CAD TOOLS FOR NONLINEAR CONTROL

by

Raja R. Kadiyala

Memorandum No. UCB/ERL M92/37

17 April 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CAD TOOLS FOR NONLINEAR CONTROL

by

Raja R. Kadiyala

Memorandum No. UCB/ERL M92/37

17 April 1992

CAD TOOLS FOR NONLINEAR CONTROL

by

Raja R. Kadiyala

Memorandum No. UCB/ERL M92/37

17 April 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CAD Tools for Nonlinear Control *

Raja R. Kadiyala

Department of Electrical Engineering
and Computer Science
207-59 Cory Hall
University of California
Berkeley, CA 94720
email: raja@robotics.berkeley.edu

April 17, 1992

*Research supported in part by the Army under grants DAAL-88-K-0106 and DAAL-91-G-0191, and NASA under grant NAG 2-243

CAD Tools for Nonlinear Control

by
Raja R. Kadiyala

Abstract

In this manuscript I present a toolbox for nonlinear control system design. This toolbox (AP_LIN) contains modules to approximate systems by polynomials systems of arbitrary order and then render them input-output linear or input-state linear with error terms of arbitrarily high order. The approximation of the full nonlinear system to a polynomial nonlinear system allows us to compute the control law numerically as opposed to symbolically. Hence the computations can be made extremely fast. Furthermore, since the AP_LIN package is a stand alone package written in C, we have the ability to repeat the computations in real-time along the trajectory of the controlled system.

The task of approximating a system and updating the control law accordingly is very similar to adaptive control and we present a technique of indirect adaptive control based on certainty equivalence for input output linearization of nonlinear systems. This adaptive control scheme does not suffer from the overparameterization drawbacks of the direct adaptive control techniques on the same plant.

We give an example of the indirect control scheme by designing a controller for a model of the induction motor with the assumption that the magnetic subsystem is linear. We find that the adaptive nonlinear control law asymptotically renders the induction motor system input-output linear and also achieves input-output decoupling. In addition, we find that for the specific case of the induction motor we are able to prove parameter convergence and asymptotic tracking of an open set of reference trajectories using the indirect adaptive controller. This differs from the general indirect controller structure, where we cannot guarantee parameter convergence.

We also present a visualization tool which allows one to view the stability regions of nonlinear ordinary differential equations in three dimensions. We find that these computations may be carried out in parallel and present an algorithm for multiple networked workstations. We also discuss various viewing alternatives for the visualization of these dynamics.

An outline of the manuscript is as follows. We start with a review of the nonlinear control techniques implemented in the package and continue with a development of the indirect adaptive control scheme followed by the induction motor example. The *AP_ LIN* package is then presented along with *Sys_View* , a tool for the visualization of stability domains for dynamical systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

There has been a great wealth of theoretical machinery built up for controlling and analyzing nonlinear systems culminating in a rather complete characterization of linearization by state feedback and coordinate transformation [Isidori, 1989] and the approximation of nonlinear systems to linear systems with polynomial remainder terms of arbitrary high order [Krener, 1984, Krener *et al.*, 1987]. Feedback linearization, however, has been somewhat slow to catch on in *real world* applications mostly due to the fact that there does not exist a good Computer Aided Design (CAD) environment which handles feedback linearization of nonlinear systems. In turn, the nonlinear CAD toolbox development has been hampered in the past since the calculations necessary to formulate the feedback law are symbolic. Although there does exist software packages that handle symbolic computations (such as *Mathematica* and *Maple*), these packages are not robust and will cannot complete the necessary calculations in realtime with current or even next generation processor power. Furthermore small variations in the dynamical equations may yield vastly different compute times, thus one would not be able to guarantee the completion of the computations within a given sampling period.

We introduce a computer aided control system design (CACSD) package for nonlinear systems. The *AP_LIN* package allows one to approximate a system to an arbitrary order polynomial system and then perform the calculations to render the approximate system input output linear or linear up to arbitrary order terms.

The idea of using a system model created by a spline fit as the basis for control

system design is currently used in a (non-symbolic) package specific to flight control developed at the NASA Ames Research Center. This package carries through the necessary calculations, which then renders an aircraft system input-output linear (see [Meyer, 1990]). The controller created may then be readily implemented on the current generation of flight control computers.

Thus if we restrict ourselves to systems of the form

$$\begin{aligned} \dot{x} &= f(x) + \sum_{i=1}^{m} g_i(x) u_i \\ y_i &= h_i(x), \end{aligned}$$

where $f(x)$, $g_i(x)$, and $h_i(x)$ are vector valued *polynomials* and $x \in \mathbb{R}^n, u \in \mathbb{R}^m, y \in \mathbb{R}^l$, we are able to compute the linearizing feedback numerically as opposed to symbolically. Furthermore, we are able to handle the tabular data found in most flight control problems and also non-smooth problems in a systematic fashion. Restricting ourselves to polynomial systems is not overly constraining as many systems look either quadratic or cubic locally and we are free to use as many approximations as necessary over a region.

We choose a toolbox architecture for the *AP_LIN* package. The notion of using individual programs for each task was partly inspired by the mind-set introduced in [Wette and Laub, 1986]. This differs from the approach taken in [Krener *et al.*, 1991], which carries through the higher order linear approximation control design in a *MATLAB* based package. Furthermore since these computations are carried through numerically and written in *C* as stand alone routines which do not rely on a higher level program for memory allocation and parsing, they are extremely fast.

With the speed of these computations one may envision a realtime setting where we constantly update the polynomial approximation and recompute a new control law based on how the system is currently behaving. This algorithm has many of the same flavors as adaptive control as we have a parametrization of the model we choose to represent the system and we constantly update the parameters which in turn triggers changes in the control law.

Indeed, there has been much recent research in the use of adaptive control techniques for improving the input output linearization by state feedback of nonlinear systems with parametric uncertainty. Techniques of direct adaptive control (with no explicit identification) were proposed and developed in [Taylor *et al.*, 1989, Kanellakopoulos *et al.*, 1989,

Georgiou and Normand-Cyrot, 1989, Sastry and Isidori, 1987], (see also [Sastry and Bodson, 1989]). Nonlinear indirect adaptive control was initiated in [Bastin and Campion, 1989, Campion and Bastin, 1989, Pomet and Praly, 1988]. It is motivated by the fact that, with exact knowledge of the plant parameters, a nonlinear state feedback law and a suitable set of coordinates can be chosen to produce linear input-output behavior. In the case of parameter uncertainty, intuition suggests that parameter estimates which are converging to their true values can be used to asymptotically linearize the system. This heuristic is known as the *certainty equivalence principle.* Indirect adaptive control differs from direct adaptive control in that it relies on an observation error to update the plant parameters rather than relying on an output error. Hence, indirect adaptive control can be broken down into two parts. First, a parameter identifier is attached to the plant and adjusts the parameter estimates on line. These estimated parameters are then used in the linearizing control law, which bears resemblance to the realtime computation above.

Once we construct a nonlinear control law one would like to see the region which the controller design is valid or stable. Phase portraits have been used to study the stability characteristics of planar two dimensional dynamical systems with a good deal of success, but we are truly limited by this visualization scheme since we have the restriction of two dimensions. We extend the concept of phase portraits to three dimensions with *Sys_View* .

Sys_View does not create three dimensional phase portraits per se. Rather, it allows one to view the stability characteristics of a three dimensional subset of the dynamics. By this we mean that one is able to see where a system is stable and hence the region of attraction and validity of a control law. *Sys_View* also allows one to view the characteristics of how a system is unstable. In short, *Sys_View* allows the user to interactively view the stability characteristics of a three dimensional system or a three dimensional submanifold of the system.

As one might imagine the task of computing the stability characteristics of a three dimensional system is a rather large computational undertaking. Fortunately we note that a majority of the calculations do not depend on each other, hence the algorithm is ideal for parallel computing. We will show a novel approach for this problem using multiple networked workstations to offload the computation.

We start with a review of nonlinear techniques in chapter 2. More precisely, we review input output linearization for both single-input single-output and multiple-input multiple-output systems. We then follow with a review of approximate reduction of nonlin-

ear systems to linear systems with arbitrary order perturbation terms (so called Poincaré linearization).

In chapter 3 we present indirect adaptive control of nonlinear systems. This is an extension of the work in [Teel *et al.*, 1991] to MIMO systems and ties in with the realtime control of nonlinear systems presented later. We give an example of indirect adaptive control scheme for the nonlinear induction motor model presented in [Marino *et al.*, 1990] in chapter 4.

We then introduce the *AP_ LIN* package in chapter 5 and give some examples to show the power of *AP_ LIN*. We also discuss the realtime control algorithm above in more detail. In chapter 6, we present the *Sys_View* tool for the visualization of nonlinear stability characteristics.

Chapter 2

Review of Nonlinear Techniques

2.1 Introduction

To provide a framework to build upon, we review the two nonlinear control techniques of input-output linearization and the approximate reduction of nonlinear to linear systems of arbitrary order. We go into more detail of the latter theory since it is relatively new and may not be as well known as exact linearization. One may find a much more extensive review of the input-output linearization in [Isidori, 1989].

2.2 Exact Linearization (SISO)

We begin by reviewing, following the notation set in [Isidori, 1989], the basic theory of linearization by state feedback for the single-input single-output case. As we shall see the multiple-input multiple-output theory is similar, but more involved. Consider the SISO system

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{2.1}$$

with $x \in \mathbf{R}^n$,$u, y \in \mathbf{R}$ and f, g are smooth vector fields and h is a smooth function. Differentiating y with respect to time, one obtains

$$\begin{aligned} \dot{y} &= \frac{\partial h}{\partial x}\dot{x} \\ &= \frac{\partial h}{\partial x}f(x) + \frac{\partial h}{\partial x}g(x)u \\ &= L_f h(x) + L_g h(x)u \end{aligned} \tag{2.2}$$

Here $L_f h$, $L_g h$ stand for the Lie derivatives of h with respect to f, g respectively ($L_f h(x) = \frac{\partial h}{\partial x} f(x)$). If $L_g h(x)$ is bounded away from zero $\forall x \in \mathbf{R}^n$ then the control law

$$u = \frac{1}{L_g h}(-L_f h + v) \tag{2.3}$$

yields the linear system

$$\dot{y} = v. \tag{2.4}$$

If $L_g h(x) \equiv 0$, one continues to differentiate obtaining

$$y^{(i)} = L_f^i h(x) + L_g L_f^{i-1} h(x) u \quad i = 1, 2, \ldots. \tag{2.5}$$

If there is a fixed integer γ such that $\forall x \in \mathbf{R}^n$ $L_g L_f^i h \equiv 0$ for $i = 0, \ldots, \gamma - 2$ and $L_g L_f^{\gamma-1} h(x) \neq 0$ then the control law

$$u = \frac{1}{L_g L_f^{\gamma-1} h(x)}(-L_f^\gamma h(x) + v) \tag{2.6}$$

yields

$$y^{(\gamma)} = v. \tag{2.7}$$

The integer γ is called the *strict relative degree* of system (2.1).

Definition 2.1 strict relative degree *The system (2.1) is said to have* strict relative degree γ *at* x^0 *if*

$$L_g L_f^i h \equiv 0, \ \forall x \in B_r(x^0)$$

for $i = 0, \ldots, \gamma - 2$ *and*

$$L_g L_f^{\gamma-1} h(x) \neq 0$$

For a system with a strict relative degree γ, it is easy to verify that at each $x^0 \in \mathbf{R}^n$ there exists a neighborhood U^0 of x^0 such that the mapping

$$\Phi : U^0 \longrightarrow \mathbf{R}^n$$

defined as

$$\begin{aligned} \Phi_1(x) &= \xi_1 = h(x) \\ \Phi_2(x) &= \xi_2 = L_f h(x) \\ &\vdots \\ \Phi_\gamma(x) &= \xi_\gamma = L_f^{\gamma-1} h(x) \end{aligned} \tag{2.8}$$

with

$$d\Phi_i(x)g(x) = 0 \quad for \quad i = \gamma + 1, \ldots, n$$

is a diffeomorphism onto its image.

