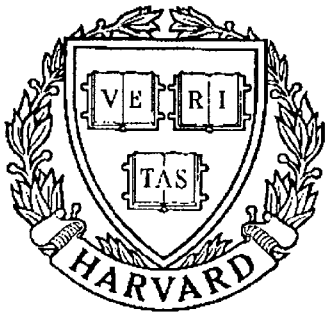
THESIS REPORT
Master's Degree

S Y S T E M S
R E S E A R C H
C E N T E R



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Using Computer Algebra for Design of Nonlinear Control Systems

by O. Akhrif
Advisor: G.L. Blankenship

M.S. 87-2
Formerly TR 87-50

Using Computer Algebra for Design of Nonlinear Control Systems

by

O. Akhrif

G. L. Blankenship

USING COMPUTER ALGEBRA FOR DESIGN OF NONLINEAR CONTROL SYSTEMS

O. Akhrif * G. L. Blankenship **

Electrical Engineering Department & Systems Research Center
University of Maryland, College Park, Maryland 20742

*This work was presented as a M.S Thesis by O. Akhrif.

**This research supported in part by NSF Grant CDR-85-00108

Abstract

A rich collection of analytical tools based on differential geometric methods has been developed for the analysis and design of nonlinear control systems. The concept of feedback equivalence among nonlinear systems is used to linearize and control certain classes of nonlinar control systems. The left and right invertibility of nonlinear systems is used to solve the output tracking problem. Using computer algebra programming methods, a software system has been developed which makes these analytical procedures available to users who need not have an extensive knowledge of differential geometry. Examples of the use of this system are reported.

Acknowledgements

We would like to thank Professor C. I. Byrnes of Arizona State University who suggested this area of research.

O. Akhrif would also like to express her appreciation of the financial support in the form of a research fellowship from the Systems Research Center, University of Maryland.

Contents

Chapter 1

Introduction

The main object of the thesis is the creation of a general-purpose tool for analysis and design of nonlinear control systems in the form of a software system using computer algebra and symbolic manipulations.

The first part of the thesis will be concerned with the analysis of deterministic servo-problems for nonlinear systems affine in control, i.e, systems which in local coordinates are described by:

$$\frac{dx}{dt} = f(x) + \sum_{i=1}^{m} u_i(t) g_i(x) \tag{1.1}$$

In recent years, the differential geometric approach to nonlinear control problems has been developing fast. The differential geometric setting allows the generalization of many known classical results in linear systems theory to the nonlinear case. For the purpose of dynamic control of nonlinear affine systems (output tracking, stabilization ...), we will use two concepts from differential geometric system theory. The first is the concept of *feedback equivalence* among nonlinear systems. The second is the concept of *left-invertibility* of nonlinear control systems.

Feedback equivalence is an equivalence relation (transitive, symmetric and reflexive) among systems and it generalizes the concept of linear feedback group which plays a role in linear system theory (*Wonham* [23]) leading, among other things, to the Brunovsky canonical form and the definition of controllability indices.

R. Brockett [2] introduced a first definition of feedback transformation which

includes state space change of coordinates, additive state feedback and control space change of coordinates linear over the reals. *Jacubczyk-Respondek* [13] and *Hunt-Su* [10], [11] generalized *Brockett*'s definition allowing control space change of coordinates, linear over the ring of smooth functions: $u = a(x) + b(x)v$.

For nonlinear systems, the appeal of feedback-equivalence concept relies on the same motivation: One can study equivalence classes of systems containing representations with special structures or canonical forms of interest for analysis or control purposes i.e, we look for structures that can be destroyed or induced by feedback transformations.

It has been known for many years that most of the relevant properties of system (1.1), in the analytic case, are precisely mirrored in the Lie configuration of L : the Lie algebra generated by $f, g_1 \dots g_m$.

A first and very important step in this direction has been accomplished by *Brockett* [2] (at an earlier stage), *Jacubczyk-Respondek* [13] and *Hunt-Su* [10], [11], who give necessary and sufficient conditions on L in order for system (1.1) to be feedback equivalent to a linear controllable system, which is certainly the best understood class of systems.

The second concept used is the concept of *invertibility* of nonlinear systems. A control system is invertible when the corresponding input-output map is injective. If this is the case, then it is possible to reconstruct uniquely the input acting on the system from the knowledge of the corresponding output. The main application of this is in the output tracking problem where we try to control a system so that its output follows some desired path. The inverse system is used to generate the required control given the desired output function.

For linear systems, this problem was first considered by *Brockett* and *Mesarovic* in 1965 [3]. Then *Silverman* [20] proposed an algorithm in 1969 for multivariable linear systems which does the following:

Given the LTI multi-input multi-output system:

$$\frac{dx}{dt} = Ax + Bu$$

$$y = Cx + Du$$

The algorithm generates an inverse system:

$$\begin{aligned} \frac{dz}{dt} &= \hat{A}z + \hat{B}\hat{y} \\ u &= \hat{C}z + \hat{D}\hat{y} \end{aligned}$$

where

$\hat{y}$ is composed of components $y(t), y^{(1)}(t), \ldots, y^{(\alpha)}(t)$.

α is the relative order of the system.

The generalization of these ideas to nonlinear control systems has been accomplished by *Hirschorn* [7], [8], [9] who gave necessary and sufficient conditions for the invertibility of nonlinear control systems of the form:

$$\begin{aligned} \dot{x} &= f(x) + ug(x) \\ y &= h(x) \end{aligned}$$

where the state space is a real analytic manifold. For invertible systems, nonlinear inverse systems can be constructed and the class of real analytic functions which can appear as outputs of a given nonlinear system can be described.

In the second part of the thesis, we present CONDENS, a software package that addresses methods for nonlinear control systems using a differential geometric approach.

Equipped with the theoretical concepts presented in previous chapters, this expert system can, given a nonlinear system affine in control, answer questions such as: Is this system feedback-equivalent to a controllable linear one? If so, can we construct the diffeomorphism that makes this equivalence explicit? Is the non linear system invertible? What is the relative order of this system? In case the system is invertible, can we construct a left-inverse for our system? Given a real analytic function $y(t)$, can $y(t)$ appear as output for our original nonlinear system? If so, what is the required control?

The expert system tries to find an answer to all these questions and finally,

uses this knowledge to solve a design problem: The output tracking problem for the given nonlinear control system.

The ultimate purpose of the system is to assist the user in the design process by automatically selecting and executing an appropriate design method. The final step is to validate and implement the design. This feature shifts a large amount of complexity of the design problem from the design engineer to the "knowledge base" of the CAE system.

The system presents an other feature: it can automatically generate numerical programs for the solution of problems posed in symbolic form or for simulation purposes. Given a new control problem, the time involved in analyzing the theoretical results and then writing the Fortran code to execute it is eliminated. The mistakes and the time required to test and debug the Fortran code is also eliminated.

Most importantly, the system allows the engineer to interact with the computer for design at the symbolic manipulation level. In this way, he can modify his analysis or design problem by modifying the symbolic functional form of the model. The Fortran subroutines that he might have to modify by hand to accomplish this in conventional design procedures are written automatically for him.

The outline of the thesis is as follows:

In Chapter 2, we introduce the basic definitions and tools from differential geometry used throughout the thesis.

In Chapter 3, the concepts of state equivalence and feedback equivalence among systems are introduced and necessary and sufficient conditions for state and feedback equivalence to linear controllable systems are recalled from the literature. The construction of state and feedback transformations is also reviewed.

In Chapter 4, necessary and sufficient conditions for the invertibility of non linear control systems of the form $\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i, y = h(x)$ are given. Also, the relative order of the system is introduced. These results are used to study the question of functional controllability for nonlinear systems where the problem is to determine functions $f(t)$ which can be realized as the outputs of the nonlinear system driven by a suitable input function. A prefilter or left-inverse is

then constructed to generate the required control.

In Chapter 5, we show how we can use the differential geometric concepts presented for the output tracking of nonlinear control systems. Two design schemes are used. The first one was proposed by *G. Meyer* at NASA Ames Research Center [17]. He used the transformation of control systems to Brunovsky canonical form in the design of model following automatic pilots for vertical and short take-off aircraft. One important aspect of this design is that all regulation is done on the canonical form and the regulator never sees the more complicated original system.

The second design scheme is based on finding the inverse of the nonlinear system which, if given the desired output will naturally generate the required control. Ideally, the Input/Output relationship of the cascade consisting of the nonlinear system and its inverse is an identity. However, disturbances dictate that a regulator be synthesized.

In Chapter 6, we present all the programming work. We show how the expert system can make available to the design engineer some design methods that rely on more or less sophisticated mathematical tools. A complete description of the different modules of the program is given. In the last section, we present some examples with simulation results to illustrate the performance of the system.

Chapter 7 summarizes the contributions of this work and gives a brief amount of directions of further research.

Chapter 2

Preliminaries

In this chapter, background material and terminology which will be freely employed throughout the thesis are introduced. Basic references are [1], [12].

M : a paracompact connected C^∞ or C^w manifold of dimension n (Throughout the thesis M will be R^n).

V(M) : is either $V^\infty(M)$ or $V^w(M)$ the set of all C^∞ or C^w vector fields defined on M. $V(M)$ can be given the structure of vector space over the field of the reals and the structure of Lie algebra with Jacobi brackets (see below) as a nonassociative multiplication with the usual properties:

- $[f,g] = -[g,f]$ anti-commutativity
- $[f,[g,h]] + [h,[f,g]] + [g,[h,f]] = 0$ Jacobi identity
- $[f, ag + bh] = a[f,g] + b[f,h]$

  $[af + bg, h] = a[f,h] + b[g,h]$

  where $f,g,h \in V(M)$; $a,b \in R$

F(M) : is either $C^\infty(M)$ or C^w, set of all C^∞ or C^w real valued functions $\phi : M \longrightarrow R$. It can be given the structure of a ring. $V(M)$ can also be given the structure of a module over the ring $F(M)$.

Jacobi Brackets : $\forall f, g \in V(M) \quad \forall x \in M$

$$[f,g](x) = ad_f g(x) = L_f g(x)$$

In local coordinates:

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

where $\frac{\partial g}{\partial x}$ and $\frac{\partial f}{\partial x}$ are the Jacobians of g and f.

$$\begin{aligned} ad_f^0 g &= g \\ ad_f^{k+1} g &= ad_f(ad_f^k g) \end{aligned}$$

$\mathbf{V^*(M)}$: denotes the set of all one forms defined on M. In local coordinates a one form w is represented as $w = w_1 dx_1 + \cdots + w_n dx_n$.

$\mathbf{d : V^*(M) \longrightarrow V^*(M)}$: denotes the differential and in local coordinates its action on $h \in F(M)$ is:

$$d : h \longrightarrow dh = \frac{\partial h}{\partial x_1} dx_1 + \cdots + \frac{\partial h}{\partial x_n} dx_n$$

$\mathbf{\prec w, f \succ : V^*(M) \times V(M) \longrightarrow F(M)}$: denotes the dual product between one forms and vector fields $w \in V^*(M), f \in V(M)$ and it is defined in local coordinates as:

$$\prec w, f \succ = w_1 f_1 + \cdots + w_n f_n$$

Definition 1: Let $f \in V(M)$, there are three kinds of Lie derivatives related to f, which are expressed in local coordinates as follows:

1. if $h \in F(M)$ $\qquad L_f : F(M) \longrightarrow F(M)$

$$L_f(h) = \prec dh, f \succ = \sum_{i=1}^{n} f_i(x) \frac{\partial h}{\partial x_i}$$

the derivative of h along the direction defined by the vector field f.

2. $g \in V(M)$ $\qquad L_f : V(M) \longrightarrow V(M)$

$$\begin{aligned} L_f g &= [f,g] \\ L_f^0 g &= g \\ L_f^{k+1} g &= L_f(L_f^k g) \end{aligned}$$

3. $w \in V^*(M)$ $\qquad L_f : V^*(M) \longrightarrow V^*(M)$

$$L_f(w) = (\frac{\partial w^*}{\partial x} f)^* + w\frac{\partial f}{\partial x}$$

where $*$ denotes transpose and $\frac{\partial w^*}{\partial x}$ and $\frac{\partial f}{\partial x}$ are Jacobian matrices.

the three types of Lie derivatives are related by the following so called **Leibnitz** formula:

$$L_f \prec f, g \succ = \prec L_f w, g \succ + \prec w, L_f g \succ$$

Definition 2: a set of vector fields $X_1 \ldots X_l \in V(M)$ is involutive if there exists scalar functions $\nu_{ijk} \in F(M)$ such that:

$$[X_i, X_j](x) = \sum_{i=1}^{l} \nu_{ijk}(x) X_k(x) \qquad 1 \leq i,j \leq l \quad i \neq j$$

Frobenius Theorem: A set of linearly independent vector fields is completely integrable if and only if it is involutive.

Definition 3: Consider the time-invariant n-dimensional linear control system with m controls:

$$\dot{x} = Ax + Bu \tag{2.1}$$

Let

$$\begin{aligned} r_0 &= \text{rank}B \\ r_j &= \text{rank}\{B, AB, \ldots, A^jB\} - \text{rank}\{B, \ldots, A^{j-1}B\} \\ & \quad 1 \leq j \leq n-1 \end{aligned}$$

The kronecker indices k_i are defined as the number of r_j's that are $\geq i$. Notice that:

$k_1 \geq k_2 \geq \cdots \geq k_m$ and

$\sum_{i=1}^{m} k_i = n$

Fact : The system (2.1) is equivalent to a linear system in **Brunovsky** canonical form :

$$\dot{z} = \hat{A}z + \hat{B}v$$

where:

$$\hat{A} = \begin{pmatrix} \underbrace{\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}}_{k_1} & \begin{matrix} 0 & 0 & \cdots & 0 \\ 0 & & & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & 0 \end{matrix} & \cdots & \begin{matrix} 0 & \cdots & 0 \\ & & \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & & \cdots & 0 \\ \vdots & & & & \vdots \\ & & & & \\ 0 & & \cdots & & 0 \\ & \vdots & & & \\ & \vdots & & & \end{matrix} & \underbrace{\begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \end{pmatrix}}_{k_2} & \cdots & \begin{matrix} 0 & \cdots & 0 \\ & & \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & \cdots & 0 \\ & & & \\ \vdots & & & \vdots \end{matrix} & \begin{matrix} \vdots & & \ddots & \vdots \\ \vdots & & \ddots & \vdots \end{matrix} & \cdots & \begin{matrix} \vdots & & \vdots \\ & & \\ 0 & \cdots & 0 \end{matrix} \\ \begin{matrix} 0 & 0 & \cdots & 0 \\ & & & \\ \vdots & & & \vdots \\ 0 & & \cdots & 0 \end{matrix} & \cdots & \cdots & \underbrace{\begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \end{pmatrix}}_{k_m} \end{pmatrix}$$

$$\widehat{B} = \begin{pmatrix} & & & & & \\ k_1 & \left\{\begin{matrix} 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \end{matrix}\right. & & & & \\ & \left.\begin{matrix} 1 & 0 & 0 & \cdots & 0 \end{matrix}\right. & & & & \\ & \left.\begin{matrix} 0 & & & \cdots & 0 \\ \vdots & & & & \vdots \end{matrix}\right. & & & & \\ k_2 & \left\{\begin{matrix} 0 & 1 & 0 & \cdots & 0 \end{matrix}\right. & & & & \\ & \vdots & & & & \\ & \left.\begin{matrix} 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \end{matrix}\right. & & & & \\ k_m & \left\{\begin{matrix} 0 & 0 & 0 & \cdots & 1 \end{matrix}\right. & & & & \end{pmatrix}$$

Chapter 3

Feedback equivalence to linear controllable systems

In this chapter, we introduce and discuss the concept of feedback equivalence among systems affine in control.

In section 3.1, we introduce the equivalence relation that exists among nonlinear systems and we characterize classes of equivalence by characterizing diffeomorphisms that exist among systems that belong to the same equivalence class.

In particular, in section 3.2, we recall a theorem of *Jacubczyk-Respondek* [13] and *Hunt-Su* [11] on necessary and sufficient conditions for a nonlinear system to be locally equivalent to a linear controllable system.

In section 3.3, we recall the definition of controllability indices for nonlinear systems affine in control and we discuss the construction of the feedback transformation given by *Hunt-Su* [10], which takes the nonlinear systems enjoying certain properties into linear controllable ones; we also point out that the controllability indices of such nonlinear systems coincide with those computed for the linearized system around any equilibrium point.

3.1 Equivalence of systems

Let us consider two control systems:

$S_1 : \dot{x} = F(x_1, x_2, \ldots, x_n, u_1, \ldots, u_m) \qquad x(0) = x_0 \in M$

$S_2 : \dot{z} = G(z_1, z_2, \ldots, z_n, v_1, \ldots, v_m) \qquad z(0) = z_0 \in N$

We consider the transformation (which will be called Feedback transformation or F-transformation) consisting of:

T : a state space diffeomorphism $T = (T_1, \ldots, T_n) : U \longrightarrow V$ where U and V are open subsets in M and N respectively.

$$\begin{aligned} z_1 &= T_1(x) \\ z_2 &= T_2(x) \\ &\vdots \\ z_n &= T_n(x) \end{aligned}$$

S : state feedback and affine change of coordinates of the control space R^m over the ring of smooth functions $C^\infty(U)$.

$$\begin{aligned} v_1 &= S_1(x, u) \\ v_2 &= S_2(x, u) \\ &\vdots \\ v_m &= S_m(x, u) \end{aligned}$$

We require the $m \times m$ matrix $[\frac{\partial S_i}{\partial u_j}(x)]$ to be invertible for every $x \in U$.

Definition: The system S_1 is F-related to the system S_2 if there exists an F-transformation (T, S) on $U \times R^m$ such that for each state $x_0 \in U$ and each admissible control u the following holds:

If we let:

$$\begin{aligned} z_0 &= T(x_0) \\ z(t) &= T(x) \\ v(t) &= S(x, u) \end{aligned}$$

then $$z(t) = z(t, z_0, v(t)).$$

It turns out that the F-relation among systems is an equivalence relation, we say then that S_1 is F-equivalent to S_2.

We are particularly interested in mapping (via an F-transformation) the nonlinear system:

$$\begin{aligned} \dot{x}(t) &= f(x(t)) + \sum_{i=1}^{m} g_i(x(t))u_i(t) \qquad (3.1) \\ x(0) &= 0 \\ f(0) &= 0 \end{aligned}$$

to a controllable linear system:

$$\begin{aligned} \dot{z}(t) &= Az(t) + Bv(t) \qquad (3.2) \\ z(0) &= 0 \end{aligned}$$

with kronecker indices $k_1, k_2, \ldots, k_m$.

