonlinear in the state vector: f(x,u), and the output vector contains');
    lprint('input(s) h(x,u)');
  elseif coun5=1 and coun2=0 then
    lprint('The system is nonaffine because the state vector contains inputs which');
    lprint('appear in 'sticked form' (u[i]*u[j]) in the state vector: f(x,u), and');
    lprint('the output vector function contains input(s) h(x,u)');
  elseif coun5=1 and coun2=1 then
    lprint('The system is nonaffine because the state vector contains inputs which');
    lprint('appear nonlinear in the state vector and inputs appear in 'sticked form');
    lprint('in the state vector: f(x,u), and the output vector contains');
    lprint('input(s) h(x,u)');
  fi;
lprint('The system is nonaffine');
  fi;
locononaffine:=true;
locf:=copy(fadd); locg:=array(l..q,l..r,sparse); loch:=copy(hin); locxnull:=copy(xnullin);
locunull:=copy(unullin);
fi;
if (coun2<>1 or coun5<>1) and coun4<>1 then
  lprint('The nonaffine system can be transformed into an affine system');
  lprint('by adding inputs to the state');
  while (opt<>yes and opt<>no) do
    opt:=readstat('Transform the system into an affine system (yes or no)?:');
  od;
  opt2:=true;
  if opt=yes then locononaffine:=false; fi;
  if opt=no then lprint('The system is nonaffine'); locononaffine:=true; fi;
fi;
if (coun2<>1 or coun5<>1) and opt=yes and coun4<>1 then
  lprint('For transforming the nonaffine system into an affine system some');
Appendix B. Procedures

lprint('additional information is needed.');
lprint('The working point for the nonaffine system is:');print(xnull);
# bepalen nieuwe xnull in nieuwe unullin
xadd:=NULL; up1:=NULL; up2:=NULL;
for i to nops(uset) do
  lprint('For the following new state a working point has to be defined:');
  print(x[q+i]);
  xadd:=xadd,x[q+i]:=readstat('The working point null for this state:');
  up1:=up1,unullin[uset[i]];
od;

xnull:=xnullin union {xadd};
lprint('The original nonaffine system is transformed into an affine system');
lprint('The following inputs are added to state to obtain an affine system:');
for i to nops(uset) do
  print(x[q+i]=u[uset[i]]);
od;
lprint('These inputs are replaced by het new inputs such that:');
for i to nops(uset) do
  print (xdot[q+i]=v[uset[i]]); v:=v;
od;
lprint('In the new system the new inputs will again be named 'u'');
locf:=copy(fgl); locg:=copy(ggl); loch:=copy(he);
locxnull:=copy(xnulln);
elif coun1=1 and coun2=0 and coun5=0 and coun4=0 and ge=0 and optadd=yes then
  lprint('The system is made affine');
  locf:=copy(fgl); locg:=copy(ggl); loch:=copy(he);
  locxnull:=copy(xnullin); locunull:=copy(unullin);
fi;

# properly affine system
if coun2=0 and coun5=0 and coun4=0 and (coun1=1 or ginc>0) and opt<>yes and optadd<>0 then
  lprint('The system is affine');
  locf:=copy(fe); locg:=copy(ge); loch:=copy(he);
  locxnull:=copy(xnullin); locunull:=copy(unullin);
fi;
else
  locf:=fe; locg:=array(l..q,1..1,sparse); loch:=he; loconaffine:=false;
  fi; # nacs<>true
fi:=eval(locf);
g:=eval(locg);
hi:=eval(loch);
if xnullin<>NULL then xnull:=xnullin else xnull:=xnullin; fi;
unull:=unullin;
if (loconaffine=false and opt<>yes) then x:=array(l..n);
else x:=array(l..q); fi;
if nacs=true then u:=0 else u:=array(1..r); fi;
nonaffine:=loconaffine;
end:
Appendix B. Procedures