If we set $\eta = (\Phi_{\gamma+1}, \ldots, \Phi_n)^T$ it follows that the system may be written in the *normal form* ([Isidori, 1989]) as

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 \\ &\vdots \\ \dot{\xi}_{\gamma-1} &= \xi_\gamma \\ \dot{\xi}_\gamma &= b(\xi,\eta) + a(\xi,\eta)u \\ \dot{\eta} &= q(\xi,\eta) \\ y &= \xi_1. \end{aligned} \tag{2.9}$$

In equation (2.9), $b(\xi,\eta)$ represents the quantity $L_f^\gamma h(x)$ and $a(\xi,\eta)$ represents $L_g L_f^{\gamma-1} h(x)$. We assume that $x = 0$ is an equilibrium point of the system (i.e. $f(0) = 0$) and we assume that $h(0) = 0$. Then the dynamics

$$\dot{\eta} = q(0,\eta) \tag{2.10}$$

are referred to as the *zero-dynamics* (see [Isidori, 1989] section 4.3 for details). The nonlinear system (2.1) is said to be minimum phase if the equilibrium point $\eta = 0$ of the zero-dynamics is asymptotically stable.

2.2.1 Asymptotic Output Tracking

We now apply the normal form and the minimum phase property to the tracking problem. We desire to have $y(t)$ track a given $y_M(t)$. With u defined by (2.6), we choose

$$v = y_M^{(\gamma)} + \alpha_1(y_M^{(\gamma-1)} - y^{(\gamma-1)}) + \ldots + \alpha_\gamma(y_M - y) \tag{2.11}$$

with $\alpha_1, \ldots, \alpha_\gamma$ chosen so that

$$s^\gamma + \alpha_1 s^{\gamma-1} + \ldots + \alpha_\gamma \tag{2.12}$$

is a Hurwitz polynomial. Note that $y^{(i-1)} = \xi_i$. If we define $e_i = y^{(i-1)} - y_M^{(i-1)}$ then we have

$$\begin{aligned} \dot{e} &= Ae \\ \dot{\eta} &= q(\xi,\eta) \\ \xi_i &= e_i + y_M^{(i-1)} \end{aligned} \tag{2.13}$$

where A is the companion matrix associated with (2.12), and hence is a Hurwitz matrix.

It is easy to see that this control results in asymptotic tracking and bounded states ξ provided $y_M, \dot{y}_M, \ldots, y_M^{(\gamma-1)}$ are bounded.

A sufficient condition for η to remain bounded is exponential stability of the zero-dynamics and Lipschitz continuity of $q(\xi, \eta)$ in ξ, η. Thus, under these conditions, (2.6) and (2.11) yield bounded tracking. (see [Sastry and Isidori, 1987]).

2.3 Exact Linearization (MIMO)

Consider the square multiple-input multiple-output (MIMO) system defined below

$$\begin{aligned} \dot{x} &= f(x) + G(x)u \\ y &= H(x) \end{aligned} \tag{2.14}$$

where

$$G(x) = [g_1(x) \mid g_2(x) \mid \cdots \mid g_m(x)]$$

$$H(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ h_3(x) \\ \vdots \\ h_m(x) \end{bmatrix}$$

with $x \in \mathbf{R}^n, u, y \in \mathbf{R}^m$. Furthermore $f(x)$ and $g_i(x)$ are smooth vector fields and $h_j(x)$ are smooth functions. For each output y_i we may compute the k^{th} derivative as

$$y_i^{(k)} = L_f{}^k h_i + \sum_{j=1}^{m} L_{g_j} L_f{}^{k-1} h_i u_j \tag{2.15}$$

In the SISO case we differentiated the output y until the input appeared, similarly continue differentiating each output until some u_j appears (i.e. $L_{g_j} L_f{}^{k-1} h_i \neq 0$ for some j, k). Let γ_i be the smallest integer such that some u_j appears in $y_i^{\gamma_i}$ and define the $m \times m$ decoupling matrix $A(x)$ as

$$A(x) = \begin{bmatrix} L_{g_1} L_f{}^{\gamma_1 - 1} h_1 & \cdots & L_{g_m} L_f{}^{\gamma_1 - 1} h_1 \\ \vdots & \ddots & \vdots \\ L_{g_1} L_f{}^{\gamma_m - 1} h_m & \cdots & L_{g_m} L_f{}^{\gamma_m - 1} h_m \end{bmatrix} \tag{2.16}$$

and the vector

$$B(x) = \begin{bmatrix} L_f{}^{\gamma_1} h_1 \\ \vdots \\ L_f{}^{\gamma_m} h_m \end{bmatrix}. \tag{2.17}$$

With these definitions made we may express $y_i{}^{(\gamma_i)}$ as

$$\begin{bmatrix} y_1{}^{(\gamma_1)} \\ \vdots \\ y_m{}^{(\gamma_m)} \end{bmatrix} = B(x) + A(x) \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}. \tag{2.18}$$

We now extend the concept of relative degree for SISO systems to MIMO systems.

Definition 2.2 vector relative degree *The system (2.14) is said to have* vector relative degree $\gamma_1, \gamma_2, \ldots, \gamma_m$ *at* x^0 *if*

$$L_{g_i} L_f{}^k h_j \equiv 0;\ 1 \leq i \leq m, 0 \leq k \leq \gamma_j - 2,\ \forall x \in B_r(x^0) \tag{2.19}$$

for $j = 1, \ldots, m$ *and the matrix* $A(x)$ *is uniformly nonsingular in* x.

Now supposing the system (2.14) has well defined vector relative degree, we may apply the following state feedback

$$u = -A^{-1}(x)B(x) + A^{-1}(x)v \tag{2.20}$$

to obtain the closed loop system

$$\begin{bmatrix} y_1{}^{(\gamma_1)} \\ \vdots \\ y_m{}^{(\gamma_m)} \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}. \tag{2.21}$$

It should be noted that not only do we achieve input-output behaviour, but output decoupling is also attained.

2.3.1 Normal form for MIMO system

Given the system (2.14) with vector relative degree $\gamma_1, \ldots, \gamma_m$ and $\bar{\gamma} = \sum_{i=1}^{m} \gamma_i \leq n$ we may extend the notion of the SISO normal form to MIMO systems by choosing

coordinates as

$$\begin{aligned}
&\xi_1^1 = h_1(x), \quad \xi_2^1 = L_f h_1(x), \quad \cdots \quad \xi_{\gamma_1}^1 = L_f^{\gamma_1 - 1} h_1(x) \\
&\xi_1^2 = h_2(x), \quad \xi_2^2 = L_f h_2(x), \quad \cdots \quad \xi_{\gamma_2}^2 = L_f^{\gamma_2 - 1} h_2(x) \\
&\cdots \\
&\xi_1^m = h_m(x), \quad \xi_2^m = L_f h_m(x), \quad \cdots \quad \xi_{\gamma_m}^m = L_f^{\gamma_m - 1} h_m(x).
\end{aligned} \tag{2.22}$$

Complete the change of coordinates by choosing $n - \bar{\gamma}$ functions $\eta_1, \eta_2, \ldots, \eta_{n-\bar{\gamma}}$ which are independent from the ξ coordinates and each other. One difference from the SISO case is that in the MIMO case we are not able, in general, to choose the η coordinates such that $d\eta_i g_j \equiv 0$ as in the SISO case. Thus the input may appear in the expression for the $\dot{\eta}$ dynamics.

With the above change of coordinates defined we obtain the following normal form

$$\begin{aligned}
\dot{\xi}_1^1 &= \xi_2^1 \\
&\vdots \\
\dot{\xi}_{\gamma_1}^1 &= b_1(\xi,\eta) + \textstyle\sum_{j=1}^m a_{1,j}(\xi,\eta) u_j \\
\dot{\xi}_1^2 &= \xi_2^2 \\
&\vdots \\
\dot{\xi}_{\gamma_1}^2 &= b_2(\xi,\eta) + \textstyle\sum_{j=1}^m a_{2,j}(\xi,\eta) u_j \\
&\vdots \\
\dot{\xi}_1^m &= \xi_2^m \\
&\vdots \\
\dot{\xi}_{\gamma_1}^m &= b_m(\xi,\eta) + \textstyle\sum_{j=1}^m a_{m,j}(\xi,\eta) u_j \\
\dot{\eta} &= q(\xi,\eta) + P(\xi,\eta) u \\
y_1 &= \xi_1^1 \\
&\vdots \\
y_m &= \xi_1^m
\end{aligned} \tag{2.23}$$

with

$$\begin{aligned}
q_i(\xi,\eta) &= L_f \eta_i \\
P_{ij}(\xi,\eta) &= L_{g_j} \eta_i
\end{aligned}$$

and $a_{i,j}$ is the (i,j) entry of the decoupling matrix $A(x)$ and b_i is the i^{th} entry of the vector defined in (2.16) and (2.17), respectively.

Excluding the $\dot{\eta}$ dynamics one can view the MIMO normal form as m copies of the SISO normal form. Hence constructing tracking control laws will be virtually the same

as in the SISO case except that we must compute the feedback laws similar to (2.11) for each output.

2.3.2 Full State Linearization of MIMO systems

We now state conditions under which there exists m output functions $h_1, \ldots, h_m$ such that the dimension of the η coordinates is zero.

Let us first define the following distributions

$$\begin{aligned} G_0(x) &= sp\{g_1(x), \ldots, g_m(x)\} \\ G_1(x) &= sp\{g_1(x), \ldots, g_m(x), ad_f g_1(x), \ldots, ad_f g_2(x)\} \\ &\vdots \\ G_i(x) &= sp\left\{ad_f{}^k g_j(x) : 0 \leq k \leq i;\ 1 \leq j \leq m\right\} \end{aligned} \tag{2.24}$$

where $i = 1, \ldots, n-1$.

Theorem 2.1 MIMO Full State *Consider the system*

$$\dot{x} = f(x) + G(x)u$$

where the rank of $G(x^0)$ *is* m *and* $x \in \mathbf{R}^n, u \in \mathbf{R}^m$. *There exists* m *functions* $\lambda_1(x), \ldots, \lambda_m(x)$ *such that with the outputs defined as*

$$y = [\lambda_1(x), \ldots, \lambda_m]^T$$

the overall system has vector relative degree $r_1, \ldots, r_m$ *with*

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

if and only if

- *For each* $0 \leq i \leq n-2$ G_i *is involutive.*
- *For each* $0 \leq i \leq n-1$ G_i *has constant dimension.*
- *The dimension of* G_{n-1} *is* n.

Proof: see [Isidori, 1989, pp. 250-256]

We shall see that the solution to the MIMO full state linearization problem is very similar to the problem of when may one transform a nonlinear system to a linear system up to arbitrary order remainder terms which we shall discuss in more detail in section 2.4.

2.4 Approximate Transformation of Nonlinear Systems to Linear Systems

Consider the MIMO system defined below

$$\begin{aligned} \dot{x} &= f(x) + G(x)u \\ y &= H(x) \end{aligned} \tag{2.25}$$

where

$$G(x) = [g_1(x) \mid g_2(x) \mid \cdots \mid g_m(x)]$$

$$H(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ h_3(x) \\ \vdots \\ h_l(x) \end{bmatrix}$$

with $x \in \mathbf{R}^n, u \in \mathbf{R}^m, y \in \mathbf{R}^l$. Furthermore $f(x)$ and $g_i(x)$ are smooth vector fields and $h_i(x)$ are smooth functions. Note that (2.25) is no longer necessarily a square system (i.e. we do not require $m = l$).

Following the technique in [Krener *et al.*, 1987], we seek a coordinate change in the state space and output space along with state feedback such that the resulting **linear plant** will agree with (2.25) up to an error term of $O(x^{p+1}, x^p u)$ (i.e. terms of $O(x^{p+1})$ and $O(x^p u)$).

This is similar to a problem investigated by Poincaré [Arnold, 1983, ch. 5]: When can we transform

$$\dot{x} = Ax + f^{[2]}(x) + f^{[3]}(x) + \cdots \tag{2.26}$$

to the linear equation

$$\dot{s} = As \tag{2.27}$$

through a change of coordinates of the form

$$x = s + \phi^{[2]}(s) + \phi^{[3]}(s) + \cdots \tag{2.28}$$

where $f^{[r]}$ and $\phi^{[r]}$ are vector valued homogeneous polynomials of degree r (i.e. $f^{[r]}(x)$ and $\phi^{[r]}(x)$ would only contain $O(x^r)$ terms)?

2.4.1 Poincaré Linearization

Poincaré came up with necessary conditions for the above transformation to result in a linear system. Before we prove Poincaré 's theorem we first, however, need a few definitions.