Since system (3.2) can be transformed by means of state space coordinates change and feedback into a system in Brunovsky canonical form, there is no change in the strength of the result if we take (A, B) already in the canonical form i.e:

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

The most straightforward case is when the nonlinear system is in Block triangular form. This form has been introduced by *G.Meyer* at NASA Ames Research Center [17], who recognized that it can be easily transformed to Brunovsky canonical form.

Definition: Given the nonlinear system (3.1), suppose $n = km$ where:

n = dimension of the state space

m = dimension of the control space

then if we partition the state x in k m-dimensional subvectors $x_1, \ldots, x_k$ and if we call $x_{k+1} = u$ then system (3.1) is in Block triangular form if:

$$\dot{x}_i = F(x_1, \ldots, x_i, x_{i+1}) \qquad i = 1, \ldots, k = \frac{n}{m}$$

It is easy to transform this system to canonical form by letting the vector:

$$\begin{array}{rcccl} z_1 & = & T(x_1) & = & x_1 \\ z_2 & = & \dot{z}_1 & = & T(x_1, x_2) \\ & \vdots & & \vdots & \\ z_k & = & \dot{z}_{k-1} & = & T(x_1, x_2, \ldots, x_k) \\ v & = & \dot{z}_k & = & S(x_1, \ldots, x_k, x_{k+1} = u) \end{array}$$

By just reindexing the new state variables we get:

$$\dot{z} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} z + \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} v$$

3.2 Necessary and sufficient conditions of transformability

3.2.1 Single-input systems

Consider the single-input system:

$$\begin{array}{rcll} \dot{x}(t) & = & f(x(t)) + g(x(t))u(t) & (3.3) \\ x(0) & = & 0 \qquad x \in M = R^n & \end{array}$$

where $f, g \in V(R^n)$ $f(0) = 0$.

The first result is due to *Brockett* [2]. He considered change of coordinates in the state space and feedback of the form $v = u + \alpha(x)$, where v is the control variable of the linearized system.

Theorem 3.1 (Brockett [2]) *Let $f, g : R^n \longrightarrow R^n$ be analytic vector fields. The system (3.3) with equilibrium point at x=0 is, in a neighborhood of 0, F-equivalent to one of the form:*

$$\dot{x} = Ax + bu$$

with (A,b) a controllable pair in canonical form if and only if $ad_f^k(g)_{k=0}^{n-1}$ spans R^n at x=0 and for k,m integers between 0 and n-1 there exists $d_i \in C^w(R^n)$ so that:

$$[ad_f^k(g), ad_f^m(g)] = \sum_{i=1}^{max(k,m)} d_i(x) ad_f^{i-1}(g)$$

These conditions have been later relaxed by *Su* [21] by allowing more general feedbacks of the form: $v = \alpha(x) + \beta(x)u$.

The necessary and sufficient conditions of existence are naturally less restrictive than Brockett's case.

Theorem 3.2 (Su [21]) *System (3.3) is transformable to Brunovsky canonical form in a neighborhood of the origin in R^n if and only if:*

- *the vectors $g, ad_f^1(g), \ldots, ad_f^{n-1}(g)$ span R^n about the origin.*
- *the set of vector fields $g, ad_f^1(g), \ldots, ad_f^{n-2}(g)$ is involutive.*

3.2.2 Multi-input systems

Hunt, Su and Meyer [10], [11] generalized Theorem 3.2 seen above to multi-input systems.

We note that in the multi-input case, the Kronecker indices of the equivalent linear system come into account.

Jacubczyk and Respondek [13] proved that the controllability (Kronecker) indices for the nonlinear system (3.3) are invariant under feedback transformations.

Thus, there is a one to one correspondance between the set of all possible controllability indices (m integers k_i with the property $\sum_{i=1}^{m} k_i = n, k_i \geq 1$) and the set of equivalence classes of systems under feedback transformations. The algorithm for the computation of the Kronecker indices of the nonlinear system was proposed by *Hunt-Su* [10] and is very similar to the one known for linear systems. (See the Preliminaries.)

* *Algorithm for computing the Kronecker indices of a non linear control system:*

→ Form the array of vectors:

$$\begin{bmatrix} g_1 & g_2 & \cdots & g_m \\ [f,g_1] & [f,g_2] & \cdots & [f,g_m] \\ \vdots & & & \vdots \\ ad_f^{n-1}g_1 & \cdots & \cdots & ad_f^{n-1}g_m \end{bmatrix}$$

→ Set:

α_0 = number of linearly independent vector fields in the first row.

α_1 = number of linearly independent vector fields in the first two rows.

$\vdots$

α_{n-1} = number of linearly independent vector fields in the array.

→ Take:

$$\begin{aligned} r_0 &= \alpha_0 \\ r_1 &= \alpha_1 - \alpha_0 \\ &\vdots \\ r_{n-1} &= \alpha_{n-1} - \alpha_{n-2} \end{aligned}$$

k_i will then be the number of r_j's $\geq i$.

The set of k_i's computed this way will completely characterize the Brunovsky canonical form non linear system will be transformed to.

* *Necessary and sufficient conditions of transformability:*

In the general case (multi-input, feedback of the form $v = \alpha(x) + \beta(x)u$), the result was given by *Hunt-Su* [11].

Theorem 3.3 (Hunt-Su [11]) *A multi-input control system $\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i$ is F-transformable to a controllable linear system in Brunovsky form in a neighborhood of the origin in R^n if and only if:*

1. *The set $C = \{g_1, [f, g_1], \ldots, ad_f^{k_1-1}(g_1), g_2, [f, g_2], \ldots, ad_f^{k_2-1}(g_2), \ldots, g_m, [f, g_m], \ldots, ad_f^{k_m-1}(g_m)\}$ spans an n-dimensional space.*
2. *The sets $C_j = \{g_1, [f, g_1], \ldots, ad_f^{k_j-2}(g_1), g_2, [f, g_2], \ldots, ad_f^{k_j-2}(g_2), \ldots, g_m, [f, g_m], \ldots, (ad_f^{k_j-2}(g_m)\}$ are involutive for $j = 1, 2, \ldots, m$ and*
3. *The span of each C_j is equal to the span of $C_j \cap C$.*

where $k_1 \geq k_2 \geq \cdots \geq k_m$ are the Kronecker indices of both the nonlinear system and the linear system.

3.3 Construction of the F-transformation

In [10], *Hunt* and *Su* gave a procedure for construction of a transformation which takes $\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i$ into a controllable linear system in canonical form $\dot{z} = Az + Bv$ where:

$$\begin{aligned} z &= T(x) \\ v &= S(x, u) \end{aligned}$$

If $k_1, k_2, \ldots, k_m$ are the Kronecker indices of both systems, let:

$$\begin{aligned} \sigma_1 &= k_1 \\ \sigma_2 &= k_1 + k_2 \\ &\vdots \\ \sigma_m &= k_1 + k_2 + \cdots + k_m = n \end{aligned}$$

From the structure of the canonical form and the fact that we are using only coordinate changes and feedback, we have:

$$\begin{aligned} z_{i+1} &= \dot{z}_i \qquad \text{or} \\ T_{i+1}(x) &= \dot{T}_i(x) \\ &= \prec dT_i, f \succ + \sum_{j=1}^{m} \prec dT_i, g_j \succ u_j \\ &\quad \forall\ i = 1, \ldots, \sigma_1 - 1, \sigma_1 + 1, \ldots, \sigma_2 - 1, \\ &\quad \sigma_2 + 1, \ldots, \sigma_{m-1} - 1, \sigma_{m-1} + 1, \ldots, \sigma_m - 1 \end{aligned}$$

but since T_{i+1} depends only on x, we have:

$$\left.\begin{aligned} \prec dT_i, g_j \succ &= 0 \\ \prec dT_i, f \succ &= T_{i+1} \\ & i = 1, \ldots, \sigma_1 - 1, \sigma_1 + 1, \ldots, \sigma_m - 1. \\ & j = 1, \ldots, m. \end{aligned}\right\} \tag{3.4}$$

on the other hand for $i = 1, 2, \ldots, m$:

$$\begin{aligned} v_i &= \dot{z}_{\sigma_i} \qquad \text{or} \\ S_i(x, u) &= \dot{T}_{\sigma_i}(x) \\ &= \prec dT_{\sigma_i}, f \succ + \sum_{j=1}^{m} \prec dT_{\sigma_i}, g_j \succ u_j \end{aligned}$$

Therefore, the set of partial differential equations the F-transformation (T, S) has to satisfy are:

$$\left.\begin{aligned} \prec dT_i, g_j \succ &= 0 \\ \prec dT_i, f \succ &= T_{i+1} \\ & i = 1, 2, \ldots, \sigma_1 - 1, \sigma_1 + 1, \ldots, \sigma_2 - 1, \sigma_2 + 1, \\ & \ldots, \sigma_{m-1} - 1, \sigma_{m-1} + 1, \ldots, \sigma_m - 1 = n - 1 \\ & j = 1, 2, \ldots, m. \end{aligned}\right\} \tag{3.5}$$

$$\begin{aligned} \prec dT_{\sigma_1}, f + \textstyle\sum_{i=1}^{m} u_i g_i \succ &= S_1 \\ &\vdots \\ \prec dT_{\sigma_m}, f + \textstyle\sum_{i=1}^{m} u_i g_i \succ &= S_m \end{aligned}$$

Using Leibnitz formula we can rewrite equations (3.5) as:

$$
\begin{aligned}
&\prec dT_1, ad_f^i(g_j) \succ &&= 0 \quad i = 0,1,\ldots,k_1-2 \text{ and } j=1,\ldots,m \\
&\prec dT_{\sigma_1+1}, ad_f^i(g_j) \succ &&= 0 \quad i = 0,1,\ldots,k_2-2 \text{ and } j=1,\ldots,m \\
&&&\vdots \\
&\prec dT_{\sigma_{m-1}+1}, ad_f^i(g_j) &&= 0 \quad i = 0,1,\ldots,k_m-2 \text{ and } j=1,\ldots,m. \qquad (3.6) \\
&\prec dT_{\sigma_1}, f+\textstyle\sum_{i=1}^m u_i g_i \succ &&= S_1 \\
&&&\vdots \\
&\prec dT_n, f+\textstyle\sum_{i=1}^m u_i g_i \succ &&= S_m
\end{aligned}
$$

with the determinant of:

$$
\begin{bmatrix}
\prec dT_1, ad_f^{k_1-1}(g_1) \succ & \cdots & \prec dT_1, ad_f^{k_1-1}(g_m) \succ \\
\vdots & & \vdots \\
\prec dT_{\sigma_{m-1}+1}, ad_f^{k_m-1}(g_1) \succ & \cdots & \prec dT_{\sigma_{m-1}+1}, ad_f^{k_m-1}(g_m) \succ
\end{bmatrix}
$$

being non zero.

Let s_i be the number of times $k_1 - i$ appears in the set C defined in Theorem 3.3. Let s_{k_1} be the number of linearly independent vectors in $\{g_1,\ldots,g_m\}$. The partial differential equations in equations (3.6) are solved by introducing parameters $t_1,\ldots,t_n$ and solving the ordinary differential equations:

$$
\begin{aligned}
\frac{dx(t_1)}{dt_1} &= ad_f^{k_1-1}(g_1) \qquad x(0)=0 \\
\frac{dx(t_1,t_2)}{dt_2} &= ad_f^{k_1-1}(g_2) \qquad x(t_1,0)=0
\end{aligned}
$$

continue in this manner to solve s_1 systems of equations ending with:

$$
\begin{aligned}
\frac{dx(t_1,\ldots,t_{s_1})}{dt_{s_1}} &= ad_f^{k_1-1}(g_{s_1}) \\
x(t_1,\ldots,t_{s_1-1},0) &= x(t_1,\ldots,t_{s_1-1})
\end{aligned}
$$

We then solve:

$$
\begin{aligned}
dx/dt_{s_1+1} &= ad_f^{k_1-2}(g_1) \qquad \text{with } x(t_1,\ldots,t_{s_1},0) = x(t_1,\ldots,t_{s_1}) \\
&\vdots \\
dx/dt_{s_1+s_2} &= ad_f^{k_1-2}(g_{s_2}) \qquad \text{with } x(t_1,\ldots,t_{s_1+s_2-1},0) = x(t_1,\ldots,t_{s_1+s_2-1})
\end{aligned}
$$

We continue this process until we have introduced parameters $t_1, \ldots, t_n$ ending with the solution of:

$$\frac{dx}{dt_n} = g_m \qquad \text{with} \quad x(t1, \ldots, t_{n-1}, 0) = x(t_1, \ldots, t_{n-1})$$

We now solve for $t_1, \ldots, t_n$ in terms of $x_1, \ldots, x_n$.

Let:

$$\begin{aligned} T_1 &= t_1(x) \\ T_{\sigma_1+1} &= \tilde{t}(x) \end{aligned}$$

where $\tilde{t}$ is the parameter which was introduced when we solved $dx/d\tilde{t} = ad_f^{k_2-1}(g_2)$.

$$T_{\sigma_2+1} = \tilde{\tilde{t}}(x)$$

where $\tilde{\tilde{t}}$ is the parameter which was introduced when we solved $dx/d\tilde{\tilde{t}} = ad_f^{k_3-1}(g_3)$.

We obtain this way the m leading components of the F-transformation $T_1, T_{\sigma_1+1}, \ldots, T_{\sigma_{m-1}+1}$.

The other components are found by:

$$\begin{aligned} T_{i+1} &= \prec dT_i, f \succ \\ &\quad i = 1, \ldots, \sigma_1 - 1, \sigma_1 + 1, \ldots, \sigma_{m-1} - 1, \\ &\quad \sigma_{m-1} + 1, \ldots, \sigma_m - 1 = n - 1 \\ S_1 &= \prec dT_{\sigma_1}, f + \sum_{i=1}^{m} u_i g_i \succ \\ &\vdots \\ S_m &= \prec dT_{\sigma_m}, f + \sum_{i=1}^{m} u_i g_i \succ \end{aligned}$$

We can see that, in order to obtain the F-transformation, we must solve n systems of n ordinary differential equations where the solution of each system depends on the solution of the previous system. As one can see, this is not always an easy nor possible task to do.

In summary, given the nonlinear system $\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i$ where the vector fields $f, g_1, \ldots, g_m$ satisfy the necessary and sufficient conditions of Theorem

3.3, there exists a transformation consisting of a change of coordinates and state feedback which transforms the nonlinear system to a controllable linear one. To construct such a transformation, the system of partial differential equations (3.6) must be solved.

Chapter 4

Invertibility of nonlinear systems

This chapter deals with the concept of left and right invertibility for multivariable nonlinear systems.

Once again, we consider systems of the following form:

$$\begin{aligned} \dot{x}(t) &= f(x(t)) + \sum_{i=1}^{m} u_i(t) g_i(x(t)) \qquad x(0) = x_0 \in M \qquad (4.1) \\ y(t) &= h(x(t)) \end{aligned}$$

where the state space M is a connected real analytic manifold, $f, g_1, \ldots, g_m$ are analytic vector fields on M, $x \longrightarrow h(x) = (h_1(x), \ldots, h_l(x))$ is a real analytic mapping from M into R^l, and $u_i \in U$, the class of real analytic functions from $[0, \infty)$ into R, the real numbers. If $x_0 \in M$ and $u = (u_1, \ldots, u_m)$ is an admissible control, we denote the resulting solution to the above differential equation by $x(t, u, x_0)$ and denote $h(x(t, u, x_0))$ by $y(t, u, x_0)$.

4.1 Left-invertibility

In the left-invertibility problem, we are interested in conditions which ensure that in system (4.1), different input functions produce different output functions. If this is the case, then the input-output map is invertible from the left and it is possible to reconstruct uniquely the input acting on the system from the knowledge of the corresponding output. Since, as we know, the input-output map of a nonlinear

system depends on the initial state x_0, one has to incorporate the dependence on the initial state into a precise definition of invertibility.

Definition: The nonlinear system (4.1) is said to be *left-invertible at* $x_0 \in M$ if whenever u and $\hat{u}$ are distinct admissible controls,

$$y(t, u, x_0) \neq y(t, \hat{u}, x_0)$$

for at least a value of $t \geq 0$.

The system (4.1) is *strongly invertible at* $x_0 \in M$ if there exists an open neighborhood V of x_0 such that for all $x \in V$, the system is left-invertible at x.

The system (4.1) is *strongly invertible* if there exists an open and dense submanifold M_0 of M such that for all $x_0 \in M_0$ the system is strongly invertible at x_0.

Clearly, left-invertibility at x_0 is equivalent to the Input-Output map described by (4.1) being injective. Thus, given the output $y(.)$ for a system which is invertible at x_0, one can, in theory, determine the control which was applied.

If a system is invertible at x_0, it is natural to look for a second system which acts as a left-inverse for the original system. The inverse system is a nonlinear system which, when driven by appropriate derivatives of $y(., u, x_0)$ produces $u(.)$ as its output. The left-inverse provides a practical method for determining u, and has many applications, e.g, the tracking problem.

For linear systems, this problem was first considered by *Brockett* and *Mesarovic* in 1965 [3]. Then Silverman [20] proposed an algorithm in 1969 for multivariable linear systems which does the following:

Given the linear multivariable time-invariant system:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

$x \in R^n, u \in R^m, y \in R^m$

The algorithm generates a left-inverse system:

$$\begin{aligned} \dot{z} &= \hat{A}z + \hat{B}\hat{y} \\ u &= \hat{C}z + \hat{D}\hat{y} \end{aligned}$$

where $\hat{y}$ is composed of components of $y(t), y^1(t), \ldots, y^\alpha(t)$ where α is the relative order of the system.

These ideas have been generalized by *Hirschorn* [7], [8], [9] to the nonlinear case.

Remark: For single input single output systems, necessary and sufficient conditions for invertibility are known and can be easily be stated. In the multivariable case, the problem is much more involved and the amount of complexity increases sharply. We will present the single input single output case first to clarify the algorithm in the multivariable case.

4.1.1 Single input single output case

Consider the single input single output affine nonlinear system:

$$\begin{aligned} \dot{x}(t) &= f(x(t)) + g(x(t))u(t) \\ y(t) &= h(x(t)) \quad x(0) = x_0 \in M \end{aligned} \tag{4.2}$$

M connected analytic manifold.

f, g are analytic vector fields on M.

h analytic real-valued function on M.

$u \in U$ the class of real analytic functions from $[0, \infty)$ into R.