B.2 Procedure "reldeg"

```markdown
# 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 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. In the nonaffine case g should be
# # defined as a sparse n x m dimensional array.
# # h : output vector function h(x), a representation of the smooth
# # vector field h, defined in MAPLE as a 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: { x1=... , ... , xn=... }
# # u : input variable vector, defined in MAPLE as an unassigned
# # m dimensional array in the nonaffine case. In case of an
# # affine system, u does not have to be defined, but must
# # always be included in the calling sequence.
# # unull : input working point, defined in MAPLE as a m dimensional
# # set: { u1=... , ... , um=... } when dealing with a
# # nonaffine system. In case of an affine system, unull does
# # not have to be defined, but must always be included in the
# # calling sequence.
# # Remark: Only if dealing with a nonaffine system u and unull should be
# # assigned. In that case g should be assigned as a sparse matrix:
# # g := array(sparse,l..n,l..m), i.e. with zero entries.
# # Affine system: dx/dt = f(x) + g(x) u
# # y = h(x)
# # Nonaffine system: dx/dt = f(x,u) (+ g(x)<---sparse)
# # y = h(x,u)
# # # output parameters: (optional in successive order)
# # rdeg : (vector) relative degree
# # rtot : sum of the (vector) relative degree
# # Adeg : decoupling matrix L_f*rdeg L_g h(x)
# # bdeg : array L_f*rdeg h(x)
# # 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.
```
For a nonaffine system the decoupling matrix cannot be computed, instead
the matrix $A_{\text{deg}}$ contains the derivative of $b_{\text{deg}}$ with respect to $u$. For the
same reason, conditions for which the matrix $A_{\text{deg}}$ is singular are not
appropriate.

d$^r$\text{deg} y / d t$^r$\text{deg} = b_{\text{deg}}(x) + A_{\text{deg}}(x) u

It may occur that the relative degree is not well defined at the chosen
working point $x_{\text{null}}$. The returned function value is a boolean that says
whether the relative degree is well-defined or not.

reldeg calls ldiff, extrank, extgaussjord, mkfullrank and several standard
and [linalg] functions
reldeg is called by normform, statelin, inoutlin, dynext and more.
reldeg is suitable for affine and nonaffine systems with or without well
defined relative degree.

# author: H.v.E., TUE-WFW, 1992
# revised by: G.P., TUE-WFW, 1994 for nonaffine systems
reldeg := proc(f,g,h,x,xnull,u,unull,rdeg,rtot,Adeg,bdeg)#,cond
local n,m,p,locrdeg,locrtot,lgh,lfh,setvar,i,j,k,ii,jj,kk,varseq,solseq,
notdefseq,def,hulpvar,lgha,condhlp,condseq,locxnull,\locunull,xwork,rddef,Adegi,value,notdef,notaff,printbd,printbd2,printbd3,printrdeg,printrtot,zinput,zc,cond:
if nargs<5 and nargs>12 then ERROR('reldeg: invalid number of arguments') fi:
locxnull := xnull;
locunull := unull;
if nonaffine=true then xwork := locxnull union locunull else xwork := locxnull fi:
n := linalg[vectdim](f); # number of state variables
m := linalg[coldim](g); # number of inputs
p := linalg[vectdim](h); # number of outputs
if not m>=p then
print('reldeg: number of inputs must be larger than or equal to number of outputs');
ERROR('see last comment');
fi:
locrdeg := array(l..p,[]);
lfh := array(l..p,[]);
lgh := array(l..p,1..m,[]);
varseq := seq[x[i],i=1..n];
if nonaffine=true then varseq := varseq,seq(u[i],i=1..m) ;
hulpvar := n+m else hulpvar := n fi:
# print('system nonaffine? ',nonaffine);

# relative degree
locrtot := 0:
rddef := 'true'; # initialise procedure result/output
notaff:=NULL;
notdef:=NULL;
notdefseq:=NULL;
for i to p do # loop for output function i
notdefseq[i]:=NULL;
def := 'true';
condseq[i]:='NULL';
value:=0;
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condhlp[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
end:

if nonaffine=true then condseq[1] := condseq[1] + 1
fi:
printrtot[condseq[1]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[1]]:=value; reldeg := proc(f,g,h,x,xnull,u,unull,rdeg,rtot,Adeg,bdeg)#,cond
local n,m,p,locrdeg,locrtot,lgh,lfh,setvar,i,j,k,ii,jj,kk,varseq,solseq,
notdefseq,def,hulpvar,lgha,condhlp,condseq,locxnull,\locunull,xwork,rddef,Adegi,value,notdef,notaff,printbd,printbd2,printbd3,printrdeg,printrtot,zinput,zc,cond:
if nargs<5 and nargs>12 then ERROR('reldeg: invalid number of arguments') fi:
locxnull := xnull;
locunull := unull;
if nonaffine=true then xwork := locxnull union locunull else xwork := locxnull fi:
n := linalg[vectdim](f); # number of state variables
m := linalg[coldim](g); # number of inputs
p := linalg[vectdim](h); # number of outputs
if not m>=p then
print('reldeg: number of inputs must be larger than or equal to number of outputs');
ERROR('see last comment');
fi:
locrdeg := array(l..p,[]);
lfh := array(l..p,[]);
lgh := array(l..p,1..m,[]);
varseq := seq[x[i],i=1..n];
if nonaffine=true then varseq := varseq,seq(u[i],i=1..m) ;
hulpvar := n+m else hulpvar := n fi:
# print('system nonaffine? ',nonaffine);