Definition 2.3 resonant *The* n-tuple $\lambda = (\lambda_1, \ldots, \lambda_n)^T$ of eigenvalues is said to be resonant *if among the eigenvalues there exists an integral relation of the form*

$$\lambda_s = m^T \lambda \tag{2.29}$$

where $m = (m_1, \ldots, m_n)^T, m_k \in Z^+ \cup \{0\}, \sum_{i=1}^n m_i \geq 2$. *Such a relation is called a* resonance. *The number* $|m| = \sum_{i=1}^n m_i$ *is the* order *of the resonance.*

Example 2.1

$$\lambda_i = 2\lambda_j$$

is a resonance of order 2.

$$\lambda_i = -\lambda_j$$

is a resonance of order 3 since we have

$$\lambda_i = 2\lambda_i + \lambda_j$$

Let $\phi^{[r]}$ be an n-dimensional vector valued homogeneous polynomial of degree $r \geq 2$.

Lemma 2.1 *The differential equation* $\dot{s} = As$ *is transformed into*

$$\dot{x} = Ax + v(x) + f^{[r+1]}(x) + f^{[r+2]}(x) + \cdots$$

by the change of variables $x = s + \phi^{[r]}(s)$, *where* $v(x) = \frac{\partial \phi^{[r]}}{\partial x} Ax - A\phi^{[r]}(x)$.

Proof: By straight calculation we have

$$\begin{aligned} \dot{x} &= \dot{s} + \frac{\partial \phi^{[r]}}{\partial s} \dot{s} \\ &= As + \frac{\partial \phi^{[r]}}{\partial s} As \\ &= \left(I + \frac{\partial \phi^{[r]}}{\partial s} \right) As \end{aligned}$$

noting that $x = s$ up to order $(r-1)$ hence $s = x - \phi^{[r]}(x)$ up through order r we get

$$\begin{aligned}
\dot{x} &= \left(I + \frac{\partial \phi^{[r]}}{\partial s}\right) A \left(x - \phi^{[r]}(x) + O(x^{r+1})\right) \\
&= \left(I + \frac{\partial \phi^{[r]}}{\partial x}\right) A \left(x - \phi^{[r]}(x) + O(x^{r+1})\right) \\
&= Ax + \left[\frac{\partial \phi^{[r]}}{\partial x} Ax - A\phi^{[r]}(x)\right] + O(x^{r+1}) \\
&= Ax + v(x) + f^{[r+1]}(x) + f^{[r+2]}(x) + \cdots
\end{aligned}$$

□

Definition 2.4 homological equation *The* homological equation *associated with the linear operator A is the equation*

$$ad_A \phi^{[r]} = v \tag{2.30}$$

where $\phi^{[r]}$ is the unknown and v is the known vector field and ad_A is the Lie bracket of the linear field Ax and $\phi^{[r]}$ and is defined by

$$ad_A \phi^{[r]} = \frac{\partial \phi^{[r]}}{\partial x} Ax - A\phi^{[r]}(x)$$

ad_A maps homogeneous vector valued polynomials of degree r to homogeneous vector valued polynomials of degree r.

In the following, assume A has distinct eigenvalues $\lambda = (\lambda_1, \ldots, \lambda_n)^T$ with eigenvectors e_i and let $x^m = {x_1}^{m_1} \ldots {x_n}^{m_n}$.

Lemma 2.2 *The eigenvectors of the linear operator ad_A are the vector-valued monomials $x^m e_s$ and the eigenvalues of ad_A are given by*

$$m^T \lambda - \lambda_s \tag{2.31}$$

Proof: w.l.o.g. we can assume A to be diagonal

$$\begin{aligned}
ad_A x^m e_s &= \frac{\partial x^m e_s}{\partial x} Ax - Ax^m e_s \\
&= \left(\sum_{i=1}^{n} \lambda_i m_i\right) x^m e_s - \lambda_s x^m e_s \\
&= \left[m^T \lambda - \lambda_s\right] x^m e_s
\end{aligned} \tag{2.32}$$

□

Remark: ad_A is invertible if all eigenvalues of ad_A are different from zero (i.e. the eigenvalues are nonresonant). For the case when A does not have distinct eigenvalues ad_A will still have the same eigenvalues as before and if the eigenvalues are nonresonant then ad_A is invertible. If there are no resonances of order k then we can solve the homological equation of degree k.

We now state Poincaré 's theorem.

Theorem 2.2 Poincaré 's Theorem

If *The eigenvalues of A are nonresonant*

then *the equation*

$$\dot{x} = Ax + f^{[2]}(x) + f^{[3]}(x) + \cdots$$

may be transformed into the linear equation

$$\dot{s} = As$$

by a change of coordinates of the form

$$x = s + \phi^{[2]}(s) + \phi^{[3]}(s) + \cdots$$

where $\phi^{[r]}$ is an n-dimensional vector valued homogeneous polynomial of degree $r \geq 2$

Proof: Let

$$\dot{x} = Ax + v_r(x) + \cdots \tag{2.33}$$

Since the eigenvalues of A are nonresonant we may solve

$$ad_A h_r = v_r \tag{2.34}$$

for h_r. Make the substitution $x = s + h_r(x)$ to get

$$\dot{s} = As + w_{r+1} + O(s^{r+2}). \tag{2.35}$$

We may repeat the process to remove terms of degree $r+1, r+2, \ldots$. The limit substitution yields

$$\dot{s} = As \tag{2.36}$$

□

Remark: We did not show series convergence, but we may move the perturbation arbitrarily far by a convergent substitution.

From a control standpoint the above solution is appetizing, but not totally fulfilling since we have the extra freedom to choose u. So let us consider the MIMO control system defined in (2.25). Suppose that this system has the Taylor series expansion

$$\begin{aligned} \dot{x} &= Ax + Bu + f^{[2]}(x) + G^{[1]}(x)u + O(x^3, x^2u) \\ y &= Cx + H^{[2]}(x) + O(x^3) \end{aligned} \tag{2.37}$$

where $f^{[2]}(x)$ is a $n \times 1$ vector of degree 2 homogeneous polynomials, $G^{[1]}(x)$ is a $n \times m$ matrix of degree 1 homogeneous polynomials, and $H^{[2]}(x)$ is a $n \times 1$ vector of degree 2 homogeneous polynomials. Apply quadratic changes in the state space and output space

$$z = x - \phi^{[2]}(x) \tag{2.38}$$

$$w = y - \gamma^{[2]}(y) \tag{2.39}$$

to obtain

$$\begin{aligned} \dot{z} &= Az + Bu + f^{[2]}(z) + G^{[1]}(z)u \\ &- \left[Az + Bu, \phi^{[2]}(z)\right] + O(x^3, x^2u) \\ w &= Cz + H^{[2]}(z) + C\phi^{[2]}(z) \\ &- \gamma^{[2]}(w) + O(x^3) \end{aligned} \tag{2.40}$$

where $\left[Az + Bu, \phi^{[2]}(z)\right]$ is the previously defined Lie bracket of $Az + Bu$ and $\phi^{[2]}(z)$ $\left([f(z), g(z)] = \frac{\partial g}{\partial z} f(z) - \frac{\partial f}{\partial z} g(z)\right)$. Note that z agrees with x up through order 1 terms, hence they may be interchanged as the argument of a homogeneous polynomial of degree 1 or greater and included in the $O(x^3, x^2u)$ term. A similar scenario holds for u and v defined below. Now, apply state feedback of the form

$$u = \alpha^{[2]}(x) + (I + \beta^{[1]}(x))v \tag{2.41}$$

to obtain

$$\begin{aligned} \dot{z} &= Az + Bv + R_1{}^{[2]}(z, u) + O(x^3, x^2u) \\ w &= Cz + R_2{}^{[2]}(z, w) + O(x^3) \end{aligned} \tag{2.42}$$

where

$$\begin{aligned} R_1{}^{[2]}(z,u) &= f^{[2]}(z) + G^{[1]}(z)u \\ &- \left[Az + Bu, \phi^{[2]}(z)\right] \\ &+ B\left(\alpha^{[2]}(z) + \beta^{[1]}(z)u\right) \\ R_2{}^{[2]}(z,w) &= H^{[2]}(z) + C\phi^{[2]}(z) - \gamma^{[2]}(w) \end{aligned} \tag{2.43}$$

The equation

$$R_1{}^{[2]}(z,u) = 0 \tag{2.44}$$

is referred to as the *controller homological equation of degree 2.* If equation (2.44) can be solved for $\phi^{[2]}(z)$, $\alpha^{[2]}(z)$, and $\beta^{[1]}(z)$ then we would have

$$\dot{z} = Az + Bv + O(z^3, z^2u) \tag{2.45}$$

Further, suppose this system can be expanded as

$$\begin{aligned} \dot{z} &= Az + Bv + f^{[3]}(z) \\ &+ G^{[2]}(z)v + O(z^4, z^3u) \end{aligned} \tag{2.46}$$

We may repeat the above computations with

$$z = x - \phi^{[3]}(x) \tag{2.47}$$

$$v = \alpha^{[3]}(x) + (I + \beta^{[2]}(x))r \tag{2.48}$$

to get a system which is linear up to $O(z^4, z^3u)$.
Continuing the induction, we have at step p

$$\begin{aligned} \dot{x} &= Ax + Bu + f^{[p]}(x) \\ &+ G^{[p-1]}(x)u + O(x^{p+1}, x^pu) \end{aligned} \tag{2.49}$$

with change of coordinates and state feedback

$$z = x - \phi^{[p]}(x) \tag{2.50}$$

$$w = y - \gamma^{[p]}(y) \tag{2.51}$$

$$u = \alpha^{[p]}(x) + (I + \beta^{[p-1]}(x))v \tag{2.52}$$

yielding

$$\begin{aligned} \dot{z} &= Az + Bv + R_1^{[p]}(z,u) + O(x^{p+1}, x^p u) \\ w &= Cz + R_2^{[p]}(z,w) + O(x^p) \end{aligned}$$

where

$$\begin{aligned} R_1^{[p]}(z,u) &= f^{[p]}(z) + G^{[p-1]}(z)u \\ &- \left[Az + Bu, \phi^{[p]}(z)\right] \\ &+ B\left(\alpha^{[p]}(z) + \beta^{[p-1]}(z)u\right) \\ R_2^{[2]}(z,w) &= H^{[p]}(z) + C\phi^{[p]}(z) - \gamma^{[p]}(w) \end{aligned} \tag{2.53}$$

The equation

$$R_1^{[p]}(z,u) = 0 \tag{2.54}$$

is the *controller homological equation of degree p.*

In contrast to section 2.2, this method attempts to linearize the state space equations and linearize the output function separately, rather than the input-output map.

2.4.2 Solution of Controller Homological Equation

Counting the number of coefficients for each of the homogeneous polynomials $\phi^{[p]}(z)$, $\alpha^{[p]}(z)$, and $\beta^{[p-1]}(z)$ we have

$$(m+n)\begin{pmatrix} n+p-1 \\ p \end{pmatrix} + m^2\begin{pmatrix} n+p-2 \\ p-1 \end{pmatrix}$$

unknowns, where

$$\begin{pmatrix} n+p-1 \\ p \end{pmatrix}$$

is the number of coefficients in a p^{th} order homogeneous polynomial of dimension n. Counting the number of equations in 2.54, we have

$$n\begin{pmatrix} n+p-1 \\ p \end{pmatrix} + nm\begin{pmatrix} n+p-2 \\ p-1 \end{pmatrix}$$

n	m	p	no. equations	no. unknowns
2	1	2	10	11
3	1	2	27	27
4	3	2	88	106
8	1	2	352	332
9	3	2	648	621

Table 2.1: Problem size for various systems

linear equations, where

$$\begin{pmatrix} n \\ m \end{pmatrix} = \frac{n!}{(n-m)!m!},$$

p is the linearization order, n, m are the dimensions of the state space and input space. Typical problem sizes are given in table 2.1.

Typically this system of equations is either underdetermined or overdetermined. Solutions may be computed using the singular value decomposition to obtain the minimum coefficient size or minimize the Euclidean norm of the coefficients for the remainder term $R_1{}^{[p]}$, i.e.

- Underdetermined

$$min \left\| \begin{matrix} \phi^{[p]} \\ \alpha^{[p]} \\ \beta^{[p]} \end{matrix} \right\|_2$$

- Overdetermined

$$min \left\| R_1{}^{[p]} \right\|_2$$

2.4.3 Controller Implementation

In addition to linearization, one must also stabilize the plant. A subtle point that needs to be addressed is whether the placement of the eigenvalues of A using the standard state feedback control law

$$Fx + v$$

is done on the original system or the transformed system.