Before stating the theorem that gives the necessary and sufficient conditions of strong invertibility of system (4.2), let us define the relative order α of system (4.2).

Definition: The relative order α of the nonlinear system (4.2) is the last nonnegative integer k such that $L_g L_f^{k-1} h \not\equiv 0$ on M and $L_g L_f^j h \equiv 0 \quad \forall\, 0 \leq j < k-1$.

or $\alpha = \infty$ if $L_g L_f^k h \equiv 0 \quad \forall k \geq 0$.

Theorem 4.1 (Hirschorn [7]) *The non linear system (4.2) is strongly invertible if and only if* $\alpha < \infty$.

To see how the relative order of the system is related to its strong invertibility, we have to remember that our aim is to solve for the control $u(t)$ as a function of the state and the output $y(t)$.

To solve the output equation $y(t) = h(x(t))$ for u, it will be necessary to differentiate y:

$$\begin{aligned} \frac{dy}{dt} &= \frac{\partial h}{\partial x}\dot{x} \\ &= \frac{\partial h}{\partial x}(f(x) + g(x)u) \\ &= L_f h(x) + L_g h(x).u(t) \end{aligned}$$

If $L_g h \not\equiv 0$ then $\alpha = 1$.

If $L_g h \equiv 0$ then we get :

$$\frac{dy}{dt} = L_f h(x)$$

therefore we should differentiate once more ;

$$\frac{d^2y}{dt^2} = L_f^2 h(x) + L_f L_g h(x).u(t)$$

If $L_f L_g h \not\equiv 0$ then $\alpha = 2$.

If $L_f L_g h \equiv 0$ then $\alpha > 2$ and we should again differentiate.

We can go on like this until we reach the relative order α, we obtain then:

$$\frac{d^\alpha y}{dt^\alpha} = L_f^\alpha h(x) + L_f L_g^{\alpha-1} h(x).u(t)$$

where $L_f L_g^{\alpha-1} h$ is $\not\equiv 0$.

Let then $M_\alpha = \{x \in M / L_f L_g^{\alpha-1} h(x) \neq 0\}$.

M_α is called *the inverse submanifold* of the system. Because of the analyticity of the function $L_f L_g^{\alpha-1} h$, M_α will be an open dense subset of M, hence a submanifold of M.

We can also see that M_α will provide the state space for the left-inverse system.

Indeed, for all $x \in M_\alpha$, we can write:

$$u(t) = \frac{y^\alpha(t) - L_f^\alpha h(x)}{L_f L_g^{\alpha-1} h(x)}$$

replacing this $u(t)$ into system (4.2), we get:

$$\begin{aligned} \dot{x} &= f(x) + g(x).\frac{y^\alpha(t) - L_f^\alpha h(x)}{L_f L_g^{\alpha-1} h(x)} \\ &= \left(f(x) - \frac{g(x).L_f^\alpha h(x)}{L_f L_g^{\alpha-1} h(x)} \right) + \frac{g(x)}{L_f L_g^{\alpha-1} h(x)}.y^\alpha(t) \end{aligned}$$

Conclusion:

Suppose that the nonlinear system (4.2) is strongly invertible with relative order α, initial state $x_0 \in M$, and inverse submanifold M_α, then the system:

$$\dot{z} = F(z) + G(z)v \qquad z(0) = x_0 \tag{4.3}$$

$$w = H(z) + K(z)v \tag{4.4}$$

where $z \in M_\alpha, v \in U$,

$$K(z) = \frac{1}{L_f L_g^{\alpha-1} h(z)} \tag{4.5}$$

$$H(z) = -\frac{L_f^\alpha h(z)}{L_f L_g^{\alpha-1} h(z)} \tag{4.6}$$

$$F(z) = f(z) - \frac{g(z) L_f^\alpha h(z)}{L_f L_g^{\alpha-1} h(z)} \tag{4.7}$$

$$G(z) = \frac{g(z)}{L_f L_g^{\alpha-1} h(z)} \tag{4.8}$$

acts as a *left-inverse* for the original system (4.2).

4.1.2 Generalization to multivariable systems

The case of multi-input multi-output systems is of course more delicate because the term that multiplies the vector input, $L_f L_g^k h(x)$, is a matrix and not a real valued function anymore, so we have to deal with non singularity of matrices instead of just checking if a function is zero or non zero.

In this section, we try to generalize the *structure algorithm* proposed by *Silverman* [20] to nonlinear systems.

The main difference is in the fact that we are dealing with matrices whose entries are smooth real valued functions instead of real numbers. So obviously, special care should be taken when checking rank conditions and nonsingularity of matrices.

Given a nonlinear system:

$$\begin{aligned} \dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x)u_i \\ y &= h(x) \end{aligned} \tag{4.9}$$

$x \in M$ where M is a connected C^∞ manifold.

$f, g_1, \ldots, g_m$ are C^∞ vector fields.

$h = (h_1, \ldots, h_m)^T \quad h_i \quad i = 1, \ldots, m$ is a C^∞ real valued function on M.

The first step is to differentiate y:

$$\frac{dy}{dt} = \begin{pmatrix} L_f h_1(x) \\ \vdots \\ L_f h_m(x) \end{pmatrix} + \begin{pmatrix} L_{g_1} h_1(x) & \cdots & L_{g_m} h_1(x) \\ \vdots & & \vdots \\ L_{g_1} h_m(x) & \cdots & L_{g_m} h_m(x) \end{pmatrix} .u$$

To make the notation simpler, we call:

$$L_f h = \begin{pmatrix} L_f h_1 \\ \vdots \\ L_f h_m \end{pmatrix} \qquad L_G h = \begin{pmatrix} L_{g_1} h_1 & \cdots & L_{g_m} h_1 \\ \vdots & & \vdots \\ L_{g_1} h_m & \cdots & L_{g_m} h_m \end{pmatrix}$$

So, we can write :

$$\frac{dy}{dt}(t) = L_f h(x(t)) + L_G h(x(t))u(t) \tag{4.10}$$

Definition: Let $N(x)$ be an $m \times m$ matrix whose entries are smooth real valued functions on M.

R = field of real numbers.

$K(C^w)$ = quotient field associated with the ring C^w of real analytic functions.

with $N(x)$ we associate :

1. $r_K(N)$ = dimension of the $K(C^w)$-vector space generated by the rows of $N(x)$.

2. $r_R(N)$ = dimension of the R-vector space generated by the rows of $N(x)$.

Remarks:

1. Clearly the two integers $r_K(N)$ and $r_R(N)$ are such that:

$$r_R(N) \geq r_K(N)$$

2. If we try to row-reduce $N(x)$ by multiplying in the left by a nonsingular matrix V of real numbers, then we can see that:

$$r_R(N) = r_K(N)$$

if and only if the process of row-reduction of N leaves VN with a number of nonzero rows equal to $r_K(N)$.

3. Both $r_K(N)$ and $r_R(N)$ are dependent on the point $x(t)$, but since in this part we do not deal with the problem of singular points, we suppose that the ranks are constant over M.

So now, starting from equation (4.10) which is:

$$\frac{dy}{dt} = L_f h(x) + L_G h(x) u$$

If $r_K(L_G h(x)) = m$, then we can solve for $u(t)$ as a function of $x(t)$ and $y(t)$:

$$u(t, x(t)) = [L_G h(x)]^{-1} (\frac{dy}{dt} - L_f h(x)) \qquad (4.11)$$

If $r_K(L_G h(x)) < m$, then let $r_K(L_G h) = q_1$

Suppose $\mathbf{r_K(L_G h) = r_R(L_G h)}$

then there exists an $m \times m$ nonsingular real matrix S_1 such that:

$$D_1(x) \stackrel{\text{def}}{=} S_1 L_G g(x) = \begin{pmatrix} \bar{D}_1(x) \\ 0 \end{pmatrix}$$

where $\bar{D}_1(x)$ has q_1 rows and rank q_1.

The system (P_1) is then defined as:

$$(P_1) \qquad \begin{cases} \dot{x}(t) &= f(x(t)) + \sum_{i=1}^{m} g_i(x(t)) u_i(t) \\ y_1(t) &= C_1(x(t)) + D_1(x(t)) u(t) \end{cases}$$

where

$$\begin{aligned} y_1(t) &= S_1 dy/dt(t) \\ C_1(x) &= S_1 L_f h(x) \\ D_1(x) &= S_1 L_G h(x) \end{aligned}$$

It will be convenient to represent $y_1(t)$ and $C_1(x)$ in the partitioned form:

$$C_1(x) = \begin{pmatrix} \bar{C}_1(x) \\ \tilde{C}_1(x) \end{pmatrix} \qquad y_1(t) = \begin{pmatrix} \bar{y}_1(t) \\ \tilde{y}_1(t) \end{pmatrix}$$

where the bar and tilde indicate the first q_1 and $m - q_1$ elements respectively of the two vectors. System (P_1) becomes then:

$$(P_1) \qquad \begin{pmatrix} \bar{y}_1(t) \\ \tilde{y}_1(t) \end{pmatrix} = \begin{pmatrix} \bar{C}_1(x) \\ \tilde{C}_1(x) \end{pmatrix} + \begin{pmatrix} \bar{D}_1(x) \\ 0 \end{pmatrix} u(t)$$

Now if we differentiate $\tilde{y}_1$:

$$\begin{aligned} \frac{d\tilde{y}}{dt} &= \frac{\partial \tilde{C}_1}{\partial x} . \dot{x} \\ &= L_f \tilde{C}_1(x) + L_G \tilde{C}_1(x) u \end{aligned}$$

If $\qquad M_1 = \left(\begin{array}{c|c} I_{q_1} & 0 \\ \hline 0 & I_{m-q_1} \frac{d}{dt} \end{array} \right)$

then $$M_1 y_1 = \begin{pmatrix} \bar{y}_1 \\ \frac{d\tilde{y}_1}{dt} \end{pmatrix} = \begin{pmatrix} \bar{C}_1(x) \\ L_f \tilde{C}_1(x) \end{pmatrix} + \begin{pmatrix} \bar{D}_1(x) \\ L_G \tilde{C}_1(x) \end{pmatrix} u$$

Let $$q_2 = r_K \begin{pmatrix} \bar{D}_1(x) \\ L_G \tilde{C}_1(x) \end{pmatrix}$$

and suppose $$\mathbf{r_R} \begin{pmatrix} \bar{D}_1(x) \\ L_G \tilde{C}_1(x) \end{pmatrix} = \mathbf{r_K} \begin{pmatrix} \bar{D}_1(x) \\ L_G \tilde{C}_1(x) \end{pmatrix}$$

then there exists a real nonsingular $m \times m$ matrix S_2 such that:

$$D_2 \stackrel{\text{def}}{=} S_2 \begin{pmatrix} \bar{D}_1(x) \\ L_G \widetilde{C}_1(x) \end{pmatrix} = \begin{pmatrix} \bar{D}_2(x) \\ 0 \end{pmatrix}$$

where $\bar{D}_2(x)$ has q_2 rows and $r_K(\bar{D}_2(x)) = q_2$.

$$\begin{aligned} y_2(t) &= S_2 M_1 y_1(t) \\ &= S_2 \begin{pmatrix} \bar{C}_1(x) \\ L_f \widetilde{C}_1(x) \end{pmatrix} + S_2 \begin{pmatrix} \bar{D}_1(x) \\ L_G \widetilde{C}_1(x) \end{pmatrix} u(t) \\ &\stackrel{\text{def}}{=} C_2(x) + D_2(x) \end{aligned}$$

Similarly, if we partition $y_2(t)$ and $C_2(x)$ into $\begin{pmatrix} \bar{y}_2(t) \\ \widetilde{y}_2(t) \end{pmatrix}$ and $\begin{pmatrix} \bar{C}_2(x) \\ \widetilde{C}_2(x) \end{pmatrix}$ where the bar and tilde indicate the first q_2 and $m - q_2$ elements, we obtain:

$$(P_2) \qquad \begin{pmatrix} \bar{y}_2(t) \\ \widetilde{y}_2(t) \end{pmatrix} = \begin{pmatrix} \bar{C}_2(x) \\ \widetilde{C}_2(x) \end{pmatrix} + \begin{pmatrix} \bar{D}_2(x) \\ 0 \end{pmatrix} u$$

The remainder of the sequence (P_i) is defined inductively.

Let (P_k) denote the k^{th} system in the sequence:

$$(P_k) \qquad \begin{cases} \dot{x}(t) &= f(x(t)) + \sum_{i=1}^{m} g_i(x(t)) u_i(t) \\ y_k(t) &= C_k(t) + D_k(t) u(t) \end{cases}$$

$$D_k(x) = \begin{pmatrix} \bar{D}_k(x) \\ 0 \end{pmatrix} \qquad C_k(x) = \begin{pmatrix} \bar{C}_k(x) \\ \widetilde{C}_k(x) \end{pmatrix}$$

where $\bar{D}_k$ has q_k rows and rank q_k.

$\bar{C}_k$ has q_k rows and $\widetilde{C}_k$ has $m - q_k$ rows.

If $q_k < m$ then let $M_k = \left(\begin{array}{c|c} I_{q_k} & 0 \\ \hline 0 & I_{m-q_k}\frac{d}{dt} \end{array}\right)$

$$M_k y_k(t) = \begin{pmatrix} \bar{y}_k \\ \frac{d\tilde{y}_k}{dt} \end{pmatrix} = \begin{pmatrix} \bar{C}_k(x) \\ L_f\widetilde{C}_k(x) \end{pmatrix} + \begin{pmatrix} \bar{D}_k(x) \\ L_G\widetilde{C}_k(x) \end{pmatrix} u$$

Let $q_{k+1} = \text{rank} \begin{pmatrix} \bar{D}_k(x) \\ L_G\widetilde{C}_k(x) \end{pmatrix}$

suppose $$\mathbf{r_K} \begin{pmatrix} \bar{D}_k(x) \\ L_G\widetilde{C}_k(x) \end{pmatrix} = \mathbf{r_R} \begin{pmatrix} \bar{D}_k(x) \\ L_G\widetilde{C}_k(x) \end{pmatrix}$$

then there exists a non singular real valued matrix S_{k+1}:

$$D_{k+1}(x) \stackrel{\text{def}}{=} S_{k+1} \begin{pmatrix} \bar{D}_k(x) \\ L_G\widetilde{C}_k(x) \end{pmatrix} = \begin{pmatrix} \bar{D}_{k+1}(x) \\ 0 \end{pmatrix}$$

where $D_{k+1}(x)$ has q_{k+1} rows and rank q_{k+1}.

$$y_{k+1}(t) = S_{k+1}M_k y_k(t) = \begin{pmatrix} \bar{y}_{k+1}(t) \\ \tilde{y}_{k+1}(t) \end{pmatrix}$$

$$C_{k+1}(x) = S_{k+1} \begin{pmatrix} \bar{C}_k(x) \\ L_f\tilde{C}_k(x) \end{pmatrix} = \begin{pmatrix} \bar{C}_{k+1}(x) \\ \tilde{C}_{k+1}(x) \end{pmatrix}$$

$$(P_{k+1}) \qquad \begin{pmatrix} \bar{y}_{k+1}(t) \\ \tilde{y}_{k+1}(t) \end{pmatrix} = \begin{pmatrix} \bar{C}_{k+1}(x) \\ \tilde{C}_{k+1}(x) \end{pmatrix} + \begin{pmatrix} \bar{D}_{k+1}(x) \\ 0 \end{pmatrix} u$$

Suppose there exists α such that $D_\alpha(x)$ has rank $m : q_\alpha = m$

Definition: α is called the *relative order* of the multivariable system.

The α^{th} system has the form:

$$(P_\alpha) \qquad y_\alpha(t) = C_\alpha(x) + D_\alpha(x)u$$

with rank $D_\alpha(x) = m$. Therefore, we can solve for the vector $u(t)$ as a function of $y_\alpha(t)$ and $x(t)$.

$$u(t) = D_\alpha^{-1}(x)\left(y_\alpha(t) - C_\alpha(x)\right) \tag{4.12}$$

Conclusion:

Given the nonlinear system:

$$\begin{aligned} \dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x)u_i \qquad (4.13) \\ y &= h(x) \end{aligned}$$

If system (4.13) is strongly invertible with relative order α , initial state $x_0 \in M$, then the system:

$$\dot{z} = F_{inv}(z) + G_{inv}(z)v \tag{4.14}$$

$$w = H_{inv}(z) + K_{inv}(z)v \tag{4.15}$$

where $z(0) = x_0, u \in U$,

$$\begin{aligned} H_{inv}(z) &= -D_\alpha^{-1}(z)C_\alpha(z) \\ K_{inv}(z) &= D_\alpha^{-1}(z) \\ F_{inv}(z) &= f(z) - G(z)D_\alpha^{-1}(z)C_\alpha(z) \\ G_{inv}(z) &= G(z)D_\alpha^{-1}(z) \end{aligned}$$

acts as a left-inverse to system (4.13).

It should be pointed out however, that the algorithm outlined above (page 26-31) to generate the left-inverse system assumes some rather restrictive assumptions on the system, e.g, rank conditions and also the number of inputs has to be equal to the number of outputs. An other remark is that by restricting the left-inverse system to be defined on M_α, the inverse submanifold of the system, we circumvent the problem of singular points.

4.2 Right-invertibility or [Functional controllability]

In this section, we are concerned with determining the functions $f(t)$ which can be realized as the output of the nonlinear system (4.1) driven by a suitable input function. While the left-invertibility is related to the *injectivity* of Input/Output map of the nonlinear system (4.1), the right-invertibility is related to the *surjectivity* of the Input/Output map.

For linear systems, this classification problem was solved by *R. W Brockett* in 1965 [3]. This classification is very important in our work since we are interested in the output tracking problem. It gives us the class of functions that the output of our nonlinear system can track. Clearly, if $f(.) = y(., u, x_0)$ for some control u, then the required control u can be generated as the output of the left-inverse system studied in section 4-1, driven by an appropriate derivative of $f(t)$. In this case, the left-inverse system in section 4-1, is said to act as a right- inverse for the original system.