# relative degree
locrtot := 0:
rddef := 'true'; # initialise procedure result/output
notaff:=NULL;
notdef:=NULL;
notdefseq:=NULL;
for i to p do # loop for output function i
notdefseq[i]:=NULL;
def := 'true';
condseq[i]:='NULL';
value:=0;
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
else
condseq[i] := condseq[i] Union condhlp[condseq[i]]
notdefseq[i] := notdefseq[i] Union notdef;
notaff := notaff Union notaff;
value := condseq[i] + 1
if nonaffine=true then value:=n+1; fi;
printrtot[condseq[i]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[i]]:=value;
end:

if nonaffine=true then condseq[1] := condseq[1] + 1
fi:
printrtot[condseq[1]]:=value;
value:=0;
if nonaffine=true then value:=n+1; fi;
printrdeg[condseq[1]]:=value;
Appendix B. Procedures

for k to value while setvar=0 do
  if nonaffine=true then
    locrdeg[i]:=k-1; # 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(diff(lfh[i],u[j]))):
      od:
    else # affine system
      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:
  fi:
  if % = lasterror then
    lprint('reldeg: WARNING current working point can not be substituted in the');
    lprint('computations are proceeded until a fatal ERROR will occur');
  elif % =0 then
    fi: # %=lasterror
    fi: # lgh[i,j]<0 then setvar:=1;
    # perform substitution of xnull in this entry of lgh
    traperror(simplify(normal(eval(subs(xwork,lgh[i,j])))));
    if % = lasterror then
      lprint('re1deg: WARNING current working point can not be substituted in the');
      lprint('computations are proceeded until a fatal ERROR will occur');
    elif %<>0 then
      # j-th element of row i is zero at the working point
      # xnull but is otherwise not identically zero. Only if
      # another element of this row is nonzero at xnull, the
      # procedure can be continued. In other cases the
      # relative degree is not well defined at the chosen
      # working point xnull.
      def := 'false';notdefseq:=notdefseq,j;
    else #(%<>0)
      # at least one element of row i has a nonzero entry at
      # the point x=xnull, the procedure will be continued
      def:='true'; break #j
  fi:
  fi: # %=lasterror
fi: # lgh[i,j]<<0 then setvar:=1;
od: # j

# if the relative degree is not well defined for this i-th output, then
# MAPLE tries to find initial conditions on the state for which this occurs
if not def then notdef:=notdef,i;
notdefseq := [notdefseq];
for kk to nops(notdefseq) do
  solseq:=[traperror(solve(lgh[i,notdefseq[kk]],[varseq]))];
  if solseq = lasterror then
    print(condseq[i]);
    print(solseq);
    lprint('reldeg: WARNING current working point can not be substituted in the');
    lprint('states for which the relative degree of this output is not');
    lprint('well defined. !!!!
  else
    # selection of nontrivial solution, may be improved !!!!
    for jj to nops(solseq) do condhlp:=NULL;
    for ii to hulpvar do
      if not (op[1,op[jj,solseq][ii]]=op[2,op[jj,solseq][ii]]) then

Appendix B. Procedures

condhlp:=condhlp,op(jj,solseq)[ii];
fi:
   od: # ii
   if condhlp=NULL then condhlp:='all' fi:
   condseq[i]:=condseq[i],condhlp;
od: # jj

condseq[i] := {condseq[i]}; #print(condseq);
fi: # (not def)

if k=value and setvar=0 then
   rdef := 'false';
   notaaff := notaaff,1;
fi:
od: # (k, count variable of the relative degree of the i-th output)