If we apply state feedback to the original system and $G^{[p-1]} \neq 0$, we gain another $O(x^p)$ term, but we know its value ($G^{[p-1]}Fx$) and may include it in $f^{[p]}$. Whereas if we apply it to the transformed system and $R^{[p]}(z,u) \neq 0$, we may introduce another $O(x^p)$ term to our system from which we just removed the $O(x^p)$ terms (or at least most of them) and we have no way to remove this new term. Thus, it is clear that one should apply state feedback before the transformation.

With this choice made, the iterations to compute the controller become:

1. Choose state feedback to place the poles of A:

$$u = Fx + v \tag{2.55}$$

2. Calculate $\phi^{[p]}(x)$, $\alpha^{[p]}(x)$, and $\beta^{[p-1]}(x)$ for the system

$$\begin{aligned} \dot{x} &= Ax + Bv + \hat{f}^{[p]}(x) \\ &+ G^{[p-1]}(x)v + O(x^{p+1}, x^p u) \end{aligned} \tag{2.56}$$

where $\hat{f}^{[p]}(x) = f^{[p]}(x) + G^{[p-1]}(x)Fx$

3. Define the coordinate change and feedback

$$\begin{aligned} z &= x - \phi^{[p]}(x) \\ v &= \alpha^{[p]}(z) + (I + \beta^{[p-1]}(z))r \end{aligned} \tag{2.57}$$

and set

$$w = y - \gamma^{[p]}(y) \tag{2.58}$$

If the state is not available, we may estimate z by

$$\dot{\hat{z}} = A\hat{z} + Bu + L(w - C\hat{z}) \tag{2.59}$$

2.4.4 Existence of Solutions to the Controller Homological Equation

In the previous section we showed how to calculate the change of coordinates and state feedback to remove arbitrary order terms, but the solvability of the set of linear equation could not be guaranteed. Following the framework in [Krener, 1984], we shall see that solvability conditions are very similar to those for the full state linearization of MIMO systems. We begin with some definitions.

Definition 2.5 order ρ basis *A Distribution $\mathcal{D}$ is said to have an* order ρ local basis *at x^0 if there exists vector fields $q_1, \ldots, q_d$ which are linearly independent at x^0 and for each $r \in \mathcal{D}$ there exists functions c_i such that*

$$r = \sum_{i=1}^{d} c_i q_i + O((x - x^0)^{\rho+1}) \tag{2.60}$$

Let $\{q_1, \ldots, q_d\}$ be an order ρ local basis at x^0, where d is the order ρ dimension of $\mathcal{D}$ at x^0.

Definition 2.6 order ρ involutive *A Distribution $\mathcal{D}$ is said to be* order ρ involutive *at x^0 if there exists functions ${C_k}^{ij}$ such that*

$$[g_i, g_j] = \sum_{k=1}^{d} {C_k}^{ij} q_k + O((x - x^0)^{\rho+1}) \tag{2.61}$$

where $g_i, g_j \in \mathcal{D}$.

Definition 2.7 order ρ integrable *A Distribution $\mathcal{D}$ is said to be* order ρ integrable *at x^0 if there exists $n - d$ independent functions $h_{d+1}, \ldots, h_n$ such that*

$$< dh_i, q_j >= O((x - x^0)^{\rho+1}) \tag{2.62}$$

We now state a variation of Frobenius' theorem.

Theorem 2.3 Frobenius with remainder *Let $\mathcal{D}$ be a distribution with order ρ basis $\{q_1, \ldots, q_d\}$ at x^0. $\mathcal{D}$ is order ρ integrable at x^0 if and only if $\mathcal{D}$ is order ρ involutive at x^0.*

Proof: see [Krener, 1984]

As in section 2.3, we define the following distributions

$$\begin{aligned} G_0(x) &= sp\{g_1(x), \ldots, g_m(x)\} \\ G_1(x) &= sp\{g_1(x), \ldots, g_m(x), ad_f g_1(x), \ldots, ad_f g_2(x)\} \\ &\vdots \\ G_i(x) &= sp\left\{ad_f{}^k g_j(x) : 0 \le k \le i;\ 1 \le j \le m\right\} \end{aligned} \tag{2.63}$$

where $i = 1, \ldots, n-1$.

Theorem 2.4 Krener *The system*

$$\dot{x} = Ax + Bu + f^{[\rho]}(x) + G^{[\rho-1]}(x)u + O(x^{\rho+1}, x^{\rho}u) \tag{2.64}$$

may be transformed into

$$\dot{z} = Az + Bv + O(z^{\rho+1}, x^{\rho}u) \tag{2.65}$$

with

$$\begin{aligned} z &= x - \phi^{[\rho]}(x) \\ u &= \alpha^{[\rho]}(x) + \left(I + \beta^{[\rho-1]}(x)\right) v \end{aligned} \tag{2.66}$$

if and only if

- *For each* $0 \leq i \leq n-2$ G_i *is order* ρ *involutive.*
- *For each* $0 \leq i \leq n-1$ G_i *has constant order* ρ *dimension.*
- *The order* ρ *dimension of* G_{n-1} *is* n.

Proof: The proof is essentially the same as for the MIMO full state linearization except we only require order ρ involutivity and use the modified version of the Frobenius theorem.

We see that the existence for the solution to the controller homological equation is a *relaxed* version of the full state linearization problem for MIMO systems. Hence we may solve the controller homological equation for a larger class of systems. Furthermore for those systems which we do not meet the above conditions we may achieve approximate solutions by finding the *closest* solution in a least squares sense.

Chapter 3

Indirect Adaptive Control of Nonlinear Systems

3.1 Introduction

We present a technique of indirect adaptive control based on certainty equivalence for the control scheme of input output linearization of nonlinear systems presented in chapter 2. This adaptive control law is proven convergent and does not suffer from the overparameterization drawbacks of the direct adaptive control techniques on the same plant.

In section 3.2, we review two identifier structures for nonlinear systems, (related observers have been presented in [Bastin and Campion, 1989, Kreisselmeier, 1977, Kudva and Narendra, 1973, Luders and Narendra, 1973]). Section 3.3 gives an outline of the indirect adaptive controller along with a proof of convergence.

3.2 Identifier Structures

Consider the system

$$\dot{x} = f(x,\theta^*) + \sum_{i=1}^{m} g_i(x,\theta^*)u_i \tag{3.1}$$

with $x \in \mathbf{R}^n$,$u \in \mathbf{R}^m$,$\theta^* \in \mathbf{R}^p$ and f, g_i are assumed to be smooth vector fields on $\mathbf{R}^n$. Further let $f(x,\theta^*)$ and $g_i(x,\theta^*)$ have the form

$$\begin{aligned} f(x,\theta^*) &= \textstyle\sum_{j=0}^{p} \theta_j^* f_j(x) \\ g_i(x,\theta^*) &= \textstyle\sum_{j=0}^{p} \theta_j^* g_i^j(x) \end{aligned} \tag{3.2}$$

Note that the indices for f and g_i start at zero. This allows us a simple way to include portions which do not depend on θ. Thus, it is assumed that $\theta_0 \equiv 1$ and $f_0(x), g_i^0(x)$ do not depend on θ. We are left with θ_i^*, $i = 1, \ldots, p$, as unknown parameters, which appear linearly, and the smooth vector fields $f_i(x)$, $g_i^j(x)$ are known. If we formulate the regressor

$$w^T(x,u) = [f_1(x) + \sum_{i=1}^{m} g_i^1(x)u_i, \ldots, f_p(x) + \sum_{i=1}^{m} g_i^p(x)u_i] \tag{3.3}$$

so that $w^T(x,u) \in \mathbb{R}^{n \times p}$ contains all of the nonlinearities of the system, then (3.1) can be written as

$$\dot{x} = w^T(x,u)\theta^* + f_0(x) + \sum_{i=1}^{m} g_i^0(x)u_i \tag{3.4}$$

3.2.1 Observer-based Identifier

To estimate the unknown parameters, we will use the identifier system

$$\begin{aligned} \dot{\hat{x}} &= A(\hat{x} - x) + w^T(x,u)\hat{\theta} + f_0(x) + \textstyle\sum_{i=1}^{m} g_i^0(x)u_i \\ \dot{\hat{\theta}} &= -w(x,u)P(\hat{x} - x) \end{aligned} \tag{3.5}$$

Here $A \in \mathbb{R}^{n \times n}$ is a Hurwitz matrix and $P \in \mathbb{R}^{n \times n} > 0$ is a solution to the Lyapunov equation

$$A^T P + PA = -Q, \quad Q > 0 \tag{3.6}$$

This identifier is reminiscent of one proposed in [Kudva and Narendra, 1973], [Kreisselmeier, 1977]. The choice $A = -\sigma I$ corresponds to a special case of the identifier. If we define $e_1 = \hat{x} - x$, the observer state error, and $\phi = \hat{\theta} - \theta^*$, the parameter error, and assume θ^* to be constant but unknown then we have the error system

$$\begin{aligned} \dot{e}_1 &= Ae_1 + w^T(x,u)\phi \\ \dot{\phi} &= -w(x,u)Pe_1 \end{aligned} \tag{3.7}$$

One should note the resemblance of the error equation above to that of the error equation of a full order *state* observer.

Theorem 3.1 Stability of Observer-based Identifier
Consider the observer-based identifier of equation (3.7),

then 1. $\phi \in L_\infty$,

2. $e_1 \in L_\infty \cap L_2$,

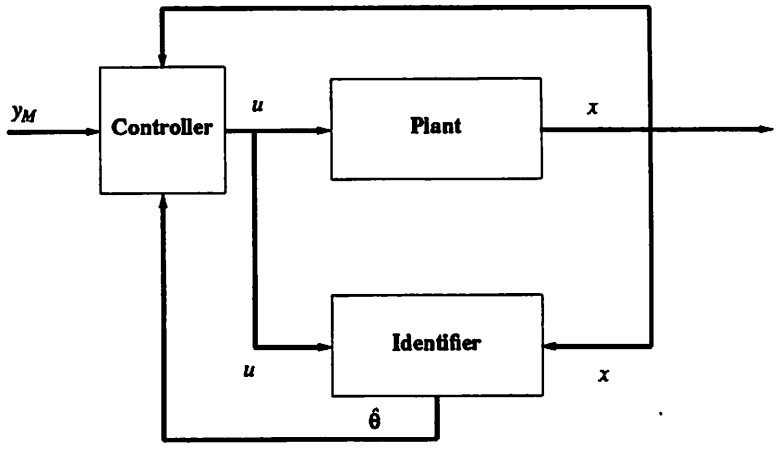

Figure 3.1: Block Diagram of an Indirect Adaptive Controller

3. If $w(x,u)$ *is bounded,*
then $\dot{e}_1 \in L_\infty$ *and* $\lim_{t\to\infty} e_1(t) = 0$.

Remarks:

1. The proof is a standard Lyapunov argument on the function

$$V(e_1,\phi) = e_1^T P e_1 + \phi^T \phi \tag{3.8}$$

2. The condition on the boundedness of w is a stability condition. In particular, if the system is bounded-input bounded-state (bibs) stable with bounded input, then w is bounded. (see [Sastry and Bodson, 1989])

3. Theorem 3.1 makes no statement about parameter convergence. As is standard in the literature one can conclude from (3.7) that e_1 and ϕ both converge exponentially to zero if w is sufficiently rich, i.e., $\exists \alpha_1, \alpha_2, \delta > 0$ such that

$$\alpha_1 I \geq \int_s^{s+\delta} w w^T dt \geq \alpha_2 I \tag{3.9}$$

This condition is impossible to verify explicitly ahead of time since w is a function of x. If we assume that the regressor is bounded it is clearly not necessary to have the upper bound in (3.9). Henceforth, when we use this result we will assume that the regressor is bounded.