Theorem 4.2 (Hirschorn [7]) *Consider the nonlinear system (4.1) with relative order α.*

If $\alpha < \infty, x_0 \in M_\alpha$ and $f \in C^w(R)$ then there exists $u \in U$ such that $y(., u, x_0) = f(.)$ if and only if :

$$f^{(k)}(0) = L_f^k h(x_0) \qquad \text{for } k = 0, 1, \ldots, \alpha - 1$$

Recall from section 4-1 that if the nonlinear system:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \qquad x \in M \end{aligned} \tag{4.16}$$

is strongly invertible then the left inverse is:

$$\begin{aligned} \dot{z} &= F(z) + G(z)v \\ w &= H(z) + K(z)v \end{aligned} \tag{4.17}$$

The theorem says that the class of trajectories that can be realized as outputs of system (4.16) are trajectories $y_d(t)$ that satisfy:

$$y_d^{(k)}(0) = L_f^k h(x_0) \qquad \text{for } k = 0, 1, \ldots, \alpha - 1 \tag{4.18}$$

To see this, let $y_d(t) \in C^w(R)$, satisfying conditions (4.18), we want to find $u \in U$ such that:

$$y_d(.) = y(., u, x_0) \qquad \text{for some } x_0 \in M_\alpha$$

Since $y_d^{(\alpha)}(.) \in U$, if we let $y_d^{(\alpha)}(.)$ be the input to the left inverse system (4.17):

$$v = y_d^{(\alpha)}$$

and set $$z(t) = z(t, y_d^{(\alpha)}, x_0)$$

then the output to system (4.17) is:

$$w(t) = H(z(t)) + K(z(t)) y_d^{(\alpha)}(t)$$

Now, take $$\mathbf{u(t) = w(t)}$$

then, by differentiating the output equation of system (4.16) α times, we get:

$$y^{(\alpha)}(t) = L_f^{(\alpha)} h(x) + L_f L_g^{(\alpha-1)} h(x) u \tag{4.19}$$

replacing $u(t)$ by $w(t)$ in equation (4.19), we get :

$$y^{(\alpha)}(t) = L_f^{(\alpha)} h(x) + L_f L_g^{(\alpha-1)} h(x) \left(H(x(t)) + K(x(t)) y_d^{(\alpha)}(t) \right)$$

where

$$\begin{aligned} H(x(t)) &= -\frac{L_f^\alpha h(x(t))}{L_f L_g^{\alpha-1} h(x(t))} \\ K(x(t)) &= \frac{1}{L_f L_g^{\alpha-1} h(x(t))} \end{aligned}$$

$$y^\alpha(t) = L_f^\alpha h(x) + L_f L_g^{\alpha-1} h(x) \left[\left(L_f L_g^{\alpha-1} h(x) \right)^{-1} \left(-L_f^\alpha h(x) + y_d^\alpha(t) \right) \right]$$

or

$$y^\alpha(t) = y_d^\alpha(t) \tag{4.20}$$

but since

$$\begin{aligned} y_d(0) &= h(x_0) &&= y(0) \\ y_d^{(1)}(0) &= L_f h(x_0) &&= y^{(1)}(0) \\ &\vdots &&\vdots \\ y_d^{(\alpha-1)}(0) &= L_f^{(\alpha-1)} h(x_0) &&= y^{(\alpha-1)}(0) \end{aligned}$$

we can conclude that $\mathbf{y(t) = y_d(t)}$

We can see clearly now how this notion of left and right invertibility of nonlinear systems can be applied to the output tracking problem. That is:

Given the nonlinear system (4.16), if we want the output to track some desired trajectory, first, we have to check if this desired path can be realized as output of system (4.16) using conditions (4.18), then, we construct the left inverse system (4.17).

It suffices then to feed system (4.17) with the α^{th} derivative of the desired path $y_d(t)$ to obtain as output the required control $u_d(t)$.

Chapter 5

Output Tracking of nonlinear systems

In this chapter, we are interested in a deterministic servo-problem i.e, we are interested in designing a control law that forces the output of a nonlinear control system to perform a particular task despite the presence of disturbances.

There is a considerable literature dealing with this problem for linear control systems (Ref[3], [23]). Here we consider the tracking problem for the class of nonlinear systems affine in control:

$$\begin{aligned} \dot{x}(t) &= f(x(t)) + \sum_{i=1}^{m} g_i(x(t))u_i(t) \\ y(t) &= h(x(t)) \end{aligned} \tag{5.1}$$

where $x(t) \in R^n$, $u(t) \in R^m$, $y(t) \in R^l$.

$f, g_1, \ldots, g_m \in V(R^n)$, $h : R^n \longrightarrow R^l$ real smooth mappings.

The function $y_d : R \longrightarrow R^l$ is a "desired" output for the system. The objective is to obtain a robust tracking controller such that the system (5.1) with the controller has acceptable tracking performance or such that $e = y_d - h(x)$, the tracking error, is kept within a desirable tolerance.

For this purpose, two design schemes based on the theoretical concepts presented in Chapter 3 and 4 are described.

5.1 Output tracking design using feedback linearization

This section describes one of the main applications of the theory presented in Chapter 3 to the control system design problem.

The key concept of the approach is to simplify the representation of the plant dynamics by means of a change of coordinates of the state and control. The design proceeds in three steps. First, the given nonlinear system is transformed into a constant, decoupled, controllable linear representation. Second, standard linear and nonlinear design techniques, such as *Bode* plot, pole placement, LQR, or phase plane methods are used to design a control law for this simple representation. Third, the resulting control law is transformed back out into the original coordinates to obtain the control law in terms of the available controls.

G.Meyer at NASA Ames Research Center [17] proposed this scheme in the design of exact model following automatic pilots for vertical and short take-off aircrafts and has been applied to several aircraft of increasing complexity.

So, given the nonlinear system:

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i(t)$$

suppose there exists a nonsingular transformation (T, S) mapping our system (5.1) to a time-invariant and controllable linear system with state variables $z = T(x) = (z_1, \ldots, z_n)^T$ and control variables $v = S(x, u) = (v_1, \ldots, v_m)^T$ where:

$$\dot{z} = Az + Bv$$

We control the plant (the nonlinear system) by controlling the linear system. The proposed structure of the complete control system is specified in Figure 1. Note that the design is carried out on the "linear" side of the transformation.

More explicitly, the basic procedure will be as follows:

Given the path we want to track, $x_d(t)$, the transformation T maps the x-space to the z-space, therefore, we will try to accomplish this by making $z(t) = T(x(t))$

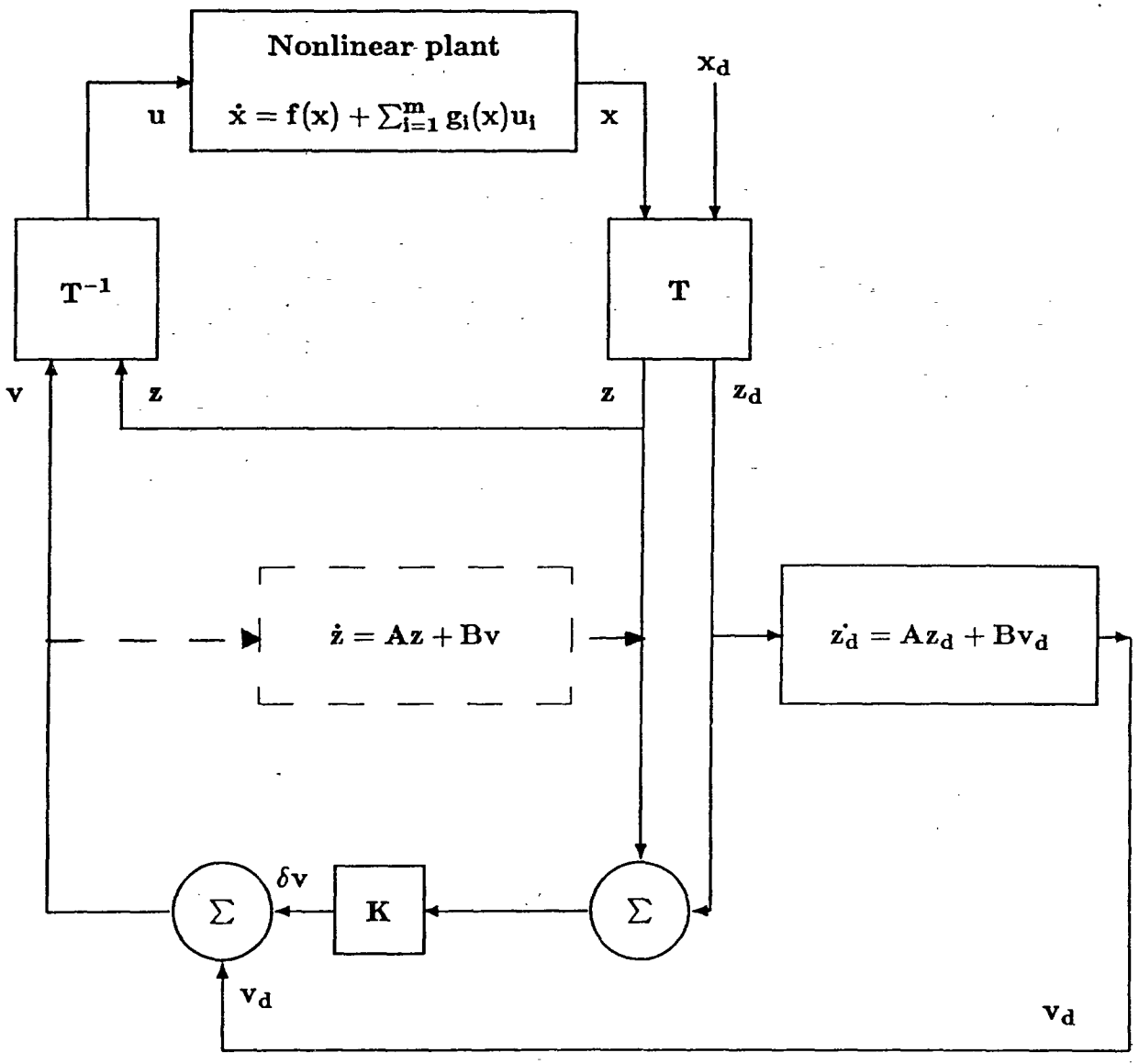


Fig. 1: Block Diagram of the first design scheme

track $z_d(t) = T(x_d(t))$.

In our theory, the linear system is in Brunovsky canonical form which is invaluable for the design of exact model followers. If we have a control system $\dot{z} = Az + Bv$ in Brunovsky form and a model to be followed $\dot{z_d} = Az_d + Bv_d$, linear design is used to find an open-loop command v_d by solving $\dot{z_d} = Az_d + Bv_d$ and since A and B are in canonical form, this is straightforward. Then z is compared to z_d to yield an error $e_z = z - z_d$.

$$\begin{aligned} \dot{e_z} = \dot{z} - \dot{z_d} &= A(z - z_d) + B(v - v_d) \\ &= Ae_z + B(v - v_d) \end{aligned}$$

Using linear feedback, we can design a regulator that stabilizes out the difference by placing the poles of the linear system in the desired positions, let:

$$\delta v = v - vd = Ke_z$$

K is an $m \times n$ matrix so that:

$$\dot{e_z} = Ae_z + BKe_z = (A + BK)e_z$$

Choosing K in such a way that $A + BK$ has negative eigenvalues, we get asymptotic stability of $e_z = 0$. Placing the eigenvalues is a simple task for a system in Brunovsky canonical form. Disturbances and variations in plant dynamics are handled in this way.

The controls v_d and δv are added and transformed through the inverse map T^{-1} to give a control $u = (u_1, \ldots, u_m)^T$ which is applied to the plant.

The main advantage of this in direct approach is that the difficult tasks of finding the open-loop control and the regulated control are performed on the linear system.

5.2 Output tracking using invertibility of systems

Based on the theory of invertibility and functional reproducibility for multivariable nonlinear systems, an application to the output tracking control of nonlinear

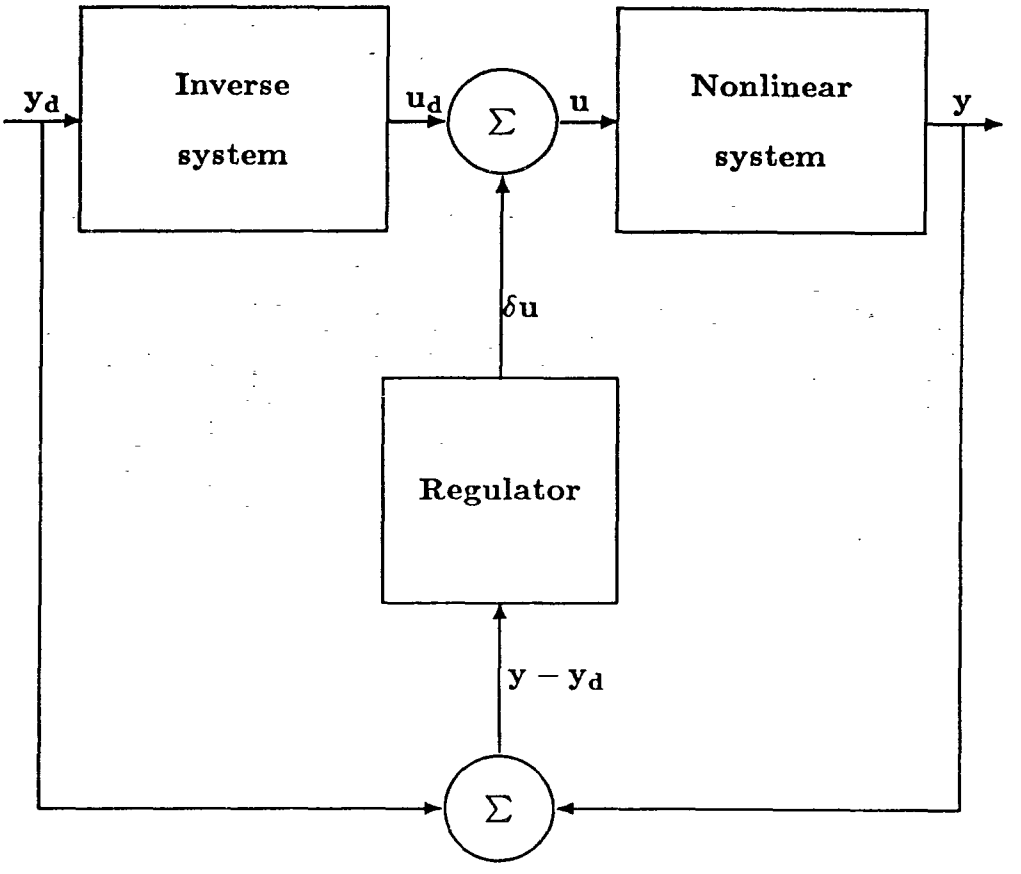


Fig. 2: Block Diagram of the second design scheme

systems is presented in this section.

The design approach presented is based on the inversion algorithm of Chapter 4. The application of the inversion algorithm to the nonlinear system gives rise to a left-inverse system which, when driven by appropriate derivatives of the output y, produces $u(.)$ as its output. The question of trajectory following by the output is related to right-invertibility of the nonlinear input-output map, and the ability of the nonlinear system to reproduce the reference path as its output. To obtain robustness in the control system under perturbations, design of a servocompensator around the inner loop using servomechanism theory is suggested.

For the system (4.1), the output tracking design will be done in three phases:

1. Find conditions on the class of outputs which the system can track (i.e identification of functions $y_d(t)$ which can appear as outputs for the system (4.1)):

$$y_d(.) = y(., u, x_0) \text{ for some admissible control } u$$

2. Construction of a control u_d for which $y_d(.) = y(., u, x_0)$. To avoid keeping track of degrees of differentiability, y_d is assumed to be infinitely differentiable (smooth).

   The basic idea in both linear and nonlinear tracking is to solve for the control, u_d, as a function of the desired output, y_d, and the state x of the system. This provides us with a left-inverse that generates the required control corresponding to the desired output.

3. Ideally, the combined relationship of $y_d(.)$ to $y(.)$ is an identity. However, robustness purposes dictate that a regulator be synthesized.

Chapter 6

CONDENS: a software package using symbolic manipulations

In this chapter, CONDENS, a software package using symbolic manipulations is presented. The main aim of CONDENS is to help the user in some symbolic calculations in differential geometry and its applications to control, with emphasis on the design of controllers for the output tracking problem.

In the first section, we present the programming objective and how the package can make available to the design engineer some design methods that rely on more or less sophisticated mathematical tools.

In the second section, a complete description of the different modules of CONDENS and how to use them is given.

In the last section, we present some examples with simulation results to test our programs and the performance of the system.

6.1 Programming objective

The impact of the computer on mathematics and its related fields is well-known. Perhaps, less well-known is the recent progress of the application of symbolic calculations in the more continuous parts of mathematics; such as mathematical analysis, differential equations, differential geometry and its applications in nonlinear control

theory.

The objective of our program is to implement the theory presented in Chapters 2-5.

The ultimate purpose of the system is to assist the user in the design process by automatically selecting and executing an appropriate design method. The final step is to validate and implement the design. This feature shifts a large amount of complexity of the design problem from the design engineer to the "knowledge base" of the CAE system.

An obvious but nonnegligible circumstance is the fact that, due to the use of symbolic calculations, one may carry out easy, "long and tedious" calculations with the computer, thus avoiding elementary mistakes, such as wrong signs, missing brackets, omitted symbols,...etc.

The system presents an other feature: It can automatically generate numerical programs for the solution of problems posed in symbolic form or for simulation purposes. Given a new control problem, the time involved in analyzing the theoretical results and then writing the Fortran code to execute it is eliminated. The mistakes and the time required to test and debug the Fortran code is also eliminated.

Most importantly, the system allows the engineer to interact with the computer for design at the symbolic manipulation level. In this way, he can modify his analysis or design problem by modifying the symbolic functional form of the model. The Fortran subroutines that he might have to modify by hand to accomplish this in conventional design procedures are written automatically for him.

There are several systems designed for symbolic calculations, e.g, FORMAC, MACSYMA, REDUCE. We have chosen the language MACSYMA as a basis for our developments. It may be implemented on any computer system supporting LISP, it is easily available and, consequently, widespread. Last, but not least, we are charmed by the interactive facilities of MACSYMA.

6.2 Description of CONDENS

Based on the theoretical concepts presented in previous chapters, the system, can treat feedback linearization and tracking control problems for certain classes of nonlinear systems.

The package contains a set of user-defined functions, it also contains two *help* functions: MENU and HELP. MENU will display the list of all the user-defined functions the package contains. HELP("fct-name") returns an information text describing the function, its syntax, how to enter its arguments, and an example.

All these functions are collected in a special file ("initfile.mac") with file addresses for all the functions. Once this file is loaded into MACSYMA, a user-defined function, not previously defined, will be automatically loaded into MACSYMA when it is called. This will be done by a login file using the setup- autoload command. So the first thing the user has to do once in MACSYMA, is to load the login file (load "initfile.mac";). He can then start the package using the command "condens();" which will give him some hints on how to use CONDENS.