locrtot := locrtot + locrdeg[i];
od: # next output function i

# output
printrdeg:=NULL;
printlht:=NULL;

if nargs>7 then rdeg:=op(locrdeg) fi:
if nargs>8 then rtot:=locrtot fi:
if nargs>9 then Adeg:=map(simplify,map(normal,op(lgh))) fi:
if nargs>10 then bdeg:=array(1..p,[]); # tijdelijke verandering nargs>1 i.p.v.nargs>10 i.v.m. andere procedures
   for i to p do bdeg[i]:=b[i];
   for j to rdeg[i] do
      bdeg[i]:=ldiff(f,bdeg[i],x);
   od:
   od:
fi:

if nargs>11 and nonaffine=true then
   if m>p then
      # augmentation of lgh with unit row vectors to an m x m matrix lghaug
      # of rank m
      mkfullrank(lgh,'lghaug');
      if not extrank(lghaug)=m then
         ERROR('re1deg: mkfullrank makes mistake') fi:
      else # (m=p)
      lghaug:=copy(lgh)
   fi:
   # conditions
   solseq := [traperror(solve(linalg[det](lghaug),[varseq]))];
   if solseq=lasterror then print(solseq):
   elif op(solseq)=NULL then
      lprint('re1deg: no initial states are found for which the matrix Adeg');
   endif
fi;
lprint('turns out to be singular');
else
k:=(solseq); condseq := NULL;
for i to helpvar do
if not (op(1, op(j, solseq)[i]) = op(2, op(j, solseq)[i])) then
condhlp := condhlp, op(j, solseq)[i];
fi:
od:
if condhlp=NULL then condhlp:="all" fi:
condseq:=condseq, (condhlp);
fi;
#(solseq)
fi; #(nargs>ll)

### lg=Adeg
ADEGI:=array(1..p, 1..m, sparse);
for i to p do
for j to m do
ADEGI[i, j] := subs(xwork, lgh[i, j]);
od;
for i to linalg[vectdim](bdeg) do
printbd2 := printbd2, eval(bdeg[i]);
printbd3 := [printbd2];
printbd := Vector(1..linalg[vectdim](bdeg)).printbd3;
lprint('matrix Adeg and vector bdeg:'); print([A[deg]=eval(Adeg), b[deg]=eval(printbd)]);
lprint('');
if nops({notaff})=0 then
rdef := 'false';
lprint('rdef: The relative degree cannot be defined for this system, the output(s)');
lprint('below are not affected by the input but are only depending on');
lprint('the initial state. ');
lprint('rdef: The rank of the matrix Adeg is less than the number of output(s)');
lprint('not_affected_outputs=eval(notaff));
lprint('');
for i to p do
if has({notaff}, i) then rdeg[i]:=nd; fi;
od;
ntot:=nd; printrdeg:=not_defined; printntot:=not_defined;
fi:
if nops({notdef})>0 then
rdef := 'false';
lprint('rdef: The relative degree is not defined at the working point for the output(s)');
lprint('listed between {} (note that this may be an incomplete list of such conditions)');
Appendix B. Procedures

lprint('reldeq: The rank of the matrix Adeg in the working point is less than the number of outputs.');
for i to nops({notdef}) do
  print(outputnr={notdef}[i],eval(condseq({notdef}[i])));
od;
lprint('');
printrdeg:=not_defined; printrtot:=not_defined;
fi;

if linalg[rank](lgh)<p and nops({notdef})=0 and nops({notaff})=0 then
  rddef:='false';
lprint('reldeq: Although each output has a well defined relative degree the rank of the matrix Adeg is less than the number of output(s) due to the fact that the following input(s) appear always later than other(s) ');
  zinput:=NULL;
  for i to m do
    zc[i]:=NULL;
    for j to p do
      zc[i]:={zc[i],lgh[j,i]};
      od;
    if {zc[i]}={o} then zinput:=zinput,i, fi;
    od;
lprint('late-inputs=zinput);

## hier de i'de kolom dan printen
lprint('');
printrdeg:=not_defined; printrtot:=not_defined;
elif linalg[rank](lgh)=n and locrtot=n and printrtot=NULL then
  lprint('The total relative degree at the working point is equal to the number of states.');
lprint('');
fi;

if nops({notdef})<>0 or nops({notaff})<>0 then
  lprint('The number(s) between () give the relative degree of each separate output if the working point would be chosen such that that row has a nonzero entry (nd=not defined, entries are zero for all working points);');
  fi;
if linalg[rank](lgh)<p and nops({notdef})=0 and nops({notaff})<>0 then
  lprint('The number(s) between () give the relative degree of each separate output');
  fi;
lprint('');
printrdeg:=eval(printrdeg),eval(rdeg));
lprint('total relative degree = ');print(eval(printrtot),eval(rtot)));
rddef; # Boolean function output
end: # end of the procedure reldeg
Appendix C

Examples

In this appendix some examples are presented. The results are produced with the "Nonlincon"-package. The examples concentrate on the "sysclass"- and "reldeg"- procedures. The presented output is not the exact output of "Nonlincon". Here only the relevant pieces of the output are displayed. Comment is added between "*".