3.3 Indirect Adaptive Tracking

In the case of parameter uncertainty, we have the system

$$\begin{aligned} \dot{x} &= f(x,\theta^*) + \textstyle\sum_{i=1}^{m} g_i(x,\theta^*)u_i \\ y_i &= h_i(x,\theta^*) \end{aligned} \tag{3.10}$$

with $\theta^* \in \mathbb{R}^p$ the vector of unknown parameters. We will make the following assumptions:

Assumption 3.1 Linear Parameter Dependence

The vector fields $f(x,\theta^*)$, $g_i(x,\theta^*)$ and the output function $h_i(x,\theta^*)$ in the system (3.10) depend linearly on the unknown parameters as

$$\begin{aligned} f(x,\theta^*) &= \textstyle\sum_{j=0}^{p} \theta_j^* f_j(x) \\ g_i(x,\theta^*) &= \textstyle\sum_{j=0}^{p} \theta_j^* g_i^j(x) \\ h_j(x,\theta^*) &= \textstyle\sum_{j=0}^{p} \theta_j^* h_j^j(x) \end{aligned}$$

where $f_i(x), g_i^j(x)$ are known smooth vector fields on $\mathbb{R}^n$ and $h_i(x)$ are known smooth scalar functions. Note that the indices start at zero (thus the portion which does not depend on parameters is included in the overall f, g_i and h_i for notational simplicity) and $\theta_0 \equiv 1$.

Assumption 3.2 Relative Degree

The vector relative degree of the true system (3.10) is $\gamma_1, \gamma_2, \ldots, \gamma_m$, and for all $\hat{\theta}$ in a ball around θ^* and all x in a neighborhood of x^0 the matrix

$$A(x,\hat{\theta}) = \begin{bmatrix} L_{g_1(x,\hat{\theta})} L_{f(x,\hat{\theta})}^{\gamma_1 - 1} h_1(x,\hat{\theta}) & \cdots & L_{g_m(x,\hat{\theta})} L_{f(x,\hat{\theta})}^{\gamma_1 - 1} h_1(x,\hat{\theta}) \\ \vdots & \ddots & \vdots \\ L_{g_1(x,\hat{\theta})} L_{f(x,\hat{\theta})}^{\gamma_m - 1} h_m(x,\hat{\theta}) & \cdots & L_{g_m(x,\hat{\theta})} L_{f(x,\hat{\theta})}^{\gamma_m - 1} h_m(x,\hat{\theta}) \end{bmatrix} \tag{3.11}$$

is nonsingular.

In the discussion that follows we will be using the implicit summation notation (ie. there is a summation over repeated indices) to keep the expressions manageable. For

example, we will write $f(x,\theta^*)$ as $\theta_j^* f_j(x)$. Now if we pick the following diffeomorphism

$$\Phi(x,\theta^*) = \begin{bmatrix} h_1(x,\theta^*) \\ L_{f(x,\theta^*)}h_1(x,\theta^*) \\ \vdots \\ L_{f(x,\theta^*)}^{\gamma_1-1}h_1(x,\theta^*) \\ \vdots \\ h_i(x,\theta^*) \\ L_{f(x,\theta^*)}h_i(x,\theta^*) \\ \vdots \\ L_{f(x,\theta^*)}^{\gamma_i-1}h_i(x,\theta^*) \\ \vdots \\ h_m(x,\theta^*) \\ L_{f(x,\theta^*)}h_m(x,\theta^*) \\ \vdots \\ L_{f(x,\theta^*)}^{\gamma_m-1}h_m(x,\theta^*) \\ \Phi_{\bar{\gamma}+1}(x,\theta^*) \\ \Phi_{\bar{\gamma}+2}(x,\theta^*) \\ \vdots \\ \Phi_n(x,\theta^*) \end{bmatrix} = \begin{bmatrix} \theta_{j_0}^* h_1{}^{j_0}(x) \\ \theta_{j_1}^*\theta_{j_0}^* L_{f_{j_1}} h_1{}^{j_0}(x) \\ \vdots \\ \theta_{j_{\gamma_1-1}}^* \cdots \theta_{j_0}^* L_{f_{j_{\gamma_1-1}}} \cdots L_{f_{j_1}} h_1{}^{j_0}(x) \\ \vdots \\ \theta_{j_0}^* h_i{}^{j_0}(x) \\ \theta_{j_1}^*\theta_{j_0}^* L_{f_{j_1}} h_i{}^{j_0}(x) \\ \vdots \\ \theta_{j_{\gamma_i-1}}^* \cdots \theta_{j_0}^* L_{f_{j_{\gamma_i-1}}} \cdots L_{f_{j_1}} h_i{}^{j_0}(x) \\ \vdots \\ \theta_{j_0}^* h_m{}^{j_0}(x) \\ \theta_{j_1}^*\theta_{j_0}^* L_{f_{j_1}} h_m{}^{j_0}(x) \\ \vdots \\ \theta_{j_{\gamma_m-1}}^* \cdots \theta_{j_0}^* L_{f_{j_{\gamma_m-1}}} \cdots L_{f_{j_1}} h_m{}^{j_0}(x) \\ \Phi_{\bar{\gamma}+1}(x,\theta^*) \\ \Phi_{\bar{\gamma}+2}(x,\theta^*) \\ \vdots \\ \Phi_n(x,\theta^*) \end{bmatrix} = \begin{bmatrix} \Phi_\xi(x,\theta^*) \\ \hline \Phi_\eta(x,\theta^*) \end{bmatrix} = \begin{bmatrix} \xi \\ \hline \eta \end{bmatrix} \tag{3.12}$$

where $\bar{\gamma} = \sum_{i=1}^m \gamma_i$ and $\Phi_{\bar{\gamma}+1}(x,\theta^*),\ldots,\Phi_n(x,\theta^*)$ are chosen so that $\Phi(x,\theta^*)$ has a nonsingular Jacobian matrix at x^0. Furthermore let us define the vector $B(x,\theta^*)$ as

$$B(x,\theta^*) = \begin{bmatrix} L_{f(x,\theta^*)}{}^{\gamma_1} h_1 \\ \vdots \\ L_{f(x,\theta^*)}{}^{\gamma_m} h_m \end{bmatrix} = \begin{bmatrix} \theta_{j_{\gamma_1}}^* \cdots \theta_{j_0}^* L_{f_{j_{\gamma_1}}} \cdots L_{f_{j_1}} h_1{}^{j_0}(x) \\ \vdots \\ \theta_{j_{\gamma_m}}^* \cdots \theta_{j_0}^* L_{f_{j_{\gamma_m}}} \cdots L_{f_{j_1}} h_m{}^{j_0}(x) \end{bmatrix}. \tag{3.13}$$

Then we have, in the normal form,

$$\begin{aligned}
\dot{\xi}_1^1 &= \xi_2^1 \\
&\vdots \\
\dot{\xi}_{\gamma_1}^1 &= b_1(\xi,\eta,\theta^*) + \textstyle\sum_{j=1}^m a_{1,j}(\xi,\eta,\theta^*)u_j \\
\dot{\xi}_1^2 &= \xi_2^2 \\
&\vdots \\
\dot{\xi}_{\gamma_1}^2 &= b_2(\xi,\eta,\theta^*) + \textstyle\sum_{j=1}^m a_{2,j}(\xi,\eta,\theta^*)u_j \\
&\vdots \\
\dot{\xi}_1^m &= \xi_2^m \\
&\vdots \\
\dot{\xi}_{\gamma_1}^m &= b_m(\xi,\eta,\theta^*) + \textstyle\sum_{j=1}^m a_{m,j}(\xi,\eta,\theta^*)u_j \\
\dot{\eta} &= q(\xi,\eta,\theta^*) + P(\xi,\eta,\theta^*)u \\
y_1 &= \xi_1^1 \\
&\vdots \\
y_m &= \xi_1^m
\end{aligned} \tag{3.14}$$

where

$$\begin{aligned}
q_i(\xi,\eta,\theta^*) &= L_{f(x,\theta^*)}\Phi_i(x,\theta^*) \quad \bar{\gamma}+1 \le i \le n \\
P_{ij}(\xi,\eta,\theta^*) &= L_{g_j(x,\theta^*)}\eta_i
\end{aligned} \tag{3.15}$$

and $a_{i,j}$ is the (i,j) entry of the decoupling matrix $A(x,\theta^*)$ and b_i is the i^{th} entry of the vector $B(x,\theta^*)$ defined in (3.11) and (3.13), respectively.

We assume that $x = 0$ is an equilibrium point of the system (3.10) (ie. $f(0,\theta^*) = 0$) and we assume $h_i(0,\theta^*) = 0$. Then the dynamics

$$\dot{\eta} = q_{\theta^*}(0,\eta) \tag{3.16}$$

are referred to as the zero-dynamics. The nonlinear system (3.10) is said to be minimum phase if the zero-dynamics are asymptotically stable. We will now impose the following assumption:

Assumption 3.3 Exponentially Stable Zero Dynamics

The equilibrium point $\eta = 0$ of the zero-dynamics of the true system (3.10) is exponentially stable.

Now let us consider our choice for the control law. The certainty equivalence principle suggests that we pick the appropriate linearizing control law but with the unknown parameters replaced by their estimates. We choose

$$u = -A^{-1}(x,\hat{\theta})B(x,\hat{\theta}) + A^{-1}(x,\hat{\theta})\hat{v} \tag{3.17}$$

To achieve tracking we pick $\hat{v}_i$ in the form of (2.11). However, we do not have exact expressions for the derivatives of y_i which involve unknown parameters. Instead we will use estimates of the derivatives of y_i obtained from the parameter estimates:

$$\hat{v}_i = y_{i_M}^{(\gamma_i)} + \alpha_{(1,i)}(y_{i_M}^{(\gamma_i-1)} - \hat{y}_i^{(\gamma_i-1)}) + \ldots + \alpha_{(\gamma_i,i)}(y_{i_M} - \hat{y}_i) \tag{3.18}$$

where

$$\hat{y}_i^{(k)} = \hat{\theta}_{j_k} \cdots \hat{\theta}_{j_0} L_{f_{j_k}} \cdots L_{f_{j_1}} h_i^{j_0}(x) \tag{3.19}$$

Now let us return to the normal form. Observe that $\dot{\xi}^i_{\gamma_i}$ can be written as

$$\begin{aligned}
\dot{\xi}^i_{\gamma_i} &= \theta^*_{j_{\gamma_i}} \cdots \theta^*_{j_0} L_{f_{j_{\gamma_i}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) + \theta^*_{j_{\gamma_i}} \cdots \theta^*_{j_0} L_{g_k} L_{f_{j_{\gamma_i-1}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) u_k \\
&- [\hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0} L_{f_{j_{\gamma_i}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) + \hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0} L_{g_k} L_{f_{j_{\gamma_i-1}}} \cdots L_{f_{j_1}} h_{j_0}(x) u_k] \\
&+ [\hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0} L_{f_{j_{\gamma_i}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) + \hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0} L_{g_k} L_{f_{j_{\gamma_i-1}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) u_k]
\end{aligned} \tag{3.20}$$

If we define the (large dimensional) vector of all multilinear parameter product errors,

$$\chi_i = (\hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0}) - (\theta^*_{j_{\gamma_i}} \cdots \theta^*_{j_0}) \tag{3.21}$$

then

$$\begin{aligned}
\dot{\xi}^i_{\gamma_i} &= \hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0} L_{f_{j_{\gamma_i}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) + \hat{\theta}_{j_{\gamma_i}} \cdots \hat{\theta}_{j_0} L_{g_k} L_{f_{j_{\gamma_i-1}}} \cdots L_{f_{j_1}} h_i^{j_0}(x) u_k \\
&+ \bar{z}_i^T(x,u)\chi_i
\end{aligned} \tag{3.22}$$

Note that if $\hat{\theta} - \theta^* \equiv \phi \to 0$ as $t \to \infty$ then $\chi_i \to 0$ as $t \to \infty$.

Substituting the certainty equivalence control law, we have

$$\dot{\xi_\gamma}^i = \hat{v}_i + \bar{z}_i^T(x,u)\chi_i \tag{3.23}$$

Therefore, in the closed loop we have

$$\begin{aligned}
\dot{e}^i &= A_i e^i + z_i^T(x,u)\chi_i \\
\dot{\eta} &= q(\xi,\eta,\theta^*) + P(\xi,\eta,\theta^*)u \\
\xi_j^{\ i} &= e_j^{\ i} + y_{i_M}^{(j-1)}
\end{aligned} \tag{3.24}$$

where A_i is a Hurwitz matrix, e^i is a $\gamma_i \times 1$ vector, and $i = 1, \ldots, m$.