All functions not explicitly described in this section are already available in the basic MACSYMA system. We recommend that the reader consult the *MACSYMA Reference Manual* for details.

Moreover, one has to be informed on how to start a MACSYMA session.

The programs that form CONDENS will be described by order of complexity, in three different subsections.

Subsection 6.2.1 will describe a set of user-defined functions that perform some differential geometric computations. Straightforward computations such as *Lie brackets* and *Lie derivatives* and more complex computations such as *Kronecker indices* or *Relative order* of nonlinear systems are included. The two help functions, MENU and HELP are also described in this subsection.

In subsection 6.2.2, we present two more sophisticated modules contained in CONDENS: TRANSFORM which addresses the feedback linearization problem, and INVERT which provides the left-inverse of a nonlinear control system.

Subsection 6.2.3 will present a more complete picture of what CONDENS can do. In particular, it describes two programs that address the output tracking design problem. FENOLS and TRACKS use the modules and user-defined functions presented in the two previous subsections to design a control law for a nonlinear system in order to force its output to follow some desired trajectory. FENOLS and TRACKS have also the capability of automatically generating FORTRAN codes for simulation purposes.

Before describing CONDENS in more detail, we shall make some general remarks relevant to all worked examples in each section.

1. Input lines always begin with a "(C_n)", which is the prompt character of MACSYMA, indicating that the system is waiting for a command.

2. Results of commands terminated by a ";" are printed.

3. Results of commands terminated by a "$" are not printed.

4. If we are working with a single-input, single-output system:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned}$$

then the vector fields f and g are entered in the form of lists:

$$\begin{aligned} f &: [f_1(x), f_2(x), \ldots, f_n(x)]; \\ g &: [g_1(x), g_2(x), \ldots, g_n(x)]; \\ h &: h(x); \end{aligned}$$

example:

For the system:

$$\begin{aligned} \dot{x_1} &= x_1^2 + 2x_1u \\ \dot{x_2} &= x_1x_2 + u \\ y &= x_1^2 + x_2^2 \end{aligned}$$

$$f : [x_1^2, x_1 x_2];$$
$$g : [2x_1, 1];$$
$$h : x_1^2 + x_2^2$$

5. If we are working with a multivariable control system:

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x) u_i$$
$$y = h(x)$$

$x \in R^n,\ u \in R^m,\ y \in R^m$

then the vecter field f is entered in the form:

$$f : [f_1(x), \ldots, f_n(x)]$$

but the m vector fields $g_1, g_2, \ldots, g_m$ are entered in the compact form:

$$g : [g_1, \ldots, g_m] \text{ where each } g_i \text{ is } g_i : [g_{i1}, \ldots, g_{in}]$$

example:

For the system:

$$\dot{x_1} = x_1^2 + 2x_1 u_1 + 2x_2 u_2$$
$$\dot{x_2} = x_1 x_2 + 2u_1 + u_2$$
$$y_1 = x_1^2$$
$$y_2 = x_1^2 + x_2^2$$

$$f : [x_1^2, x_1 x_2];$$
$$g : [[2x_1, 2], [x_2, 1]];$$
$$h : [x_1^2, x_1^2 + x_2^2];$$

6.2.1 User-defined functions:

A description of the user-defined functions available in CONDENS is given:

MENU: returns a list of all the user-defined functions contained in CONDENS.

HELP(fun-name): generates an information text describing the function, its syntax, how to enter its arguments as well as an example.

JACOB(f): computes the *Jacobian* of f. That is returns the matrix:

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

LIE(f,g): computes the *Lie brackets* of the vector fields f and g:

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

ADJ(f,g,k): computes the k $\overset{th}{-}$ adjoint of f and g:

$$\begin{aligned} ad_f^k g &= [f, ad_f^{k-1} g] \\ ad_f^0 g &= g \end{aligned}$$

LIDEV(f,h): computes the *Lie derivative* of the real valued function h along the direction defined by the vector field f: $\mathbf{L_f h}$.

NLIDEV(f,h,k): computes the k^{th} *Lie derivative* of h along f:

$$\begin{aligned} L_f^k h &= L_f \left(L_f^{k-1} h \right) \\ L_f^0 h &= h \end{aligned}$$

KRONECK(f,g): used for multivariable systems,where g represents the set of vector fields $g_1, \ldots, g_m$

This function is useful when the nonlinear system is transformable to a linear controllable system in Brunovsky canonical form. It computes the *Kronecker*

indices of the equivalent linear system the original nonlinear system is transformed to. It returns a set of numbers:

$$k_1 \geq k_2 \geq \cdots \geq k_m$$

$$k_1 + k_2 + \cdots + k_m = n$$

BTRIANG(f,g): This function checks if the nonlinear system $\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i$ is in *block triangular form* (see section 3.1 for definition). The argument g of the function represents the m vector fields $g_1, \ldots, g_m$.

RELORD(f,g,h): computes the *relative order* (see section 4.1.1 for definition) of the single-input single-output nonlinear system:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned}$$

6.2.2 TRANSFORM and INVERT

TRANSFORM and INVERT are two independent modules that use the user-defined functions presented in the previous subsection to study two theoretical problems: feedback linearization and invertibility of nonlinear control systems.

TRANSFORM (f,g):

TRANSFORM treats the feedback linearization problem presented in Chapter 3, that is, given the nonlinear control system:

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i \tag{6.1}$$

TRANSFORM investigates the existence of a one to one transformation (consisting of a change of coordinates and feedback) which transforms system (6.1) to a controllable linear system in canonical form. Moreover, in case the transformation, say T, exists, TRANSFORM generates the set of partial differential equations that T satisfies. In some cases, the module TRANSFORM can solve these partial differential equations and returns the transformation T and its inverse.

TRANSFORM proceeds in the following manner:

- Takes as input the nonlinear dynamics $f, g_1, \ldots, g_m, h_1, \ldots, h_m$ and the desired trajectory $y_d(t)$.

- Invstigates the left and right invertibility of the nonlinear system.

  Left-invertibility: by computing the relative order of the system and making sure that it is finite.

  Right-invertibility: by checking if the desired trajectory $y_d(t)$ is trackable by our system, that is, if $y_d(t)$ satisfies conditions

- Uses the package INVERT to construct the left-inverse to our system.

- Feeds the left-inverse system obtained with $y_d(t)$ to obtain, as output, the required control $u_d(t)$.

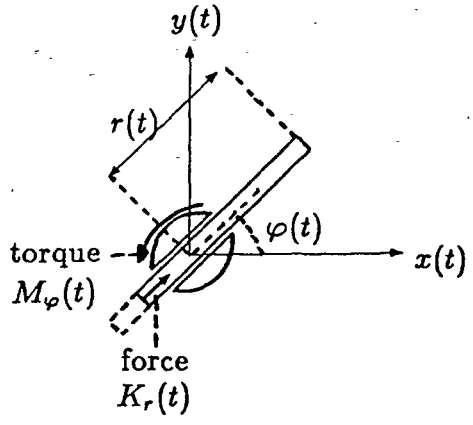
- Generates upon request a Fortran program for simulation purposes.

6.3 Examples

In this section, two examples are treated using CONDENS.

EXAMPLE 1: Basic industrial robot

This is the example of a *basic industrial robot.* It has one rotational joint and a translational joint in the (x, y) plane.



Using a polar coordinate system for modelling the robot, the kinetic equations are:

$$\begin{aligned}
\ddot{r} &= r\dot{\varphi}^2 - \frac{m_R l}{2(m_R+m_L)}\varphi^2 + \frac{K_r}{m_R+m_L} \\
\ddot{\varphi} &= \frac{-2(m_R-m_L)r+ml}{k-m_R l r+(m_R+m_L)r^2}\dot{r}\dot{\varphi} + \frac{M_\varphi}{k-m_R l r+(m_R+m_L)r^2} \\
y_1 &= r \\
y_2 &= \varphi
\end{aligned}$$

with $r(t)$ as translational motion and $\varphi(t)$ as the angle of rotation. y_1 and y_2 are the outputs that have to be controlled. $K_r(t)$ and M_φ are the drives corresponding to $r(t)$ and $\varphi(t)$.

If the state variables and the inputs are chosen to be:

$$\begin{aligned}
x_1(t) = r(t) \quad & x_2(t) = \dot{r}(t) \\
x_3(t) = \varphi(t) \quad & x_4(t) = \dot{\varphi}(t) \\
u_1(t) = K_r(t) \quad & u_2(t) = M_\varphi(t)
\end{aligned}$$

then the system can be described by a state space representation of the form:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ f_1(\mathbf{x}) \\ x_4 \\ f_2(\mathbf{x}) \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{1}{m_R+m_L} & 0 \\ 0 & 0 \\ 0 & \frac{1}{k-m_R l x_1+(m_R+m_L)x_1^2} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

where

$$\begin{aligned}
f_1(\mathbf{x}) &= x_1 x_4^2 - \frac{m_R l}{2(m_R+m_L)}x_4^2 \\
f_2(\mathbf{x}) &= \frac{-2[(m_R+m_L)x_1 - m_R\frac{l}{2}]}{k-m_R l x_1+(m_R+m_L)x_1^2}x_2 x_4
\end{aligned}$$

We are interested in designing a tracking controller to track the trajectories:

$$\begin{aligned}
y_1(t) &= x_1(t) = exp(t) \\
y_2(t) &= x_3(t) = t
\end{aligned}$$

Using the first technique: Exact Linearization.

with $n_k = m_L = l = 2, k = 5$

```
(c2) load("initfile.mac");
(d2)                          initfile.mac
(c3) fenols();
Hello,FENOLS tries to solve the problem:                                    #

        Given the non linear system:

                                  m
                                 ====
                          dx     \
                          -- =   >     u  g (x) + f(x)
                          dt     /      i  i
                                 ====
                                 i = 1

find a non singular transformation that takes this system to a controllable li#

near system:

                                  dz
                                  -- = b . v + a . z
                                  dt

enter dimension of the state space
4;

enter dimension of the control space
2;

enter the values of f in the form[f1(x),f2(x),....,fn(x)] followed by ,
[x2,x1*x4**2-x4**2/2,x4,(4*x2*x4-8*x1*x2*x4)/(4*x1**2-4*x1+5)];

enter g (x) in the form [ g     (x) .... g     (x) ]
       1                  1, 1           1, 4
[0,1,0,0];

enter g (x) in the form [ g     (x) .... g     (x) ]
       2                  1, 1           1, 4
[0,0,0,4/(4*x1**2-4*x1+5)];

Hello,TRANSFORM tries to solve the problem:                                 #

        Given the non linear system:

                          dx
                          -- = f(x) + u  g (x) + u  g (x)
                          dt           2  2       1  1

find a non singular transformation that takes this system to a controllable

linear system:

                                  dz
                                  -- = b . v + a . z
                                  dt

checking if the system is in block triangular form....

system in block triangular form

the transformation is easy to construct
```

The new state variables are:

```
z[ 1 ]= x1

z[ 2 ]= x3

z[ 3 ]= x2

z[ 4 ]= x4
```

The new control variables are:

```
                         2
                 2    x4
v[ 1 ]= x1 x4   -  ---  + u1
                       2

          4 x2 x4 - 8 x1 x2 x4              4 u2
v[ 2 ]= -------------------- + ----------------
              2                      2
          4 x1  - 4 x1 + 5      4 x1  - 4 x1 + 5
```

enter the desired trajectory in the form [xd(1),...xd(n)]
[exp(t),exp(t),t,1];

enter the desired eigenvalues of the linear controller in the form [delta(1),.#

..,delta(n)] followed by a ,
[-2,-2,-2,-2];

The tracking controller is :

```
                                 2                                t
                    (2 x1 - 1) x4  - 2 (- 4 x3 - 4 x1 + 5 %e  + 4 t)
u(x,xd)[ 1 ]= - ------------------------------------------------
                                          2

                                              2                                  t
u(x,xd)[ 2 ]= ((8 x1 x2 - 4 x2) x4 + 4 x1  (- 4 x4 - 4 x2 + 4 %e  + 5)

                                          t                                       t
       - 4 x1 (- 4 x4 - 4 x2 + 4 %e  + 5) + 5 (- 4 x4 - 4 x2 + 4 %e  + 5))/4
```

Are you interested in simulation results?(answer y or n)
y;

enter filename of fortran code(without adding'.f')
exampl;

enter initial time you would like the simulation to start from
0.0;

enter final time tf
5.0;

enter step size h
0.01;

enter initial condition in the form[xo[1],...xo[n]]
[0,0,0,0];

```
      dimension x( 4 ),dx( 4 ),datad(1000, 4 ),data(1000,
     1 4 ),u( 2 ),y( 4 )
c     set no of equations
      n= 4
      m= 2
c     set initial conditions
      x(1) = 0
      x(2) = 0
      x(3) = 0
      x(4) = 0
      y(1) = x(1)
      y(2) = x(2)
      y(3) = x(3)
      y(4) = x(4)
c     set initial & final time
      t= 0.0
      tf= 5.0
c     set step size
      h= 0.01
```

```
c     desired trajectory
      no= 500
      do 15 i=1,no
      datad(i,1) = exp(t)
      datad(i,2) = exp(t)
      datad(i,3) = t
      datad(i,4) = 1
      t=t+h
 15   continue
c     store initial values for plotting
      do 25 i=1, 4
      data(1,i)=y(i)
 25   continue
      t=0.0
      call control(x,t,u)
c     initialize k & mm
      k=0
      mm=1
      print *,t,datad(mm,1),data(mm,1)
c     write down the differential equations
  1   n= 4
      dx(1) = x(2)


      dx(2) = x(1)*x(4)**2-x(4)**2/2.0+u(1)
      dx(3) = x(4)
      dx(4) = (4*x(2)*x(4)-8*x(1)*x(2)*x(4))/(4*x(1)**2-4*x(1)+5)+4*u(2)
     1   /(4*x(1)**2-4*x(1)+5)
      call runta(n,k,ii,x,dx,t,h,u)
      go to (1,2),ii
  2   mm=mm+1
      y( 1 )= x(1)
      y( 2 )= x(2)
      y( 3 )= x(3)
      y( 4 )= x(4)
      do 30 i=1, 4
      data(mm,i)=y(i)
 30   continue
      print *,t,datad(mm,1),data(mm,1)
      if(t.le.tf) go to 1
      stop
      end

      subroutine runta(n,k,ii,x,dx,t,h,u)
      dimension yf( 4 ),z( 4 ),x( 4 ),dx( 4 ),u( 2 )
      k=k+1
      go to (1,2,3,4,5),k
  2   do 10 j=1,n
      z(j)=dx(j)
      yf(j)=x(j)
 10   x(j)=yf(j)+0.5*h*dx(j)
 25   t=t+0.5*h
      call control(x,t,u)
  1   ii=1
      return
  3   do 15 j=1,n
      z(j)=z(j)+2.0*dx(j)
 15   x(j)=yf(j)+0.5*h*dx(j)
      call control(x,t,u)
      ii=1
      return
  4   do 20 j=1,n
      z(j)=z(j)+2.0*dx(j)
 20   x(j)=yf(j)+h*dx(j)
      go to 25
  5   do 30 j=1,n
 30   x(j)=yf(j)+(z(j)+dx(j))*h/6.0
      call control(x,t,u)
      ii=2
      k=0
      return
      end

      subroutine control(x,t,u)
      dimension u( 2 ),x( 4 )
      u(1) = -((2*x(1)-1)*x(4)**2-2*(5*exp(t)+4*t-4*x(3)-4*x(1)))/2.0
      u(2) = (4*x(1)**2*(4*exp(t)-4*x(4)-4*x(2)+5)-4*x(1)*(4*exp(t)-4*x(
     1   4)-4*x(2)+5)+5*(4*exp(t)-4*x(4)-4*x(2)+5)+(8*x(1)*x(2)-4*x(2))*
     2   x(4))/4.0
      return
      end
```

(d3) done

Using the second technique: Inverse System.

```
(c2) load("initfile.mac");

(d2)                                initfile.mac

(c3) tracks();

Hello,TRACKS tries to solve the problem:                                    #

        Given the non linear system:

                                    m
                                   ====
                                   \
                            dx      >     u  g (x) + f(x)
                            -- =    /       i  i
                            dt     ====
                                   i = 1


                                 y(t)=h(x)

where      x is an n-dim vector
           u is an m-dim vector
           y is an m-dim vector

find the inverse system that takes as input the desired trajectory yd(t) and  #

generates the required control ud(t)

enter dimension of the state space
4;

enter dimension of the control space
2;

enter the values of f in the form [ f (x) ... f (x) ] followed by ,
                                    1          4
[x2,x1*x4**2-x4**2/2,x4,(4*x2*x4-8*x1*x2*x4)/(4*x1**2-4*x1+5)];

enter g (x) in the form [ g     (x) .... g     (x) ] followed by ,
       1                  1, 1           1, 4
[0,1,0,0];

enter g (x) in the form [ g     (x) .... g     (x) ] followed by ,
       2                  2, 1           2, 4
[0,0,0,4/(4*x1**2-4*x1+5)];

enter the values of h in the form [ h (x) ... h (x) ] followed by ,
                                    1          2
[x1,x3];

enter the initial state x0 in the form [ x0 (x) ... x0 (x) followed by ,
                                         1           4
[1,1,0,1];

enter the desired trajectory you want the output of your system to track



yd[ 1 ](t)=
exp(t);

yd[ 2 ](t)=
t;


relative order of system = 2

The inverse system to our non linear control system is :
```

$$\frac{dx}{dt} = binv(x) \cdot yi(t) + ainv(x)$$

$$u(t) = Cinv(x) + Dinv(x) \cdot yi(t)$$

where $yi(t) = \left[\frac{d^2 y1}{dt^2}, \frac{d^2 y2}{dt^2}\right]$

$$Ainv(x) = \begin{bmatrix} x2 \\ 0 \\ x4 \\ 0 \end{bmatrix}$$

$$Binv(x) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Cinv(x) = \begin{bmatrix} -\frac{2\,x1\,x4^2 - x4^2}{2} \\ 2\,x1\,x2\,x4 - x2\,x4 \end{bmatrix}$$

$$Dinv(x) = \begin{bmatrix} 1 & 0 \\ \frac{4\,x1^2 - 4\,x1 + 5}{4} & 0 \end{bmatrix}$$

Checking if yd(t) is trackable by our system (right-invertibility)

yd(t) can be tracked by the output y(t)

The control ud(t) that makes y(t) track yd(t) is:

$$ud(t)[1] = \left[-\frac{2\,x1\,x4^2 - x4^2 - 2\,\%e^t}{2},\ 2\,x1\,x2\,x4 - x2\,x4\right]$$

$$ud(t)[2] = \left[-\frac{2\,x1\,x4^2 - x4^2 - 2\,\%e^{-t}}{2},\ 2\,x1\,x2\,x4 - x2\,x4\right]$$