The following examples are treated:

1. A model is written in the nonaffine form but because the inputs appear only linear in the vector $f$ and not in "sticked" form, the system can be rewritten in the affine form (this example is taken from Isidori 2nd edition, p 257). The relative degree of the system described in the nonaffine form has to be the same as the relative degree of the system in affine form.

2. A model where an input appears nonlinear in the state vector $f$. This example demonstrates the principle of adding input(s) to the state. In this example the relative degree is defined (for a proper choice of the new working point) if a nonlinear appearing input is added to the state. If the model is maintained nonaffine then the relative degree is not defined.

3. A simple affine model (this example is taken from Khalil p 525). For this model the relative degree and the input-output linearization are computed.
Appendix C. Examples

C.1 Example 1

The given model is, \( f, g, h \):

\[
\begin{bmatrix}
    x_2 + x_2^2 + u_2 \\
    x_3 - x_1 x_4 + x_4 x_3 \\
    x_2 x_4 + x_1 x_4 - x_3^2 + \cos(x_1 - x_4) u_4 + u_2 \\
    x_5 \\
    x_2^2 + u_2
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    0 \\
    0 \\
    0
\end{bmatrix}
\begin{bmatrix}
    x_1 - x_4 \\
    x_4
\end{bmatrix}
\]

The system is now written in the nonaffine form:

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

but because the input appears only linear in \( f(x,u) \) the system can be rewritten in an affine form.

Write the system in explicit affine form (yes or no)? yes;

The system is made affine.

The model used is, \( f, g, h \):

\[
\begin{bmatrix}
    x_2 + x_2^2 \\
    x_3 - x_1 x_4 + x_4 x_3 \\
    x_2 x_4 + x_1 x_4 - x_3^2 + \cos(x_1 - x_4) u_4 + u_2 \\
    x_5 \\
    x_2^2
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    \cos(x_1 - x_4) \\
    0 \\
    0
\end{bmatrix}
\begin{bmatrix}
    x_1 - x_4 \\
    x_4
\end{bmatrix}
\]

The working point \( x_{null} \) is:

\( x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0 \)

> ***Computation of the relative degree***

The model used is, \( f, g, h \):

\[
\begin{bmatrix}
    x_2 + x_2^2 \\
    x_3 - x_1 x_4 + x_4 x_3 \\
    x_2 x_4 + x_1 x_4 - x_3^2 + \cos(x_1 - x_4) u_4 + u_2 \\
    x_5 \\
    x_2^2
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    \cos(x_1 - x_4) \\
    0 \\
    0
\end{bmatrix}
\begin{bmatrix}
    x_1 - x_4 \\
    x_4
\end{bmatrix}
\]

The working point \( x_{null} \) is:

\( x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0 \)

Matrix \( A_{deg} \) and vector \( b_{deg} \):

\[
A_{deg} = \begin{bmatrix}
    \cos(x_1 - x_4) \\
    0
\end{bmatrix},
\quad b_{deg} = \begin{bmatrix}
    0 \\
    x_2^2
\end{bmatrix}
\]

Vector relative degree = [3, 2]

Total relative degree = [5]
Appendix C. Examples

> ***Reloading model***

The given model is, \( f, g, h \):

\[
\begin{bmatrix}
    x_1 + x_2^2 + u_2 \\
    x_3 - x_1 x_2 + x_4 x_5 \\
    x_2 x_4 + x_1 x_5 - x_2^2 + \cos(x_1 - x_5) u_1 + u_2 \\
    x_5 \\
    x_2^2 + u_2
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    0 \\
    0 \\
    0
\end{bmatrix}
\begin{bmatrix}
    x_1 - x_5 \\
    x_4 \\
    x_1 - x_3 \\
    x_4 \\
    x_3 - x_2^2 + u_2
\end{bmatrix}
\]

The system is now written in the nonaffine form:

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

but because the input appears only linear in \( f(x, u) \) the system can be rewritten in an affine form.