We will now state the following bounded tracking result under parameter uncertainty:

Theorem 3.2 Convergence of Indirect Adaptive Controller When Identifier Input Is Sufficiently Rich

Consider the plant of equation (3.10) and the control objective of tracking the trajectory y_{i_M} and for each $i = 1, \ldots, m$

If (A1) *Assumption 3.1 holds (Linear Parameter Dependence),*

(A2) *Assumption 3.2 holds (Relative Degree),*

(A3) *Assumption 3.3 holds (Exponentially Stable Zero Dynamics),*

(A4) $|\chi_i| \to 0$ *as* $t \to \infty$,

(A5) $z_i^T(x,u)$ *is "cone bounded" in* x *and uniform in* u,

ie. $|z_i^T(x,u)| \leq \ell_{\bar{z}}|x| \ \forall u \in \mathbf{R}$,

(A6) A_i *is a Hurwitz matrix,*

(A7) $q(\xi^i, \eta)$ *is globally Lipschitz in* ξ^i, η,

(A8) $y_{i_M}, \dot{y}_{i_M}, \ldots, y_{i_M}^{(\gamma_i - 1)}$ *are bounded*

then *the control law given by (3.17) and (3.18) results in bounded tracking for the system (3.10). (ie., $x \in \mathbf{R}^n$ is bounded and $y_i(t) \to y_{i_M}(t)$.)*

Remarks:

1. The drawback with this result is that it needs the convergence of the identifier for its proof of asymptotic tracking. In turn, this requires the presence of sufficient richness which is not explicit in terms of conditions on the input. This is in contrast to the direct adaptive controller ([Sastry and Isidori, 1987]) where parameter convergence is not needed for stability and asymptotic tracking.

2. We reiterate that we are assuming the boundedness of the regressor, w, as stated in section 3.2. *Thus we explicitly disallow the possibility of finite escape time.*

Proof: For each $= 1, \ldots, m$ we may define b_d to be a bound on y_{i_M} and its derivatives. Then from (A8)

$$|\xi^i| \leq |e^i| + b_d \tag{3.25}$$

From (A6) $\exists P > 0$ such that

$$A^T P + PA = -I \tag{3.26}$$

Because x is a local diffeomorphism of (ξ, η)

$$|x| \leq \ell_x(|\xi| + |\eta|) \tag{3.27}$$

From (A5) and P defined in (3.26)

$$|2Pz_i{}^T(x,u)| \leq \ell_z|x| \tag{3.28}$$

From (A3) $\exists v_2(\eta)$ and positive constants $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ satisfying

$$\begin{array}{rcl} \sigma_1|\eta|^2 & \leq & v_2(\eta) \leq \sigma_2|\eta|^2 \\ \frac{\partial v_2}{\partial \eta} q(0,\eta) & \leq & -\sigma_3|\eta|^2 \\ |\frac{\partial v_2(\eta)}{\partial \eta}| & \leq & \sigma_4|\eta|. \end{array} \tag{3.29}$$

From (A7)

$$|q(\xi^i, \eta) - q(0,\eta)| \leq \ell_q|\xi^i| \tag{3.30}$$

From these bounds

$$\begin{array}{rcl} \frac{\partial v_2}{\partial \eta} q(\xi^i, \eta) & = & \frac{\partial v_2}{\partial \eta} q(0,\eta) + \frac{\partial v_2}{\partial \eta}(q(\xi^i,\eta) - q(0,\eta)) \\ & \leq & -\sigma_3|\eta|^2 + \sigma_4\ell_q|\eta|(|e^i| + b_d) \end{array} \tag{3.31}$$

With these preliminaries, we will show that e and η are bounded. Consider a Lyapunov function for the system (3.24)

$$V(e^i, \eta) = e^{iT} P e^i + \mu v_2(\eta) \quad \mu > 0. \tag{3.32}$$

Taking the derivative of $V(\cdot,\cdot)$ along the trajectories of (3.24) yields

$$\begin{array}{rcl} \dot{V} & = & e^{iT}(A^TP + PA)e^i + 2e^{iT}TPz_i{}^T(x,u)\chi_i + \mu\frac{\partial v_2}{\partial \eta} q(\xi^i,\eta) \\ & \leq & -|e^i|^2 + \ell_z\ell_x|e^i|(|e^i| + b_d + |\eta|)|\chi_i| + \mu(-\sigma_3|\eta|^2 + \sigma_4\ell_q|\eta|(|e^i| + b_d)) \\ & \leq & -(\frac{|e^i|}{2} - \ell_z\ell_x b_d|\chi_i|)^2 + (\ell_z\ell_x b_d|\chi_i|)^2 \\ & & -(\frac{|e^i|}{2} - (\ell_z\ell_x|\chi_i| + \mu\sigma_4\ell_q)|\eta|)^2 + (\ell_z\ell_x|\chi_i| + \mu\sigma_4\ell_q)^2|\eta|^2 \\ & & -\mu\sigma_3(\frac{|\eta|}{2} - \frac{\sigma_4\ell_q b_d}{\sigma_3}{}^2 + \frac{\mu(\sigma_4\ell_q b_d)^2}{\sigma_3} \\ & & -(\frac{1}{2} - |\chi_i|\ell_z\ell_x)|e^i|^2 - \frac{3}{4}\mu\sigma_3|\eta|^2 \\ & \leq & -(\frac{1}{2} - |\chi_i|\ell_z\ell_x)|e^i|^2 - [\frac{3}{4}\mu\sigma_3 - (\ell_z\ell_x|\chi_i| + \mu\sigma_4\ell_q)^2]|\eta|^2 \\ & & +(\ell_z\ell_x b_d|\chi_i|)^2 + \frac{\mu(\sigma_4\ell_q b_d)^2}{\sigma_3} \end{array} \tag{3.33}$$

Define

$$\mu_0 = \frac{\sigma_3}{4(\ell_z \ell_x + \sigma_4 \ell_q)^2} \tag{3.34}$$

Then, for $\mu \leq \mu_0$ and $|\chi_i| \leq min(\mu, \frac{1}{4\ell_z \ell_x})$, we have

$$\dot{V} \leq -\frac{|e^i|^2}{4} - \frac{\mu\sigma_3|\eta|^2}{2} + \frac{\mu(\sigma_4 \ell_q b_d)^2}{\sigma_3} + (\ell_z \ell_x b_d |\chi_i|)^2 \tag{3.35}$$

We can assume that $|\chi_i| \leq min(\mu, \frac{1}{4\ell_z \ell_x})$ for all $t \geq T$ from (A4). Also, the only previous restriction on μ was $\mu > 0$. Thus, for all $t \geq T, \dot{V} < 0$ when $|\eta|$ or $|e^i|$ is large which implies that $|\eta|$ and $|e^i|$, and hence $|\xi^i|$ and $|x|$ are bounded. If $q(\xi, \eta)$ is locally Lipschitz in (ξ, η) only on a set U and not all of $\mathbf{R}^n$ then the preceding analysis would hold so long as $|\chi(0)|$ is chosen small enough to guarantee that (ξ, η) lies in U. Consequently,

$$\dot{e}^i = A_i e^i + z_i{}^T(x, u)\chi_i \tag{3.36}$$

is an exponentially stable linear system driven by an input that approaches zero asymptotically. Thus, we conclude that the tracking error converges asymptotically to zero. □

3.4 Conclusion

We have presented convergence results for an nonlinear adaptive identifier and an output tracking result using indirect adaptive control. This approach was based on certainty equivalence for input output linearization of nonlinear systems. The form of the identifier did not need to be specified for the convergence result and overparameterization was not necessary. However, the result was based on an assumption of identifier convergence. An example of the indirect adaptive control scheme is given in the next chapter.

Chapter 4

Indirect Adaptive Control of Induction Motors

4.1 Introduction

The use of nonadaptive feedback linearization for the control of an induction motor has been quite popular and has been used successfully in [Krzeminski, 1987, Luca and Ulivi, 1987] and others. Furthermore, direct adaptive nonlinear controllers have been developed in both [Georgiou and Normand-Cyrot, 1989] and [Marino *et al.*, 1990]. However, in general, the direct adaptive control scheme requires overparametrization, i.e. extra parameters must be added in the controller (see [Sastry and Isidori, 1987]). The indirect adaptive control scheme does not suffer from this drawback.

In this chapter we use the techniques constructed in chapter 3 as applied to a fifth order symmetric induction motor model with linear magnetic circuits. This model was used in [Marino *et al.*, 1990]. We proceed with a description of the fifth order induction motor model in section 4.2 and carry through the calculations necessary for the identification scheme and the indirect adaptive controller. We end with simulations comparing a nonadaptive feedback linearization and an indirect adaptive controller.

4.2 Induction Machine Model

The model for a 2-phase symmetric induction machine with linear magnetics is derived in [Marino *et al.*, 1990]. To set notation we have $R_{\{r,s\}}, i_{\{sa,sb,ra,rb\}}, \psi_{\{sa,sb,ra,rb\}}$

representing the resistance, current and flux linkage with respect to the stator (s), rotor (r) and the stationary stator reference frame (a,b). The inputs are taken as the voltages to the stator, namely v_{sa}, v_{sb} and $x = [\omega, \psi_{ra}, \psi_{rb}, i_{sa}, i_{sb}]^T$.

We have, in the familiar $\dot{x} = f(x) + g_1(x)v_{sa} + g_2(x)v_{sb}$ form

$$f(x) = \begin{bmatrix} \frac{n_p M}{J L_r}(\psi_{ra} i_{sb} - \psi_{rb} i_{sa}) - \frac{T_L}{J} \\ \frac{R_r}{L_r}(M i_{sa} - \psi_{ra}) - n_p \omega \psi_{rb} \\ n_p \omega \psi_{ra} + \frac{R_r}{L_r}(M i_{sb} - \psi_{rb}) \\ \frac{M}{\sigma L_r}\left(\frac{R_r}{L_r}\psi_{ra} + n_p \omega \psi_{rb}\right) - \left(\frac{M^2 R_r + R_s {L_r}^2}{\sigma {L_r}^2}\right) i_{sa} \\ \frac{M}{\sigma L_r}\left(\frac{R_r}{L_r}\psi_{rb} - n_p \omega \psi_{ra}\right) - \left(\frac{M^2 R_r + R_s {L_r}^2}{\sigma {L_r}^2}\right) i_{sb} \end{bmatrix} \tag{4.1}$$

and

$$\begin{aligned} g_1 &= \begin{bmatrix} 0 & 0 & 0 & \frac{1}{\sigma} & 0 \end{bmatrix}^T \\ g_2 &= \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{\sigma} \end{bmatrix}^T \end{aligned} \tag{4.2}$$

where $\sigma = L_s - \frac{M^2}{L_r}$ and n_p is the number of pole pairs. Now choose the outputs for tracking as

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \omega \\ \psi_{ra}^2 + \psi_{rb}^2 \end{bmatrix}. \tag{4.3}$$

4.2.1 Input-Output Linearization of the Induction Motor

With the dynamics and outputs defined as above, we find that the system has vector relative degree of [2 2], hence the η dynamics are one dimensional. Thus the change of coordinates may be defined as

$$\Phi(x) = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_4 \\ \eta \end{pmatrix} = \begin{pmatrix} y_1 \\ \dot{y_1} \\ y_2 \\ \dot{y_2} \\ \arctan\left(\frac{\psi_{ra}}{\psi_{rb}}\right) \end{pmatrix} = \begin{pmatrix} \omega \\ \frac{n_p M}{J L_r}(\psi_{ra} i_{sb} - \psi_{rb} i_{sa}) - \frac{T_L}{J} \\ \psi_{ra}^2 + \psi_{rb}^2 \\ \frac{2R_r}{L_r}\left[M(\psi_{ra} i_{sa} + \psi_{rb} i_{sb}) - \psi_{ra}^2 - \psi_{rb}^2\right] \\ \arctan\left(\frac{\psi_{ra}}{\psi_{rb}}\right) \end{pmatrix} \tag{4.4}$$

where η completes the change of coordinates. Since $det\left(\frac{\partial \Phi(x)}{\partial x}\right) = \frac{4MR_r}{L_r}(\psi_{ra}^2 + \psi_{rb}^2)$ we must have the quantity $(\psi_{ra}^2 + \psi_{rb}^2)$ nonzero, which is true provided the motor is rotating. For $\Phi(x)$ to be a diffeomorphism we must also have the angle $\arctan\left(\frac{\psi_{ra}}{\psi_{rb}}\right) \in (\frac{-\pi}{2}, \frac{\pi}{2})$.