Are you interested in simulation results ? (answer y or n)
y;

enter filename of fortran code
fort;

enter initial time you would the simulation to start from
0.0;

enter final time tf
5.0;

enter step size h
0.05;

enter initial condition in the form [xo[1],...,xo[4]
[0,0,0,0];

```
      dimension x( 4 ),dx( 4 ),datad(1000, 2 ),data(1000,
     1 2 ),u( 2 ),y( 2 )
c     set no of equations
      n= 4
      m= 2
c     set initial conditions
      x(1) = 0
      x(2) = 0
      x(3) = 0
      x(4) = 0
      y(1) = x(1)
      y(2) = x(3)
c     set initial & final time
      t= 0.0
      tf= 5.0
c     set step size
      h= 0.05
c     desired trajectory
      no= 100
      do 15 i=1,no
      datad(i,1) = exp(t)
      datad(i,2) = t
      t=t+h
 15   continue

c     store initial values for plotting
      do 25 i=1, 2
      data(1,1)=y(i)
 25   continue
      t=0.0
      call control(x,t,u)
c     initialize k & mm
      k=0
      mm=1
      print *,t,datad(mm,1),data(mm,1)
c     write down the differential equations
 1    n= 4
      dx(1) = x(2)
      dx(2) = x(1)*x(4)**2-x(4)**2/2.0+u(1)
      dx(3) = x(4)
      dx(4) = (4*x(2)*x(4)-8*x(1)*x(2)*x(4))/(4*x(1)**2-4*x(1)+5)+4*u(2)
     1   /(4*x(1)**2-4*x(1)+5)
      call runta(n,k,ii,x,dx,t,h,u)
      go to (1,2),ii
 2    mm=mm+1
      y( 1 )= x(1)
      y( 2 )= x(3)
      do 30 i=1, 2
      data(mm,i)=y(i)
 30   continue
      print *,t,datad(mm,1),data(mm,1)
      if(t .le.tf) go to 1
      stop
      end

      subroutine runta(n,k,ii,x,dx,t,h,u)
      dimension yf( 4 ),z( 4 ),x( 4 ),dx( 4 ),u( 2 )
      k=k+1
      go to (1,2,3,4,5),k
 2    do 10 j=1,n
      z(j)=dx(j)
      yf(j)=x(j)
 10   x(j)=yf(j)+0.5*h*dx(j)
 25   t=t+0.5*h
      call control(x,t,u)
 1    ii=1
      return
 3    do 15 j=1,n
      z(j)=z(j)+2.0*dx(j)
 15   x(j)=yf(j)+0.5*h*dx(j)
      call control(x,t,u)
      ii=1
      return
 4    do 20 j=1,n
      z(j)=z(j)+2.0*dx(j)
 20   x(j)=yf(j)+h*dx(j)
      go to 25
 5    do 30 j=1,n
 30   x(j)=yf(j)+(z(j)+dx(j))*h/6.0
      call control(x,t,u)
      ii=2
      k=0
      return
      end

      subroutine control(x,t,u)
      dimension u( 2 ),x( 4 )
      u(1) = exp(t)-x(1)*x(4)**2+x(4)**2/2.0
      u(2) = -(4*x(2)*x(4)-8*x(1)*x(2)*x(4))/4.0
      return
      end
```

Example 2

Check if the following nonlinear system is feedback-linearizable to a controllable linear system and if so, find the F-transformation.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \end{pmatrix} = \begin{pmatrix} \sin(x_2) \\ \sin(x_3) \\ x_4^3 \\ x_5 + x_4^3 - x_1^{10} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

```
(c2) load("initfile.mac");
(d2)                                initfile.mac
(c3) f:[sin(x2),sin(x3),x4**3,x5+x4**3-x1**10,0];
                                                3          3     10
(d3)              [sin(x2), sin(x3), x4 , x5 + x4  - x1  , 0]
(c4) g:[[0,0,1,0,0],[0,0,0,0,1]];
(d4)                    [[0, 0, 1, 0, 0], [0, 0, 0, 0, 1]]
(c5) transform(f,g);

Hello,TRANSFORM tries to solve the problem:                                    #

          Given the non linear system:

                                dx
                                -- = f(x) + u  g (x) + u  g (x)
                                dt           2  2       1  1

find a non singular transformation that takes this system to a controllable

linear system:

                                dz
                                -- = b . v + a . z
                                dt

checking if the system is in block triangular form....
system not in block triangular form ==> trying the general method...
the system is multi-input ==> computing first the Kronecker indices of the equ#

ivalent controllable linear system

Kronecker indices are:
k[ 1 ]= 3
k[ 2 ]= 2
checking the first condition of transformability.....
checking the second condition of transformability...
checking the third condition of transformability....
```

```
span of c1 =span of c1 /C
span of c2 =span of c2 /C
All the conditions are satisfied
trying to construct (if possible) the transformation....
The new state variables are :
z[ 1 ]= x1
z[ 2 ]= sin(x2)
z[ 3 ]= cos(x2) sin(x3)
z[ 4 ]= - x4
                          3      10
z[ 5 ]= - x5 - x4  + x1
The new control variables are :
                                  3                    2
v[ 1 ]= cos(x2) cos(x3) (x4  + u1) - sin(x2) sin (x3)
                       2          3     10          9
v[ 2 ]= - 3 x4  (x5 + x4  - x1  ) + 10 x1  sin(x2) - u2
(d5)                                    done
```

Chapter 7

Conclusion

A rich collection of analytical tools based on differential geometric methods has been developed for the analysis and design of nonlinear control systems. These tools include: The *Hunt-Su-Meyer* technique for exact linearization of nonlinear control systems and the right and left invertibility of nonlinear sytems.

Using computer algebra programming methods, a software system, CONDENS, has been developed which make these analytical procedures available to users who need not have an extensive knowledge of differential geometry.

It should be pointed out that these methods are limited to a certain class of nonlinear systems; namely the class of systems which are either invertible or meet sufficient conditions of transformability. However, many important control problems do belong to this class (e.g, Robot manipulators, Power systems).

This work may be viewed as a component of an expert system for the treatment of nonlinear control problems. In particular, CONDENS can be extended to include some recent results on the analysis of nonlinear control systems such as:

- A global version of the exact linearization technique.
- Transformation of a nonlinear control system with output to a controllable and observable linear control system with linear output.
- Output tracking with singular points.