Write the system in explicit affine form (yes or no)?: no;

The model used is, \( f, g, h \):

\[
\begin{bmatrix}
    x_1 + x_2^2 + u_2 \\
    x_3 - x_1 x_2 + x_4 x_5 \\
    x_2 x_4 + x_1 x_5 - x_2^2 + \cos(x_1 - x_5) u_1 + u_2 \\
    x_5 \\
    x_2^2 + u_2
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    0 \\
    0 \\
    0
\end{bmatrix}
\begin{bmatrix}
    x_1 - x_5 \\
    x_4 \\
    x_1 - x_3 \\
    x_4 \\
    x_3 - x_2^2 + u_2
\end{bmatrix}
\]

the working point \( x_{null} \) is:

\( x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0 \)

the working point \( u_{null} \) is:

\( u_1 = 0, u_2 = 0 \)

> ***Computation of the relative degree***

Make your choice (close with ;) > 5:

The model used is, \( f, g, h \):

\[
\begin{bmatrix}
    x_2 + x_2^2 + u_2 \\
    x_3 - x_1 x_2 + x_4 x_5 \\
    x_2 x_4 + x_1 x_5 - x_2^2 + \cos(x_1 - x_5) u_1 + u_2 \\
    x_5 \\
    x_2^2 + u_2
\end{bmatrix}
\begin{bmatrix}
    0 \\
    0 \\
    0 \\
    0 \\
    0
\end{bmatrix}
\begin{bmatrix}
    x_1 - x_5 \\
    x_4 \\
    x_1 - x_3 \\
    x_4 \\
    x_3 - x_2^2 + u_2
\end{bmatrix}
\]

the working point \( x_{null} \) is:

\( x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 0 \)

the working point \( u_{null} \) is:

\( u_1 = 0, u_2 = 0 \)

matrix \( A_{deg} \) and vector \( b_{deg} \):

\[
A_{deg} = \begin{bmatrix}
\cos(x_1 - x_5) & 1 \\
0 & 1
\end{bmatrix}, \quad b_{deg} = \begin{bmatrix}
\cos(x_1 - x_3) u_1 + u_2 \\
x_2^2 + u_2
\end{bmatrix}
\]

vector relative degree = [3, 2]

total relative degree = 5
Appendix C. Examples

C.2 Example 2

The given model is, \( f, g, h \):
\[
\begin{bmatrix}
-x_5 + u_1^2 \\
x_1 x_1 + u_1 \\
x_2 + u_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

The system is nonaffine because the state vector contains input(s) which appear nonlinear in the state vector: \( f(x,u) \).
The nonaffine system can be transformed into an affine system by adding inputs to the state.

**Transform the system into an affine system (yes or no)?** No.

The system is nonaffine.
The model used is, \( f, g, h \):
\[
\begin{bmatrix}
-x_5 + u_1^2 \\
x_1 x_1 + u_1 \\
x_2 + u_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

The working point \( x_{null} \) is:
\[
x_1 = x_3, x_2 = x_1, x_2 = x_2
\]

The working point \( u_{null} \) is:
\[
u_1 = u_1, u_2 = u_2
\]

***Computation of the relative degree***
The model used is, \( f, g, h \):
\[
\begin{bmatrix}
-x_5 + u_1^2 \\
x_1 x_1 + u_1 \\
x_2 + u_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

The working point \( x_{null} \) is:
\[
x_1 = x_3, x_1 = x_1, x_2 = x_2
\]

The working point \( u_{null} \) is:
\[
u_1 = u_1, u_2 = u_2
\]

**matrix Adeg and vector bdeg:**
\[
A_{deg} = \begin{bmatrix} 2 & 1 \\ u_1 & 0 \end{bmatrix},
\quad b_{deg} = \begin{bmatrix} -x_2 + u_1^2 \\ x_1 x_1 + u_1 \end{bmatrix}
\]

The number(s) between [] give the relative degree of each separate output.

reldeg: Although each output has a well defined relative degree the rank of the matrix Adeg is less than the number of output(s) due to the fact that the following input(s) appear always later than other(s).

late_inputs = 2

The number(s) between [] give the relative degree of each separate output.

vector relative degree =
\[
not_defined, [1,1]
\]

total relative degree =
\[
not_defined, [2]
\]
Appendix C. Examples

> **Reloading model**

'The given model is, \( f, g, h \)

\[
\begin{bmatrix}
-x_2 + u_1^2 \\
x_3 x_1 + u_1 \\
x_3 + u_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