With this change of coordinates defined, we now proceed with the calculations to render the induction motor input-output linear and decoupled. If one differentiates y_1 and

$\dot{y}_2$ once each we obtain

$$\begin{bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{bmatrix} = \begin{bmatrix} L^2{}_f h_1(x) \\ L^2{}_f h_2(x) \end{bmatrix} + A(x) \begin{bmatrix} v_{sa} \\ v_{sb} \end{bmatrix} \tag{4.5}$$

where

$$\begin{aligned} A(x) &= \begin{bmatrix} L_{g_1} L_f h_1 & L_{g_2} L_f h_1 \\ L_{g_1} L_f h_2 & L_{g_2} L_f h_2 \end{bmatrix} \\ &= \frac{1}{\sigma L_r} \begin{bmatrix} -\frac{n_p M \psi_{rb}}{J} & \frac{n_p M \psi_{ra}}{J} \\ 2MR_r \psi_{ra} & 2MR_r \psi_{rb} \end{bmatrix}. \end{aligned} \tag{4.6}$$

which is nonsingular provided $(\psi_{ra}^2 + \psi_{rb}^2)$ is nonzero.

Choosing the state feedback of

$$\begin{bmatrix} v_{sa} \\ v_{sb} \end{bmatrix} = A^{-1}(x) \left\{ \begin{bmatrix} -L^2{}_f h_1(x) \\ -L^2{}_f h_2(x) \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \right\} \tag{4.7}$$

we get

$$\begin{bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{4.8}$$

which is input-output linear and decoupled. We may then apply a linear feedback of the form of (2.11, 3.18) to asymptotically track desired trajectories.

At this stage we point out a few important properties that hold for the induction motor model

- The parameters T_L and R_r enter linearly.
- The decoupling matrix $A(x)$ only depends on R_r and not T_L. The only way $A(x)$ can become singular through variations in R_r is if $R_r = 0$, hence the vector relative degree is well defined with respect to parameter variations.
- η remains bounded since it is a bounded function.
- The functions $f_i(x)$ multiplying the parameters are linear and hence Lipschitz.

4.2.2 Partitioning the Model for Adaptation

We wish to adapt to the load torque and the rotor resistance, T_L and R_r. As can be seen from (4.1) (4.2), both of these parameters enter linearly in $f(x)$ only, hence we need only partition f(x) as:

$$
\begin{aligned}
f_0(x) &= \begin{bmatrix} \frac{n_p M}{J L_r}(\psi_{ra} i_{sb} - \psi_{rb} i_{sa}) \\ -n_p \omega \psi_{rb} \\ n_p \omega \psi_{ra} \\ \frac{M}{\sigma L_r} n_p \omega \psi_{rb} - \frac{R_s}{\sigma} i_{sa} \\ -\frac{M}{\sigma L_r} n_p \omega \psi_{ra} - \frac{R_s}{\sigma} i_{sb} \end{bmatrix} \\
f_1(x) &= \begin{bmatrix} -\frac{1}{J} & 0 & 0 & 0 & 0 \end{bmatrix}^T \\
f_2(x) &= \begin{bmatrix} 0 \\ -\frac{1}{L_r}\psi_{ra} + \frac{M}{L_r} i_{sa} \\ -\frac{1}{L_r}\psi_{rb} + \frac{M}{L_r} i_{sb} \\ \frac{M}{\sigma L_r{}^2}(\psi_{ra} - M i_{sa}) \\ \frac{M}{\sigma L_r{}^2}(\psi_{rb} - M i_{sb}) \end{bmatrix}
\end{aligned}
\tag{4.9}
$$

with $[\theta_1{}^* \ \theta_2{}^*]^T = [T_L \ R_r]^T$ we have

$$
\begin{aligned}
\dot{x} &= f_0(x) + f_1(x)\theta_1{}^* + f_2(x)\theta_2{}^* \\
&+ g_1(x)v_{sa} + g_2(x)v_{sb}
\end{aligned}
\tag{4.10}
$$

and the regressor for the system is simply

$$
w^T(x) = [f_1(x) \ \ f_2(x)]. \tag{4.11}
$$

4.2.3 Conditioning of the Regressor

Since we need persistency of excitation to guarantee that the parameter error is driven to zero, let us examine the regressor more closely. By straight forward calculation we have for $ww^T(x)$

$$
\begin{bmatrix} \frac{1}{J^2} & 0 \\ 0 & \left(\begin{array}{c} \frac{M^2 + \sigma^2 L_r{}^2}{\sigma^2 L_r{}^4} \cdot \\ \left((\psi_{ra} - M i_{sa})^2 + (\psi_{rb} - M i_{sb})^2\right) \end{array} \right) \end{bmatrix}. \tag{4.12}
$$

Clearly this matrix is positive semi-definite and has rank of at least one (in fact with the parameters and trajectories used in the simulations this matrix was full rank throughout). For the case of persistency of excitation we need only concern ourselves with the conditioning of $\int ww^T(x)d\tau$ over some window of time.

We will assume that this integral is a matrix of full rank. This assumption is not constraining in any sense since $ww^T(x)$ is singular only if $\psi_{rb} = Mi_{sb}$ *and* $\psi_{ra} = Mi_{sa}$, which will not happen over an extended period of time provided the motor is rotating. Hence requiring these two equations to not hold identically over some period of time is not in the least bit restrictive and is met in all but a singular case.

4.2.4 Adaptive Input-Output Linearization of the Induction Motor

With the nonadaptive linearization framework set in section 4.2.1, we now proceed with the adaptive law. We first must identify the unknown parameters T_L and R_r. Using the equations from section 3.2 we have

$$\begin{aligned} \dot{\hat{x}} &= A(\hat{x}-x) + w^T(x,u)\hat{\theta} \\ &+ f_0(x) + g_1(x)v_{sa} + g_2(x)v_{sb} \\ \dot{\hat{\theta}} &= -w(x,u)P(\hat{x}-x) \end{aligned} \tag{4.13}$$

where $\hat{\theta} = [\hat{T_L}\ \hat{R_r}]^T$.

Note that since we know $f_0(x), g_1(x), g_2(x)$ exactly, there is no need to include it in the regressor. As can be seen in (3.17), the same control law as in section 4.2.1 is used except $\hat{T_L}$ and $\hat{R_r}$ are used in place of the true values. Thus the control law for the indirect method is very easy to implement since we use the same law as the nonadaptive case except parameter estimates are used for the unknowns. The identifier is the only additional piece that need be added.

Furthermore, since we have persistency of excitation and since the properties listed in section 4.2.1 hold for the induction motor we can guarantee asymptotic tracking of desired trajectories and the parameter error being driven to zero from the results in section 3.3.

4.3 Simulation Results

A 3-Hp induction motor was simulated using the indirect adaptive scheme with the motor parameters given in table 4.1. These values may be found in [Krause, 1986, p.

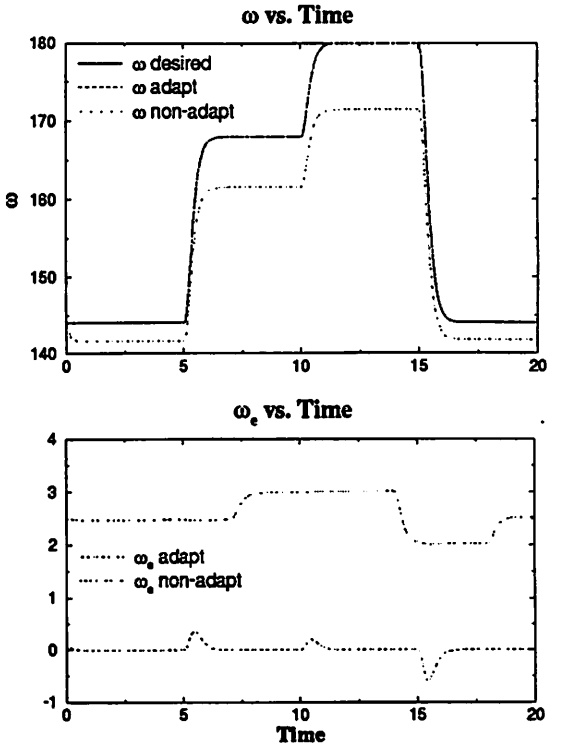

Figure 4.1: Tracking Results and Error for ω

190].

In the following we are allowing the parameters to be time varying. While it is assumed that the true parameters are constant but unknown for the proof to carry through, we simulate the more realistic scenario of the parameters varying with time.

The rotor resistance, R_r, can vary $\pm 50\%$ from its nominal value. In light of this we allowed the actual resistance to ramp from 50% of its nominal value to 150%. This simulates the rotor coils heating up, causing the resistance to increase. The initial estimate for R_r was set to the nominal value.

The load torque will be a function of the rotor speed. We model this in a similar fashion to [Nath and Berg, 1981] where T_L is related to ω quadratically. More specifically we have $T_L = 0.0012(0.05 + 0.3\omega^2)$. The initial estimate of T_L was 8 $N \cdot m$.

All of the simulations were carried out using $MATRIX_X$ version 2.4 with the variable step Kutta-Merson integration algorithm.

As can be seen in figures 4.1 and 4.2, the indirect adaptive control scheme worked extremely well. The trajectories for the indirect controller were virtually indistinguishable

Parameter	Value	Units
ω_{nom}	180	rad/sec
T_{max}	12	$N \cdot m$
J	0.089	$kg \cdot m^2$
n_p	2	
R_s	0.435	Ω
R_r	0.816	Ω
L_s	0.002	H
L_r	0.002	H
M	0.069	H

Table 4.1: Parameters for a 3-Hp Induction Motor

from the desired trajectories of ω and $\psi_{ra}^2 + \psi_{rb}^2$. Furthermore, from figure 4.3 one sees that the estimates for T_L and R_r reach their true values from initial offsets of 4 and 0.3, respectively, and then track the true parameters throughout.

Observing the error for the nonadaptive controller in the bottom half of figure 4.1, one sees that the shape of this waveform is similar to the desired trajectory for $\psi_{ra}^2+\psi_{rb}^2$. This is due to the fact that for the nonadaptive controller we do not achieve output decoupling since there is parameter mismatch. As a result, the outputs interact. This is the main advantage of using the adaptive scheme since one may always argue that a simple PID loop will regulate the offset in the tracking error. This would result in the steady state error being driven to zero (for constant trajectories), but the outputs will never be decoupled if there is any parameter mismatch.

4.4 Conclusion

A nonlinear indirect adaptive controller was designed for a fifth order induction motor model. The simulation results were quite good and achieved both asymptotic tracking and output decoupling and the parameter estimates for T_L and R_r converged to their true values. It should be noted that we assumed the rotor flux linkages, ψ_{ra} and ψ_{rb}, were measured. This typically would involve flux coils to be installed in the rotor. One may also use the observers developed in [Verghese and Sanders, 1988] to estimate these quantities.