Appendix

```
/*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
 This is an init file for the package CONDENS:CONTROL DESIGN OF NONLINEAR
SYSTEMS. This is the only file that has to be loaded in MACSYMA by the user,
all the other files will be automatically loaded as they are needed.
 So to get started, the user just has to type once : LOAD("initfile.mac"),
followed by the command CONDENS() which will give him some hints on how to
use the package.
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*/

loadprint:false$
setup_autoload("CONDENS/menu.mac",menu)$
setup_autoload("CONDENS/geofun1.mac",jacob,pjacob,lie,plie,adj)$
setup_autoload("CONDENS/geofun2.mac",lidev,plidev,nlidev,relord)$
setup_autoload("CONDENS/transform.mac",transform)$
setup_autoload("CONDENS/fenols.mac",fenols)$
setup_autoload("CONDENS/tracks.mac",tracks)$
setup_autoload("CONDENS/invert.mac",invert)$
setup_autoload("CONDENS/btriang.mac",btriang)$
setup_autoload("CONDENS/simu.mac",ff)$
setup_autoload("CONDENS/condens.mac",condens)$
print_true:false$
help(arg):=arg(help)$



condens():=block(clearscreen(),
print("*********************************************************************"),
print("*                                                                   *"),
print("*  Hello ! WELCOME to CONDENS :  CONtrol DEsign of Nonlinear Systems *"),
print("*                                                                   *"),
print("*  a MACSYMA package for the design of controllers for the output   *"),
print("*  tracking problem using differential geometric concepts.          *"),
print("*                                                                   *"),
print("*      by:                  WASSIMA AKHRIF                          *"),
print("*                                                                   *"),
print("*      under the supervision of : Prof. GILMER BLANKENSHIP          *"),
print("*                                                                   *"),
print("*********************************************************************"),
print("        f,g and h,whenever mentioned, always designate the nonlinear "),
print("        dynamics of the nonlinear system affine in control:          "),
print("                  dx/dt = f(x) + g(x) u       "),
print("                      y  = h(x)               "),
print("  "),
print("        Type  MENU(), to get a list of all functions available in"),
print("        CONDENS.   "),
print("        "))$



menu():=block(
print("HELP(fun-name): generates an information text describing the function."),
print("JACOB(f):computes the Jacobian matrix of f."),
print("LIE(f,g): computes the Lie brackets of the vector fields f and g."),
print("ADJ(f,g,k): computes the k-th adjoint of f and g."),
print("LIDEV(f,h): computes the Lie derivative of the real valued function h"),
print("          along the direction defined by the vector field f."),
print("NLIDEV(f,h,k): computes the k-th Lie derivative of h along f."),
print("BTRIANG(f,g): checks if the nonlinear system is in block triangular form"),
print("RELORD(f,g,h): computes the relative order of the scalar control system"),
print("              dx/dt=f(x)+g(x)u, y=h(x).  "),
print("TRANSFORM(f,g): treats the feedback linearization problem,that is checks"),
print("                if the system is linearizable and solves for the nonlinear"
print("                transformation."),
print("FENOLS(): module that treats the output tracking problem using feedback"),
print("          linearization."),
print("INVERT(f,g,h): finds the left inverse to the original nonlinear system."),
print("TRACKS(): module that treats the output tracking problem using the left-"),
print("          inverse system."),done)$
```

```
/*************************************************************************/
/*                                                                       */
/*  This file contains 3 functions:                                      */
/*                                                                       */
/*      Jacob(f): computes the Jacobian of a vector function f.          */
/*                                                                       */
/*      Lie(f,g): computes the Lie brackets of f and g.                  */
/*                                dg       df                            */
/*                [f,g]=          --  .f - --  .g                       */
/*                                dx       dx                            */
/*                                                                       */
/*      Adj(f,g,n): computes the adjoint of f and g of order n.          */
/*                                                                       */
/*                (ad.n f,g)=[f,(ad.n-1 f,g)]                            */
/*                (ad.0 f,g)= g                                          */
/*                                                                       */
/*    CODE    ***************************************************************/
/*    ----                                                               */


jacob(arg1):=block([p,temp1,temp2,temp3],
            if arg1=help then return((
print(" jacob(f) : computes the Jacobian matrix of f.  "),
print("            f is entered in the form   f:[f1(x),...,fn(x)],     "),
print("            example:(if n=2)   f:[x1**2,x2],      "),done)),
           p:length(arg1),temp1:[],
           for i1:1 thru p do
           (temp2:makelist(diff(arg1[i1],concat('x,j1)),j1,1,p),
           temp1:endcons(temp2,temp1)),
           temp3:apply(matrix,temp1),
           'df/dx= temp3)$

pjacob(arg):=part(jacob(arg),2)$


lie(arg1,[arg2]):=block([tem,g1],
                 if arg1=help then return((
print(" lie(f,g) : computes the Lie brackets of the vector fields f and g "),
print("            f and g are entered in the form  f:[f1(x),...,fn(x)],  "),
print("                                             g:[g1(x),...,gn(x)],  "),
print("            Example:(if n=2)   f:[x1,x2-1],  g:[x1**2,x2**2],"),done)),
                gl:part(arg2,1),
                tem:(pjacob(gl).transpose(arg1)-pjacob(arg1).transpose(gl)),
                tem:makelist(tem[i2,1],i2,1,length(arg1)),
                '[f,g]= tem)$


plie(arg1,arg2):=part(lie(arg1,arg2),2)$

adj(arg1,[arg2]):=block([tem1,g1,n1],
                if arg1=help then return((
print(" adj(f,g,k) :  computes the k-th adjoint of f and g "),
print("               f and g are entered in the form  f:[f1(x),...,fn(x)],"),
print(
```

```
/*************************************************************************/
/*                                                                       */
/*   This file contains 3 functions:                                     */
/*                                                                       */
/*       Lidev(f,h) : computes the Lie derivative of the real valued     */
/*                    function h along the direction defined by the      */
/*                    vector field f :                                   */
/*                                                                       */
/*                          Lf(h) = < dh,f >                             */
/*                                                                       */
/*       Nlidev(f,h,k) : computes the k-th order derivative of h along f: */
/*                          k       k-1                                  */
/*                        Lf(h) = Lf(Lf(h))                              */
/*                          0                                            */
/*                        Lf(h) = h                                      */
/*                                                                       */
/*       Relord(f,g,h) : computes the relative order of the scalar       */
/*                       nonlinear system :                              */
/*                                          x = f(x) + g(x) u            */
/*                                          y = h(x)                     */
/*                                                                       */
/*************************************************************************/


lidev(arg1,[arg2]):=block([x,res,hl],
                    if arg1=help then return((
print(" lidev(f,h) : computes the Lie derivative of h along the direction f"),
print("       h : scalar real valued function, entered in the form h:h(x),"),
print("       Example:  h:x1+x2,    "),
print("       f : vector field entered in the form f:[f1(x),...,fn(x)], "),
print("       Example: (if n=2) f:[x1**2,x1+x2],              "),done)),
                   hl:part(arg2,1),
                   x:makelist(concat(x,i),i,1,length(arg1)),
                   res:sum(diff(hl,x[i])*arg1[i],i,1,length(arg1)),
                   'Lf('h)= res)$


plidev(arg1,arg2):=part(lidev(arg1,arg2),2)$

nlidev(arg1,[arg2]):=block([tem,hl,nl],
                     if arg1=help then return((
print(" nlidev(f,h,n) : computes the n-th order Lie derivative of h along the dire
print("                 h : scalar real valued function, entered in the form h:h(x
print("                 f : vector field entered in the form f:[f1(x),f2(x), ...,f
print("                 Example: (if n=2) f:[x1**2,x1+x2],    "),done)),
                    hl:part(arg2,1),
                    nl:part(arg2,2),
                    tem:hl,
                    if nl#0 then   for i:1 thru nl do
                    tem:plidev(arg1,tem),
                    'Lf[nl]('h)= tem)$


pnlidev(ar1,ar2,ar3):=part(nlidev(ar1,ar2,ar3),2)$

gfh(f,g,h,n):=plidev(g,pnlidev(f,h,n))$



relord(arg1,[arg2]):=block([gl,hl,nl],
                    if arg1=help then return((
print(" relord(f,g,h) : computes the relative order of the scalar nonlinear "),
print("                 system:   dx/dt = f(x) + g(x) u                    "),
print("                             y   = h(x)                             "),
print("                 f is entered in the form   f:[f1(x),....,fn(x)],"),
print("                 example:(if n=2)        f:[x1,x2**2],"),
print("                 g is entered in the form   g:[g1(x),....,gn(x)],"),
print("                 example:(if n=2)        g:[x2,x1*x2], "),
print("                 h is entered in the form - h:h(x),"),
print("                 example:                h:x1+x2, "),done)),
                   gl:part(arg2,1),
                   hl:part(arg2,2),
                   nl:0,lab,
                   if gfh(arg1,gl,hl,nl)#0 then
                   (print("relative order of system =",nl+1),done)
                   else (nl:nl+1,go(lab)))$
```

```
/*****************************************************************************/
/*                                                                           */
/*     This function checks if the system is in block triangular form        */
/*                                                                           */
/*****************************************************************************/


btriang(argl, [arg2]):=block([x,xl,u,ind],
                      if argl=help then return((
print("Btriang(f,g) : checks if the multivarible system dx/dt=f(x)+g(x)u is "),
print("               in block triangular form.  "),
print("               f and g are entered in the form f:[f1(x),...,fn(x)],   "),
print("                                               g:[[g1(x)],..[gm(x)]], "),
print("               Example:(if n=2,m=2)            f:[x1,x2**2],          "),
print("                                               g:[[x1,1],[x2,x1]],  ")
,done)),

                      f:argl,g:part(arg2,1),n:length(f),m:length(g),
                      x:makelist(concat('x,i),i,1,n),
                      ind:makelist(concat('ind,i),i,1,n),
                      ind[1]:0,d:1,p1:1,it:n,
                      u:makelist(concat('u,i),i,1,m),
                      xl:makelist([],i,1,n),xl[p1]:u,
                      for i:1 thru n do
                      equ[i]:f[i]+sum(g[j][i]*u[j],j,1,m),
                      for p:1 thru n/m do(l:0,los:0,
                          for i:n thru 1 step -1 do (
                      for id :1 thru d do
                      if i=ind[id] then (if i>1 then i:i-1
                      else (los:1,id:d)),
                      if los #1 then for r:1 thru length(xl[p]) do
                            if equ[i]#subst(0,xl[p][r],equ[i]) then (
                            l:l+1,cx[it]:x[i],it:it-1,
                            xl[p+1]:endcons(x[i],xl[p+1]),
                            ind[d]:i,if d=n then (i:1,p:n/m),d:d+1,
                            r:length(xl[p]))),
                      if l#m then
                      (print("system is not in block triangular form"),
                      resul:0,p:n/m)
                      else resul:1 ),if resul=1 then
                      print("system in block triangular form"),done)$
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*        TRANSFORM tries to solve the problem:                              */
/*                                                                           */
/*        Given the non linear control system:                               */
/*                                        i=m                                */
/*                                        ---                                */
/*                                        \                                  */
/*                    dx                  /                                  */
/*                    -- =   F(x) +       /    Gi(x) . u(i)                  */
/*                    dt                  ----                               */
/*                                        i=1                                */
/*                                                                           */
/*        Find a non singular transformation that takes this system          */
/*                                                                           */
/*        to a controllable linear system :                                  */
/*                                                                           */
/*                    dz                                                     */
/*                    -- =    A z  +  B v                                    */
/*                    dt                                                     */
/*                                                                           */
/*****************************************************************************/
```

```
transform(arg1,[arg2]):=block([cx,resul],
if arg1=help then return((
print("Transform(f,g): given the nonlinear system dx/dt = f(x)+g(x)u,this "),
print("                function checks if this system is transformable, via"),
print("                a nonlinear change of coordinates and feedback, to a "),
print("                controllable linear system in canonical form. If so, "),
print("                Transform tries to solve for the transformation and its"),

print("                inverse."),
print("                f,g are entered in a list format.One must be careful"),
print("                in entering g ."),
print("Example: if (n=2,m=2)"),
print("                      dx1/dt = x1 + x2*u1 + u2    "),
print("                      dx2/dt = x2+x1 + x1*u2    "),
print(" f:[x1,x1+x2],     g:[ [x2,0] , [1,x1] ],"),done)),
f:arg1,g:part(arg2,1),
n:length(f),m:length(g),

print("Hello,TRANSFORM tries to solve the problem:
display('diff(x,t)='f(x)+sum('g[i](x)*u[i],i,1,m)),
 print("find a non singular transformation that takes this system to a controllabl
linear system:"),
display('diff(z,t)= A.z + B.'v),
n:length(f),
m:length(g),
print("checking if the system is in block triangular form...."),
cx:makelist(concat('cx,i),i,1,n),
if integerp(n/m)=true then checkform(f,g),
for i:1 thru m
do k[i]:n/m ,if resul=1 then (print("the transformation is easy to construct"),

feed(f,g), return(bye))
        else print("system not in block triangular form ==> trying the general
if m=1 then (k[1]:n,
print("system is single-input ==> trying first a more restrictive type of
feedback v=u+a(x) [Brockett] since pde's easier to solve..."),
print("checking if the system is transformable..."),
print("checking the first condition of transformability..."),
indep(f,g),
if resul=1 then (print("system not transformable"),return(bye))
        else print("checking the second condition of transformability..."),
invol(f,g,1),
if resul=1 then (print("the feedback v=u+a(x) is too restrictive,trying the
more general feedback a(x)+b(x)u..."),kill(resul))
        else (print("system is transformable"),
print("trying to construct(if possible)the transformation..."),
broc(f,g),return(bye))),
      if m>1 then (print("the system is multi-input ==> computing first the Krone
kronecker(f,g)),
print("checking the first condition of transformability....."),
indep(f,g),
if resul=1 then (print("system not transformable"),return(bye))
        else print("checking the second condition of transformability..."),
invol(f,g,2),
if resul=1 then (print("system not transformable"),return(bye))
        else if m>1 then(print("checking the third condition of transformabilit
span(f,g),
if resul=1 then (print("system not transformable"),return(bye))),
print("All the conditions are satisfied"),
print("trying to construct (if possible) the transformation...."),

kcy(f,g))$
```

```
/*************************************************************************/
/*                                                                       */
/*                      KRONECKER indices                                */
/*                      -----------------                                */
/*                                                                       */
/*     This function computes the kronecker indices of the equivalent    */
/*     controllable linear system in Brunovsky canonical form.           */
/*                                                                       */
/*                 K1 >= K2 >=..... >= Km                                */
/*                 K1 + K2 + K3 +.....+ Km = n                           */
/*                                                                       */
/**** CODE ****************************************************************/
/*    ----                                                               */

 kronecker(f,g):=block([dep1,dep2,a,dep,r],dep1:[],n:length(f),
                       m:length(g),
                       for j:1 thru m do
                       (dep2[j]:g[j],dep1:endcons(dep2[j],dep1)),
                       dep[0]:apply(matrix,dep1),a[0]:rank(dep[0]),
                       r[0]:a[0],
                       if n>1 then for i:1 thru n-1 do ( for j:1 thru m do (
                       dep2[j]:plie(f,dep2[j]),dep1:endcons(dep2[j],dep1)),
                       dep[i]:apply(matrix,dep1),a[i]:rank(dep[i]),
                       r[i]:a[i]-a[i-1]),for l:1 thru m do (k[l]:0,
                       for j:0 thru n-1 do if r[j]>=l then k[l]:k[l]+1
                       else k[l]:k[l]),print("Kronecker indices are:"),
                       for i:1 thru m do print("k[",i,"]=",k[i]))$


/*************************************************************************/
/*                                                                       */
/*     First Nec & Suff. condition of existence of the transformation T. */
/*     ----------------------------------------------------------------- */
/*                                                                       */
/*   The set C={g1,[f,g1],.....,(ad.k1-1 f,g1),g2,[f,g2],...(ad.k2-1 f,g2)*/
/*             .......gm,[f,gm],.......,(ad.km-1 f,gm)} spans an n-dim   */
/*             space.                                                    */
/*                                                                       */
/** CODE ******************************************************************/
/*  ----                                                                 */

indep(f,g):=block([depen1,depen2,h],depen2:[],for i4:1 thru m  do(
                  depen1:g[i4],depen2:endcons(depen1,depen2),
                  if k[i4]>1 then for j4:1 thru k[i4]-1 do
                  (depen1:plie(f,depen1),
                  depen2:endcons(depen1,depen2))),
                  h:apply(matrix,depen2),if rank(h)=0 then("dependent vectors",
                  resul:1)
                  else"The set C spans an n-dim space")$


/*************************************************************************/
/*                                                                       */
/*   Second condition                                                    */
/*   ----------------                                                    */
/*                                                                       */
/*   The sets Cj={g1,[f,g1],..,(ad.kj-2 f,g1),g2,[f,g2],..,(ad.kj-2 f,g2)*/
/*               ...,gm,[f,gm],.....,(ad.kj-2 f,gm)}                     */
/*               are involutive for j=1,2.....m.                         */
/*                                                                       */
/*** CODE *****************************************************************/
/*   ----                                                                */

invol(f,g,w):=block([dep1,dep2,dep3,dep4,l,m1,r1,r2],dep3:[],for d:1 thru m do
                    (dep1:[],for i5:1 thru m do(
                    dep2:g[i5],dep1:endcons(dep2,dep1),
                    if k[d]>w then for j5:1 thru k[d]-w do
                    (dep2:plie(f,dep2),dep1:endcons(dep2,dep1))),
                    l:apply(matrix,dep1),if (m*(k[d]-1)+1)>n
                    then dep4:[concat(c,d),"involutive"]
                    else for j6:1 thru m*(k[d]-1)-1 do
                    for j7:j6+1 thru m*(k[d]-1) do
                    (r1:makelist(l[j6,t],t,1,n),
                     r2:makelist(l[j7,t],t,1,n),
                    m1:addrow(l,plie(r1,r2)),
                    if rank(m1)<(m*(k[d]-1)+1) then
                    dep4:[concat(c,d),"is involutive"]
                    else(dep4:[concat(c,d),"not invol"],resul:1)),
                    dep3:endcons(dep4,dep3)),dep3)$
```

```
/*****************************************************************************/
/*                                                                           */
/*    Third condition                                                        */
/*    ---------------                                                        */
/*                                                                           */
/*   The span of each Cj is equal to the span of Cj /\ C .                   */
/*                                                                           */
/*** CODE ********************************************************************/
/*   -----                                                                   */

span (f,g):=block([dep1,dep2,dep3,dep4,q1,q2],
                  for d:1 thru m do (
                  dep1:[],dep3:[],for i:1 thru m do (
                  dep2:g[i],dep1:endcons(dep2,dep1),
                  mi:min(k[i]-1,k[d]-2),
                  if mi>=1 then for j:1 thru mi do(
                  dep2:plie(f,dep2),dep1:endcons(dep2,dep1)),
                  dep4:g[i],dep3:endcons(dep4,dep3),
                  if k[d]-2 >= 1 then for r:1 thru (k[d]-2) do
                  (dep4:plie(f,dep4),
                  dep3:endcons(dep4,dep3))),q1:apply(matrix,dep1),
                  q2:apply(matrix,dep3),if rank(q1)=rank(q2)then
                  print("span of",concat(C,d),"=span of",concat(C,d),
                  "/\C") else(print("ko"),resul:1)))$



/*****************************************************************************/
/*                                                                           */
/*     This function checks if the system is in block triangular form        */
/*                                                                           */
/*****************************************************************************/

checkform(f,g):=block([x,xl,u,ind,d,p,it,l],x:makelist(concat('x,i),i,1,n),
                      ind:makelist(concat('ind,i),i,1,n),
                      ind[1]:0,d:1,p1:1,it:n,
                      u:makelist(concat('u,i),i,1,m),
                      xl:makelist([],i,1,n),xl[p1]:u,
                      for i:1 thru n do
                      equ[i]:f[i]+sum(g[j][i]*u[j],j,1,m),
                      for p:1 thru n/m do(l:0,los:0,
                      for i:n thru 1 step -1 do (
                      for id :1 thru d do
                      if i=ind[id] then (if i>1 then i:i-1
                      else (id:d,los:1)),
                      if los#1 then for r:1 thru length(xl[p]) do
                      if equ[i]#subst(0,xl[p][r],equ[i]) then (
                      l:l+1,cx[it]:x[i],it:it-1,xl[p+1]:endcons(x[i],xl[p+1]),
                      ind[d]:i,if d=n then (i:1,p:n/m),d:d+1,
                      r:length(xl[p]))),
                      if l#m then (resul:0,
                      print("system is not in block triangular form"),
                      p:n/m)
                      else resul:1 ),if resul=1 then
                      print("system in block triangular form"),done)$



/*****************************************************************************/
/*                                                                           */
/*       This block constructs the transformation when the system is in      */
/*       block triangular form.                                              */
/*                                                                           */
/*****************************************************************************/


feed(f,g):=block([x,u,eqa],x:makelist(concat('x,i),i,1,n),
                  u:makelist(concat('u,i),i,1,m),
                  zr:makelist(concat('zr,i),i,1,n),
                  vr:makelist(concat('vr,i),i,1,m),
                  for i:1 thru m do zr[i]:cx[i],
                  for i:m+1 thru n do
                  zr[i]:sum(diff(zr[i-m],x[j])*equ[j],j,1,n),
                  for i:1 thru m do
                  vr[i]:sum(diff(zr[n-m+i],x[j])*equ[j],j,1,n),
                  print("The new state variables are:"),
                  for i:1 thru n do print("z[",i,"]=",zr[i]),
                  print ("The new control variables are:"),
                  for i:1 thru m do print("v[",i,"]=",vr[i]),
                  eqa:makelist(concat('v,i)=vr[i],i,1,m),
                  for i:1 thru m do dd2[i]:rhs(solve(eqa,u)[1][i]))$
```

```
/*************************************************************************/
/*                                                                       */
/*          This block constructs the transformation in the single       */
/*          input case with the feedback restriction:                    */
/*                                                                       */
/*                    v = u + h(x)                                       */
/*                                                                       */
/*************************************************************************/

broc(f,g):=block([x,v,s,a,b,T,u,eqa],x:makelist(concat('x,i),i,1,n),
                 u:[concat('u,1)],
                 zr:makelist(concat('z,i),i,1,n),dep:g[1],
                 for j:1 thru n do s[1,j]:dep[j],
                 if n>=2 then for i:2 thru n do ( dep:plie(f,dep),for j:1
                 thru n do
                 s[i,j]:dep[j]),a:genmatrix(s,n,n),
                 b:a^^-1,for i:1 thru n do T[i]:b[i,n],
                 zr[1]:integrate(T[1],x[1],0,x[1]),for i:2 thru n do (
                       for j:1 thru i-1 do T[i]:subst(0,x[j],T[i]),
                       zr[1]:zr[1]+integrate(T[i],x[i],0,x[i])),
                 for i:2 thru n do zr[i]:makelist(diff(zr[i-1],x[j]),j,1,n).
                                        transpose(f+g[1]*u[1]),
                 vr[1]:makelist(diff(zr[1][n],x[j]),j,1,n).transpose
                 (f+g[1]*u[1]),
                 print("The new state variables are:"),
                 for i:1 thru n do print ("z[",i,"]=",zr[i]),
                 print("The new control variables are:"),
                 for i:1 thru m do print ("v[",i,"]=",vr[i]),
                 eqa:concat('v,1)=vr[1],
                 dd2[1]:rhs(solve(eqa,u[1])[1]))$



/*************************************************************************/
/*                                                                       */
/*              This block constucts the transformation in the general   */
/*                                                                       */
/*              case when the feedback is of the form:                   */
/*                                                                       */
/*                              v = a(x) + b(x) * u                      */
/*                                                                       */
/*************************************************************************/


cyr(f,g):=block([x,u,s,l,p,mit,sk,xoi,ti,equ,xi,y,b,al,c,trl,sigma,fx],
                 for j:1 thru m do( s[1,j]:g[j],if k[1] >= 2 then
                 for i:2 thru k[1] do if i<= k[j] then
                 s[i,j]:plie(f,s[i-1,j])  else s[i,j]:0),
                 l:k[1],p:1,for j:1 thru n do(
                 if s[l,p]=0 then (l:l-1,p:1),for i:1 thru n do
                 mit[i,j]:s[l,p][i],if l=k[p] then sk[p]:j,
                 if p<m then p:p+1 else (l:l-1,p:1)),
                 xoi:makelist(0,i,1,n),
                 ti:makelist(concat('t,i),i,1,n),
                 equ:makelist(concat('eq,i),i,1,n),
                 xi:makelist(concat('x,i),i,1,n),
                 for kl:1 thru n do(y:makelist(xi[j](ti[kl]),j,1,n),
                 for i:1 thru n do(for r:1 thru n do mit[i,kl]:subst(y[r],xi[r],
                 mit[i,kl]),
                 equ[i]:'diff(y[i],ti[kl])=mit[i,kl],
                 atvalue(y[i],ti[kl]=0,xoi[i])),
                 B:desolve(equ,y),Al:solve(B,y),C:ev(Al[1],laplace,ilt),
                 for i:1 thru n do xoi[i]:rhs(C[i])),
                 for i:1 thru n do C:subst(xi[i],y[i],C),
                 trl:solve(C,ti)[1], sigma[1]:1,
                 for i:2 thru m do sigma[i]:sum(k[j],j,1,i-1)+1,
                 zr:makelist(concat('zr,i),i,1,n),
                 u:makelist(concat('u,i),i,1,m),
                 vr:makelist(concat('vr,i),i,1,m),
                 fx:f+sum(g[i]*u[i],i,1,m),
                 for j:1 thru m do ( zr[sigma[j]]:rhs(trl[sk[j]]),
                 for i:1 thru k[j]-1 do
                 zr[sigma[j]+i]:plidev(fx,zr[sigma[j]+i-1])),
                 for j:1 thru m do vr[j]:plidev(fx,zr[k[j]+sigma[j]-1]),
                 eqa:makelist(concat(v,i)=vr[i],i,1,m),
                 for i:1 thru m do dd2[i]:rhs(solve(eqa,u)[1][i]),
                 print("The new state variables are :"),
                 for i:1 thru n do print("z[",i,"]=",zr[i]),
                 print("The new control variables are :"),
                 for i:1 thru m do print("v[",i,"]=",vr[i]))$
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*        FENOLS tries to solve the problem:                                 */
/*                                                                           */
/*        Given the non linear control system:                               */
/*                                        i=m                                */
/*                                        ---                                */
/*                                       \                                   */
/*                  dx                   /                                   */
/*                  --  =   F(x) +      /     Gi(x) . u(i)                   */
/*                  dt                   ----                                */
/*                                       i=1                                 */
/*                                                                           */
/*        Find the tracking control ud(t) that forces x(t) to follow a       */
/*                                                                           */
/*        desired path xd(t). FENOLS accomplishes this by transforming       */
/*                                                                           */
/*        the nonlinear system to a controllable linear one, using the       */
/*                                                                           */
/*        function TRANSFORM(f,g).                                           */
/*                                                                           */
/*****************************************************************************/
```

```
fenols([arg]):=block([zr,zd,v,vr,kf,sig,k,dd2,pol],
if length(arg)=0 then go(lsl),
            if part(arg,1)=help then return((
print("fenols() :given the multivariable nonlinear system dx/dt=f(x)+g(x)u "),
print("          this function finds the tracking control that makes the state"),
print("          x(t) follow some desired path xd(t).Fenols() accomplishes this"),
print("          by transforming first the nonlinear system to an equivalent"),
print("          controllable linear system in Brunovsky canonical form."),
print("          The module asks progressively for the data as it needs it.")
,done)),
lsl,
             print("Hello,FENOLS tries to solve the problem:
             display('diff(x,t)='f(x)+sum('g[i](x)*u[i],i,1,m)),
             print("find a non singular transformation that takes this system to
display('diff(z,t)= 'A.'z + 'B.'v),
n:read("enter dimension of the state space"),
m:read("enter dimension of the control space"),
f:read("enter the values of f in the form[f1(x),f2(x),....,fn(x)] followed by ,"),
for i:1 thru m do
gi[i]:read("enter",'g[i](x),"in the form [",'g[1,1](x),"....",'g[1,n](x),"]"),
g:makelist(gi[i],i,1,m),
transform(f,g),
xd:read("enter the desired trajectory"),
delta:read("enter the desired eigenvalues of the linear controller in the form [de
zd:zr,
for i:1 thru n do
zd:subst(xd[i],concat('x,i),zd),
/* construction of linear control */
sig[0]:0,for i:1 thru m do (sig[i]:sum(k[j],j,1,i),
pol[i]:expand(prod((s1-delta[j]),j,sig[i-1]+1,sig[i]))),

kf:zeromatrix(m,n),
for j:1 thru m do
    for i1:sig[j-1]+1 thru sig[j] do
kf[j,i1]:-coeff(pol[j],s1,i1-sig[j-1]-1),
for j:1 thru m do wd[j]:diff(zd[j],t)-sum(kf[j,i]*zd[i],i,sig[j-1]+1,sig[j]),
v:makelist(concat('v,i),i,1,m),
for i:1 thru m do
v[i]::sum(kf[i,j]*zr[j],j,1,n)+wd[i],
for i:1 thru m do dd2[i]:ev(dd2[i])
print("The tracking controller is :"),
for i:1 thru m do print("u(x,xd)[",i,"]=",dd2[i]),
ans:read("Are you interested in simulation results?(answer y or n)"),
if ans=y then(file:read("enter filename of fortran code(without adding'.f')"),
t1:read("enter initial time you would like the simulation to start from"),
tf:read("enter final time tf"),
hi:read("enter step size h"),
xo:read("enter initial condition in the form[xo[1],...xo[n]]"),
f1:f,g1:g,h1:makelist(x(i),i,1,n),m2:n,
for i:1 thru n do (
f1:subst(x(i),concat('x,i),f1),
g1:subst(x(i),concat('x,i),g1)),
yd:xd,
ff(file)),
return("bye,have a good day")
)$
```

```
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*         INVERT tries to solve the problem:                                */
/*                                                                           */
/*         Given the non linear control system:                              */
/*                                        i=m                                */
/*                                        ---                                */
/*                                       \                                   */
/*                    dx                 /                                  */
/*                    --  =   F(x) +    /     Gi(x) . u(i)                  */
/*                    dt                 ----                               */
/*            (*)                        i=1                                */
/*                    y(t) = h(x)                                            */
/*                                                                           */
/*         where     x  is  n-dimensional                                    */
/*                   u  is  m-dimensional                                    */
/*                   y  is  m-dimensional                                    */
/*                                                                           */
/*          Find the left-inverse to (*), i.e, a system having as input      */
/*        y(t) (or more precisely an appropriate derivative of y(t)), and    */
/*        as output u(t).                                                    */
/*                                                                           */
/*****************************************************************************/



invert(arg1,[arg2]):=block([x,n,m,alpha],
                    if arg1=help then return((
print("invert(f,g,h) : finds the left-inverse to the multivariable nonlinear"),
print("                control system: dx/dt=f(x)+g(x)u, y=h(x).        "),
print("                f,g,h are entered in list format. One must be careful "),

print("                in entering g.   "),
print("Example: (if n=2,m=2) "),
print("                     dx1/dt = x1**2 + u1 + x1*u2  "),
print("                     dx2/dt = x1**2*u1 + u2       "),
print("                        y1  = x2  "),
print("                        y2  = x1**3"),
print("then,  f:[x1**2,0], g:[ [1,x1**2] , [x1,1] ], h:[x2,x1**3], "),done)),
f:arg1,g:part(arg2,1),h:part(arg2,2),
n:length(f),
m:length(h),
x:makelist(concat('x,i),i,1,n),
y:makelist(concat('y,i),i,1,m),depends(y,t),


/* initialisation */
/*****************************************************************************/

k:0,q[k]:0,yi[k]:y,ci[k]:h,for i:1 thru m do for j:1 thru m do dinit[i,j]:0,
di[k]:genmatrix(dinit,m,m),lal,,
if q[k]#0 then

for i:1 thru q[k] do (   myi[i]:yi[k][i],
                         mci[i]:ci[k][i],
                         for j:1 thru m do mdi[i,j]:di[k][i,j]),
for j1:1 thru m do
for i:q[k]+1 thru m do ( myi[i]:diff(yi[k][i],t),
                         mci[i]:plidev(f,makelist(ci[k][j],j,1+q[k],m))[i],
                   mdi[i,j1]:plidev(g[j1],makelist(ci[k][j],j,1+q[k],m))[i]),
                   mmdi:genmatrix(mdi,m,m),


/* checking ranks */
/*****************************************************************************/

alpha:makelist(concat('alpha,i),i,1,m),
eq:makelist(sum(alpha[j]*mdi[j,i],j,1,m),i,1,m),
alpha1:solve(eq,alpha)[1],
alpha2:alpha1,
for i:1 thru n do alpha2:subst(0,x[i],alpha2),
if alpha2 # alpha1 then w:1
                  else (     q[k+1]:rank(mmdi),
                             if q[k+1]=m then (msa:ident(m),msb:ident(m),w:2),

                          if q[k+1]=0 then(msa:ident(m),msb:ident(m),w:3),
  if (q[k+1]#m and q[k+1]#0) then  (  /* construction of sk+1 */
for i:1 thru m  do for j:1 thru m do sb[i,j]:0,
if q[k]#0 then for i:1 thru q[k] do sb[i,i]:1,
for i:q[k]+1 thru q[k+1] do sb[i,ind(m,q[k+1],mmdi)[i]]:1,
for i:q[k+1]+1 thru m do  for j:q[k]+1 thru m do
if sum(sb[l,j],l,q[k]+1,i-1)=0 then (sb[i,j]:1,j:m),
msb:genmatrix(sb,m,m),
msa:ident(m),sbd:msb.mmdi,nl:genmatrix('nl,m-q[k+1],q[k+1]), mmdb:sbd,
mmdt:sbd,
for i:q[k+1]+1 thru m do mmdb:submatrix(i,mmdb),
for i:1 thru q[k+1] do mmdt:submatrix(i,mmdt),tem:[],
```

```
eql:nl.mmdb+mmdt,for i:1 thru m-q[k+1] do tem:append(tem,eql[i]),
coef:[],for i:1 thru m-q[k+1] do coef:append(coef,nl[i]),
coef2:solve(tem,coef),for i:1 thru m-q[k+1] do
                      for j:1 thru q[k+1] do
                      msa[i+q[k+1],j]:coef2[j+(i-1)*q[k+1]]),
 k:k+1,yi[k]:msa.msb.transpose(makelist(myi[i],i,1,m)),
       yi[k]:makelist(yi[k][i][1],i,1,m),
       ci[k]:msa.msb.transpose(makelist(mci[i],i,1,m)),
       ci[k]:makelist(ci[k][i][1],i,1,m),
       di[k]:msa.msb.mmdi    ),
if w=1 then return(print("impossible, the system is not invertible")),
if w#2 then go(la1)   ,
print("relative order of system =",k),

/* construct the inverse system */
/*****************************************************************************/
d:di[k]^^-1,d2:yi[k]-ci[k],
for j:1 thru m do
                        for i:1 thru n do
                              tg[i,j]:g[j][i],matg:genmatrix(tg,n,m),
                   Ainv1 : ratsimp(transpose(f)-matg.d.ci[k]),
                   Binv1 : ratsimp(matg.d),
                   Cinv1 : ratsimp(-d.ci[k]),Dinv1 : ratsimp(d),
print("The inverse system to our non linear control system is :"),
print("         "),
display(     'diff('x,t) = Ainv('x) + Binv('x) . yi(t)),
print("      "),
print("            u(t) = Cinv(x) + Dinv(x) . yi(t)"),
print("      "),
print("where      yi(t) =",yi[k]),
print(" Ainv(x) =",Ainv1),
print(" Binv(x) =",Binv1),
print(" Cinv(x) =",Cinv1),
print(" Dinv(x) =",Dinv1),done)$


ind(m,q,d):=block([a],a:makelist(concat(a,i),i,1,q),ul:0,
                for i:1 thru q do a[i]:i,11,
                for i:1 thru q do
                    for j:1 thru m do b[i,j]:d[a[i],j],
                c:genmatrix(b,q,m),
                if rank(c)=q then  ul:1
                              else (  ii:q,jj:0,for i:q thru 1 step -1 do
                                      if a[ii]=m-jj then (ii:ii-1,jj:jj+1)
                                                    else i:1,
                                       if ii=q
                      then a[ii]:a[ii]+1
                      else (a[ii]:a[ii]+1,for i:ii+1 thru q do
                      a[i]:a[i-1]+1)),if ul=1 then go(12),go(11),12,a)$


/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*        TRACKS tries to solve the problem:                                 */
/*                                                                           */
/*        Given the non linear control system:                               */
/*                                      i=m                                  */
/*                                      ---                                  */
/*                                      \                                    */
/*                   dx                 /                                   */
/*                   --  =   F(x) +     /    Gi(x) . u(i)                   */
/*                   dt                 ----                                 */
/*           (*)                        i=1                                  */
/*                   y(t) = h(x)                                             */
/*                                                                           */
/*        where    x  is  n-dimensional                                      */
/*                 u  is  m-dimensional                                      */
/*                 y  is  m-dimensional                                      */
/*                                                                           */
/*   Suppose we want the output y(t) to track a desired trajectory yd(t),    */
/*                                                                           */
/*   using the inverse system generated by INVERT,TRACKS generates the       */
/*                                                                           */
/*   required control ud(t) and if we are interested in simulation results, */
/*                                                                           */
/*   TRACKS will generate a fortran code for that purpose.                   */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/



tracks([arg]):=block(
if length(arg)=0 then go(ls2),
if part(arg,1)=help then return((
```

```
print("Tracks() : given the multivariable system dx/dt = f(x) +g(x)u,y=h(x),"),
print("           this function finds the control that makes the output y(t)"),
print("           track some desired trajectory yd(t).Tracks() accomplishes "),
print("           this by constructing first the left-inverse system using "),
print("           the function INVERT(f,g,h).  "),
print("           Tracks asks progressively for the data as it is needed."),
done)),ls2,

print("Hello,TRACKS tries to solve the problem:
display('diff('x,t)='f('x)+sum('g[i]('x)*'u[i],i,1,m)),
print("                              y(t)=h(x)"),
print("where    x is an n-dim vector
          u is an m-dim vector
          y is an m-dim vector"),

print("find the inverse system that takes as input the desired trajectory yd(t) an
n:read("enter dimension of the state space"),
m:read("enter dimension of the control space"),
f:read("enter the values of f in the form ",'f[1]('x),"...",'f[n]('x),"] followed
for i:1 thru m do
gi[i]:read("enter",'g[i]('x),"in the form ",'g[i,1]('x),"....",'g[i,n]('x),"] fol

g:makelist(gi[i],i,1,m),
h:read("enter the values of h in the form ",'h[1]('x),"...",'h[m]('x),"] followed
x0:read("enter the initial state x0 in the form ",'x0[1]('x),"...",'x0[n]('x),"fo
print ("enter the desired trajectory you want the output
of your system to track "),
for i:1 thru m do
yd[i]:read("yd[",i,"](t)= "),invert(f,g,h),
print("Checking if yd(t) is trackable by our system (right-invertibility)"),
lh1:0,
for i:0 thru k-1 do   for i1:1 thru m do (lh0:pnlidev(f,h[i1],i),
        for j:1 thru n do  lh0:subst(x0[j],concat('x,j),lh0),
          lh2:diff(yd[i1],t,i),
          if ev(lh2,t=0)#lh0 then (lh1:1,return(print(
      "The given yd(t) cannot be tracked by our system")))),
if lh1#1 then print("yd(t) can be tracked by the output y(t)"),kill(dd2),
              dd2:d.d2,
                    for j:1 thru m do
                  dd2:subst(yd[j],y[j],dd2),dd2:ev(dd2,diff),
dd2:makelist(dd2[i,1],i,1,m),
print(" The control ud(t) that makes y(t) track yd(t) is:"),
print("          "),
for i:1 thru m do print("    ud(t)[",i,"=]",ratsimp(dd2)),
ans:read("Are you interested in simulation results ? (answer y or n)"),
if ans = y then (file :read("enter filename of fortran code"),
                 ti:read("enter initial time you would the simulation to start fro
                 tf:read("enter final time tf"),
                 hi:read("enter step size h"),
                 xo:read("enter initial condition in the form [xo[1],...,xo[",n,"]
                 fl:f,gl:g,hl:h,
                 m2:m,
                 for i:1 thru n do(
                 fl:subst(x(i),concat('x,i),fl),
                 gl:subst(x(i),concat('x,i),gl),
                 hl:subst(x(i),concat('x,i),hl)),
                              ff(file)),
resp:read("would you like to try an other system ? (answer y or n)"),
if resp = y then go(init),return ("bye,have a good day !"))$
```

```
/*************************************************************************/
/*                                                                       */
/*    This function generates a fortran file using Runge-Kutta methods   */
/*    for simulation purposes.                                           */
/*                                                                       */
/*************************************************************************/
```

```
ff(fil):=block([equ],apply(writefile,[fil]),
        equ:makelist(concat('equ,i),i,1,n),
      print("      dimension x(",n,"),dx(",n,"),datad(1000,",m2,
           "),data(1000,"),
      print("     1",m2,"),u(",m,"),y(",m2,")"),
      print("c     set no of equations"),
      print("      n=",n),
      print("      m=",m),
      print("c     set initial conditions"),
      for j:1 thru n do
      fortran(x[j]:xo[j]),
      kill(x),
      for i:1 thru m2 do
      fortran(y[i]:hl[i]),
      kill(y),
```

```
print("c     set initial & final time"),
print("      t=",ti),
print("      tf=",tf),
print("c     set step size"),
print("      h=",hi),
print("c     desired trajectory"),
print("      no=",fix((tf-ti)/hi)),
print("      do 15 i=1,no"),
for j:1 thru m2 do
fortran(datad[i,j]:yd[j]),
print("      t=t+h"),
print(" 15   continue"),
print("c     store initial values for plotting"),
print("      do 25 i=1,",m2),
print("      data(1,i)=y(i)"),
print(" 25   continue"),
print("      t=0.0"),
print("      call control(x,t,u)"),
print("c     initialize k & mm"),
print("      k=0"),
print("      mm=1"),
print("      print *,t,datad(mm,1),data(mm,1)"),
print("c     write down the differential equations"),
kill(u),
for i:1 thru n do equ[i]: fl[i]+sum(gl[j][i]*u(j),j,1,m),
print(" 1    n=",n),
for j:1 thru n do
fortran(dx[j]:equ[j]),
print("      call runta(n,k,ii,x,dx,t,h,u)"),
print("      go to (1,2),ii"),
print(" 2    mm=mm+1"),
for i:1 thru m2 do
print("      y(",i,")=",hl[i]),

print("      do 30 i=1,",m2),
print("      data(mm,i)=y(i)"),
print(" 30   continue"),
print("      print *,t,datad(mm,1),data(mm,1)"),
print("      if(t.le.tf) go to 1"),
print("      stop"),
print("      end"),
for i:1 thru 4 do print("      "),
print("      subroutine runta(n,k,ii,x,dx,t,h,u)"),
print("      dimension yf(",n,"),z(",n,"),x(",n,"),dx(",n,"),u(",m,")"),
print("      k=k+1"),
print("      go to (1,2,3,4,5),k"),
print(" 2    do 10 j=1,n"),
print("      z(j)=dx(j)"),
print("      yf(j)=x(j)"),
print(" 10   x(j)=yf(j)+0.5*h*dx(j)"),
print(" 25   t=t+0.5*h"),
print("      call control(x,t,u)"),
print(" 1    ii=1"),
print("      return"),
print(" 3    do 15 j=1,n"),
print("      z(j)=z(j)+2.0*dx(j)"),
print(" 15   x(j)=yf(j)+0.5*h*dx(j)"),
print("      call control(x,t,u)"),
print("      ii=1"),
print("      return"),
print(" 4    do 20 j=1,n"),
print("      z(j)=z(j)+2.0*dx(j)"),
print(" 20   x(j)=yf(j)+h*dx(j)"),
print("      go to 25"),
print(" 5    do 30 j=1,n"),
print(" 30   x(j)=yf(j)+(z(j)+dx(j))*h/6.0"),
print("      call control(x,t,u)"),
print("      ii=2"),
print("      k=0"),
print("      return"),
print("      end"),
for j:1 thru 4 do print("      "),
print("      subroutine control(x,t,u)"),
print("      dimension u(",m,"),x(",n,")"),
for i:1 thru m do (for j:1 thru n do dd2[i]:subst(x(j),concat('x,j),
dd2[i]),
fortran(u[i]:dd2[i])),
print("      return"),
kill(u),
print("      end"),
apply(closefile,[fil]))$
```