'The system is nonaffine because the state vector contains input(s) which
appear nonlinear in the state vector: \( f(x,u) \)
'The nonaffine system can be transformed into an affine system
by adding inputs to the state'

**Transform the system into an affine system (yes or no)?:** yes;

'For transforming the nonaffine system into an affine system some
additional information is needed.'
'The working point for the nonaffine system is:'
\( \{x_1 = x_5, x_1 = x_1, x_2 = x_2\} \)

'For the following new state a working point has to be defined:'
\( x_4 \)

**The working point xnull for this state:** \( x[4] \);

'The original nonaffine system is transformed into an affine system'
'The following inputs are added to state to obtain an affine system:'
\( x_4 = u_1 \)
'These inputs are replaced by hat new inputs such that:'
\( x_4d_{x_4} = v_1 \)

'In the new system the new inputs will again be named 'u' '
'The model used is, \( f, g, h \)

\[
\begin{bmatrix}
-x_2 + x_4^2 \\
x_3 x_1 + x_4 \\
x_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

'the working point xnull is'
\( x_4 = x_4, x_5 = x_5, x_1 = x_5, x_2 = x_2 \)

> **Computation of the relative degree**

'The model used is, \( f, g, h \)

\[
\begin{bmatrix}
-x_2 + x_4^2 \\
x_3 x_1 + x_4 \\
x_2
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

'the working point xnull is'
\( x_4 = x_4, x_5 = x_5, x_1 = x_5, x_2 = x_2 \)

'matrix Adeg and vector bdeg:'
\[
A_{\text{deg}} = \begin{bmatrix} 2x_4 & 0 \\ 1 & x_{1,d} \end{bmatrix}, \quad b_{\text{deg}} = \begin{bmatrix} -x_3 x_1 - x_4 \\ -x_3 x_2 + x_3 x_4 + x_1 x_2 \end{bmatrix}
\]

'vector relative degree = '
\([2, 2]\)
'total relative degree = '
\([4]\)
The system is affine
The model used is, f, g, h
\[
\begin{bmatrix}
\sin(x_2)
\end{bmatrix}
\begin{bmatrix}
x_2
\end{bmatrix}
\begin{bmatrix}
0
\end{bmatrix}
\begin{bmatrix}
x_1
\end{bmatrix}
\begin{bmatrix}
-x_1^2
\end{bmatrix}
\begin{bmatrix}
1
\end{bmatrix}
\begin{bmatrix}
x_2
\end{bmatrix}
\]

The working point xnull is
\[x_1 = 0, x_2 = 0\]

> *** Computation of the relative degree***
The model used is, f, g, h
\[
\begin{bmatrix}
\sin(x_2)
\end{bmatrix}
\begin{bmatrix}
0
\end{bmatrix}
\begin{bmatrix}
x_2
\end{bmatrix}
\begin{bmatrix}
-x_1^2
\end{bmatrix}
\begin{bmatrix}
1
\end{bmatrix}
\begin{bmatrix}
x_2
\end{bmatrix}
\]
The working point xnull is
\[x_1 = 0, x_2 = 0\]

Matrix Adeg and vector bdeg:
\[A_{deg} = \begin{bmatrix} 1 \end{bmatrix}, b_{deg} = \begin{bmatrix} -x_1^2 \end{bmatrix}\]

Vector relative degree = 1
Total relative degree = 1

> *** Input-Output linearization ***
The model used is, f, g, h
\[
\begin{bmatrix}
\sin(x_2)
\end{bmatrix}
\begin{bmatrix}
0
\end{bmatrix}
\begin{bmatrix}
x_2
\end{bmatrix}
\begin{bmatrix}
-x_1^2
\end{bmatrix}
\begin{bmatrix}
1
\end{bmatrix}
\begin{bmatrix}
x_2
\end{bmatrix}
\]
The working point xnull is
\[x_1 = 0, x_2 = 0\]

Io linearizing feedback and linearized system dynamics (ulin, fl, gl)
\[v_1 = x_1^2\]
\[\begin{bmatrix} a \sin(x_2), 0 \end{bmatrix}
\begin{bmatrix}
0
\end{bmatrix}
\begin{bmatrix}
1
\end{bmatrix}\]
The explicit feedback (alpha, beta)
\[\begin{bmatrix} x_1^2 \end{bmatrix}
\begin{bmatrix} 1 \end{bmatrix}\]