References

[1] BOOTHBY, W. M., 1975, "*An Introduction to Differentiable Manifolds and Riemannian Geometry*," Academic Press, Inc.m New York.

[2] BROCKETT, R. W., 1978, "*Feedback Invariants for Nonlinear Systems*," IFAC Congress, Helsinki. p. 1115-1120.

[3] BROCKETT, R. W. and MESAROVIC, M. D., 1965, "*The Reproducibility of multivariable control systems*," J.Math.Anal.Appl., vol. 11, pp. 548-563.

[4] BRUNOVSKY, P., 1980, "*A classification of linear controllable systems*," Kibernetika (Praha), vol. 6, pp. 173-188.

[5] FORD, C. H., 1983, "*Numerical and Symbolic Methods for Transforming Control Systems to Canonical Form*," Ph.D Dissertation, Texas Tex University.

[6] FREUND, E., 1982, "*Fast nonlinear control with arbitrary pole placement for industrial robots and manipulators*," in Robot Motion: Planning and Control, ed. M. Brady, J. M. Hollerbach, T. L. Johnson, T. Lorenzo-Perez and M. T. Mason, MIT Press, Cambridge, Mass, pp. 147-168.

[7] HIRSCHORN, R. M., 1979, "*Invertibility of nonlinear control systems*," SIAM J Control Optim, 17, pp. 289-297.

[8] HIRSCHORN, R. M., 1979, "*Invertibility of multivariable nonlinear systems*," IEEE Trans Automat Control, AC-24, No. 6, pp. 855-865.

[9] HIRSCHORN, R. M. and DAVIS, J., 1985, "*Output Tracking for Nonlinear Systems With Singular Points*,"

[10] HUNT, R. L., SU, R., and MEYER, G., 1983, "*Design for multi-input systems*," Differential Geometric Control Theory, edited by R. Brockett, R. Millman and H. J. Sussman, Birkhauser, Boston, vol. 27, pp. 268-298.

[11] HUNT, R. L., SU, R., and MEYER, G., 1983, "*Global Transformations of Nonlinear Systems*," IEEE Trans Aut Control, AC-28, No. 1, pp. 24-31.

[12] ISIDORI, A., 1985, "*Nonlinear Control Systems: an Introduction*," Lecture Notes in Control and Information Science, 72, Springer-Verlag, Berlin.

[13] JAKUBCZYK, B., and RESPONDEK, W., 1980, "*On linearization of control systems*," Bull Acad Polon Sci, Ser Sci Math Astronom phys, 28, pp.517-522.

[14] KRENER, A. J., 1973, "*On the equivalence of control systems and the linearization of nonlinear systems*," SIAM J Control, 11, pp. 670-676.

[15] MARINO, R., 1982, "*Feedback Equivalence of Nonlinear Systems with Applications to Power Systems*," D.Sc. Dissertation, Washington University, St. Louis, Mo.

[16] MEYER, G., and CICOLANI, L., 1980, "*Application of nonlinear system inverses to automatic flight control design-system concepts and flight evaluations*," AGARDograph 251 on Theory and Applications of Optimal Control in Aerospace Systems, P.Kent, ed., reprinted by NATO.

[17] MEYER, G., 1981, "*The design of exact nonlinear model followers*," Proceedings of Joint Automatic Control Conference, FA3A.

[18] NIJMEIJER, H., 1982, "*Invertibility of affine nonlinear control systems: A geometric approach*," Syst Control Lett., 2, pp. 163-168.

[19] NIJMEIJER, H., 1986, "*Right-Invertibility for a class of nonlinear control systems: A geometric approach*," Syst Control Lett., 7, pp. 125-132.

[20] SILVERMAN, L. M., 1969, "*Inversion of multivariable linear systems*," IEEE Trans. Aut. Control, AC-14, No. 3, pp. 270-276.

[21] SU, R., 1982, "*On the linear equivalents of nonlinear systems*," Systems and Control Letters, 2, No 1, pp. 48-52.

[22] THE MATLAB GROUP LABORATORY FOR COMPUTER SCIENCE, 1983, "*MACSYMA Reference Manual*," M.I.T., Cambridge, Mass.

[23] WONHAM, W. M., 1979, "*Linear Multivariable Control: A Geometric Approach*," (New York: Springer-Verlag).