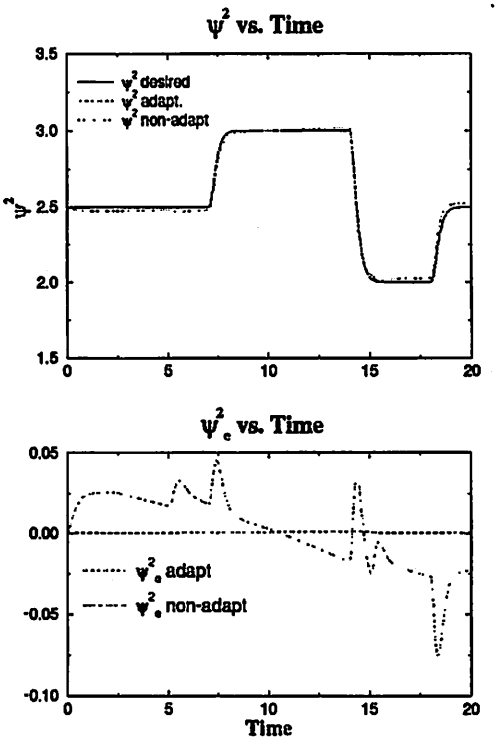

Figure 4.2: Tracking Results and Error for $\psi_{ra}^2 + \psi_{rb}^2$

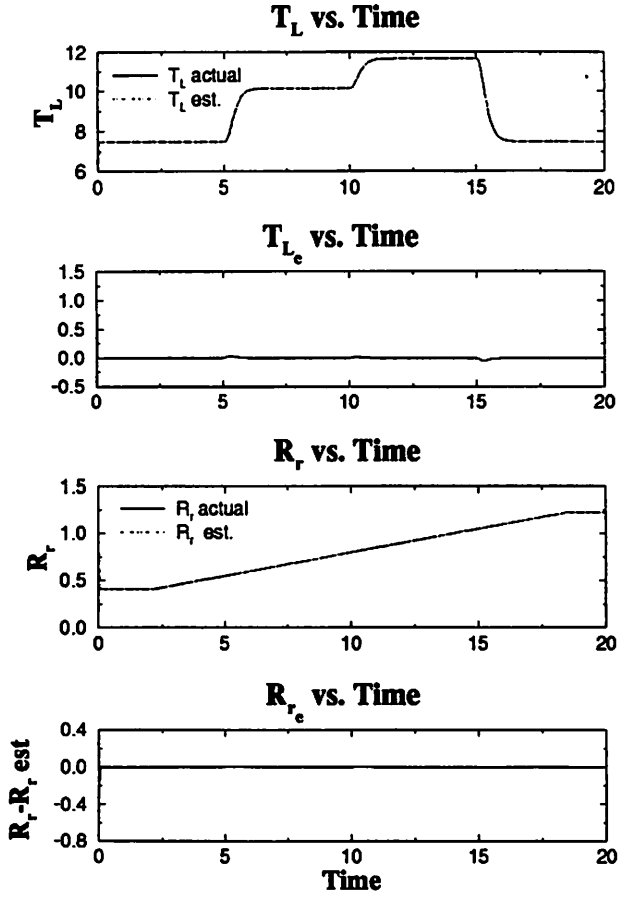

Figure 4.3: Parameter Estimates and Errors

Chapter 5

The AP_LIN Toolbox

5.1 Introduction

We present the *AP_ LIN* toolbox for the approximation and control of nonlinear systems. The toolbox includes a system approximator which takes a nonlinear system and gives back a polynomial system of arbitrary order. From this approximation we may generate a feedback linearizing controller or a controller which renders the system linear up to arbitrary order terms as in chapter 2.

The toolbox itself is discussed in section 5.2 and to show some of the features of the package we give a few examples in section 5.6 and a possible scenario for realtime control is given in section 5.7.

5.2 The *AP_ LIN* Toolbox

The *AP_ LIN* toolbox is a stand-alone set of programs all implemented in standard *C* to run on any UNIX platform. The notion of using individual programs for each task was partly inspired by the *toolbox* mind-set introduced in [Wette and Laub, 1986]. This differs from the approach taken in [Krener *et al.*, 1991], which carries through the higher order linear approximation control design in a *MATLAB* based package.

The heart of the toolbox is the polynomial system approximator which is discussed in section 5.3. As seen in figure 5.1 the toolbox can accept various input forms from CACSD packages and symbolic packages along with user defined subroutines. Other input forms will be added as time and resources permit. From these system descriptions the system

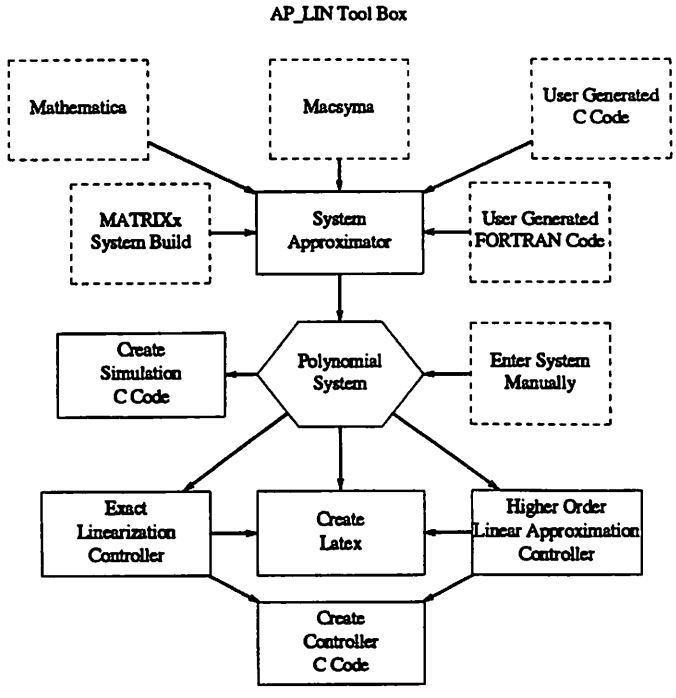

Figure 5.1: The *AP_ LIN* Toolbox Flow

approximator will then give back a polynomial system.

This polynomial system may be viewed (via LaTeX) or we may simulate the approximate system to check the validity of the spline fit. We may also create (based on the approximate system) the two controllers mentioned previously and generate *C* subroutines which may included in a simulation or executed on a real-time controller.

Currently there is a *SunWindows* interface to the various routines. In the future we plan to use *X* as the standard window system for the user-interface, but one can always use the simple UNIX command line sequences from a dumb terminal to perform the desired operations.

5.3 Polynomial System Approximator

The polynomial system approximator creates a multivariate spline fit of **arbitrary order** of

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

n	σ	no. parameters
2	2	2
2	4	8
3	3	18
4	3	60
8	2	660

Table 5.1: Number of parameters to be identified for various systems

to the system

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

where $\hat{f}(x)$, $\hat{h}(x)$ are vector valued polynomials in x and $\hat{G}(x)$ is a matrix of polynomials in x.

As stated previously, the approximation code accepts numerous input forms such as $MATRIX_X$, *Mathematica*, and user generated subroutines in *C* or *FORTRAN*. The spline fit can be about an input trajectory or a prescribed state trajectory and the computations can be made parallel for increased speed.

5.3.1 Computation of Coefficients for Spline Approximation

For each f_i, h_i, G_{ij}

$$\sum_{i=1}^{\sigma} \begin{pmatrix} n+i-1 \\ i \end{pmatrix}$$

parameters must be identified, where σ is the order of the approximation, and n is the dimension of the state space. $(m+2)$ singular value decompositions must then be computed to get the closest approximation in a least square sense. One advantage of assuming that the input enters affinely is that f_i and G_{ij} may be identified separately, thus reducing the size the identification problem in (at least) half.

One can easily see from table 5.1 that the number of parameters to be identified becomes quite large, but this is only if we take a ***black-box*** approach. Typically we will know the structure of our system and how the nonlinear terms come in and which state variables the nonlinear functions depend on. So, a more reasonable scenario is to have a system which

is mostly linear except for a few nonlinear terms which are a function of a small subset of the state variables. In this case we would get a much more reasonable problem size.

5.4 Computation of Approximate Reduction Control Law

AP_ LIN computes the change of coordinates and feedback for the scheme described in section 2.4 by matching the coefficients of the unknown (i.e. $\phi^{[r]}, \alpha^{[r]}, \beta^{[r-1]}$) polynomials to the terms present in the dynamics of the system. The singular value decomposition is the used to solve the linear least squares problem and obtain the polynomial coefficients. *AP_ LIN* can carry through the problem for arbitrary order, but as stated before the number of equations and unknowns becomes very large as the order increases.

Once the control law has been computed we may create LaTeXto view the controller and create *C* code that implements the control scheme.

5.5 Computation of Exact Linearization Control Law

The computation of the exact linearization control law can be carried through numerically (as opposed to symbolically) once the system has been approximated by a polynomial system. *AP_ LIN* basically caries through the calculations in section 2.2 for both SISO and MIMO systems. These computations are quite fast since we are only traversing data structures (integer operations) and the only floating point operations are multiplication and addition to update the derivatives of the outputs. Furthermore since all the operations are implemented in *C* we are able to store the structure and dynamics of the system compactly. One problem with relying on a matrix and vector storage based program (such as *MATLAB* or $MATRIX_X$) as the core to base a nonlinear CACSD package on is the wasted space. Typically one represents a multivariate polynomial as a large matrix with each element in the matrix representing a possible coefficient for the total polynomial. Depending on the computations we also may have duplicate entries (i.e. we may have allocated storage for both x_1x_2 and x_2x_1). While this structure lends itself to easy manipulation it is an extreme case of overallocation of resources (memory) since many of the entries in the matrix will be empty. Computations based on this type of storage will also be slowed since we may not assume that any entry is zero, thus each entry must be treated the same.

In doing these computations a set of data structures and core routines that handle

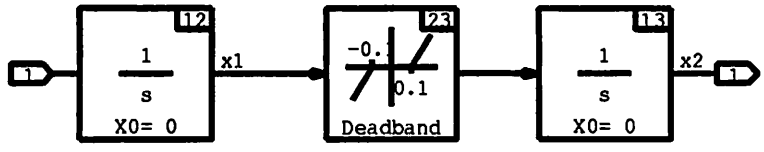

Figure 5.2: Backlash system from *SystemBuild*

basic operations on polynomials were developed such as multivariate polynomial addition, multiplication, and differentiation. The existence of these functions will speed development in the future as almost all algorithms will use these core routines.

5.6 Examples

In this section we give a couple of examples to show some the capabilities of the *AP_ LIN* toolbox. In the following it should be noted that all system and controller equations we generated by *AP_ LIN* in LaTeXform.

5.6.1 Backlash Example

In figure 5.2 we have a *SystemBuild* block generated within $MATRIX_X$. This *super − block* represents a simple model of backlash in a gear train. We have two states and a nonlinearity (dead-zone) sandwiched between them. Note that this system is not controllable when x_1 lies in the dead-zone region.

If we approximate this system with a third order polynomial system about a sinusoidal trajectory and ask for the system equations in LaTeXform we get

System Spline Model

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

and

$$y = h(x)$$

where $x \in \mathbf{R}^2$, $y \in \mathbf{R}^1$, and $u \in \mathbf{R}^1$. With

$$f(x) = \begin{bmatrix} 0 \\ 3.1 \cdot 10^3 {x_1}^3 \end{bmatrix}$$

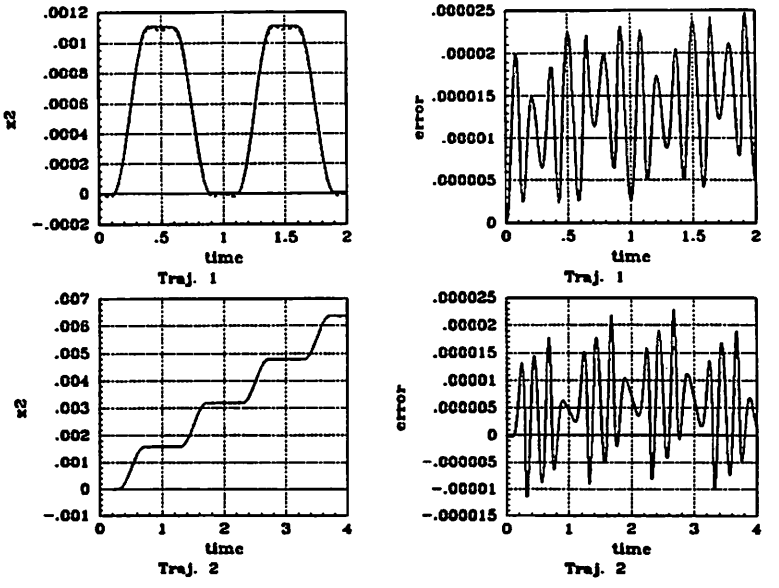

Figure 5.3: Third order approximation to the backlash system

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

$$h(x) = \begin{bmatrix} x_2 \end{bmatrix}$$

These set of equations are expected since the dead-zone can be reasonably approximated by a cubic function. The results of two simulations comparing the approximate system to the actual system about two trajectories is given in figure 5.3. The approximation comes extremely close in both cases and can be used as a model to base our control design on in this region. In fact it was so hard to differentiate the approximate and the actual system that we had to add the error plots.

Unfortunately finding a control law is not trivial since an exact linearization control law will have a singularity at $x_1 = 0$ and the Jacobian linearization of the approximate system is not controllable, hence the higher order linearization will not yield anything fruitful. We may apply the exact linearization control law with a preload function to avoid the singularity (i.e. if $\|x_1\| \leq \delta$ then use $x_1 = \epsilon sgn(x_1)$ in the control law). This controller is essentially high gain and therefore not very robust and we only present it to continue through the example. Results of a simulation to track an offset sinusoid through the dead-zone region is given in figure 5.4. The actual position (the dashed line) comes reasonably

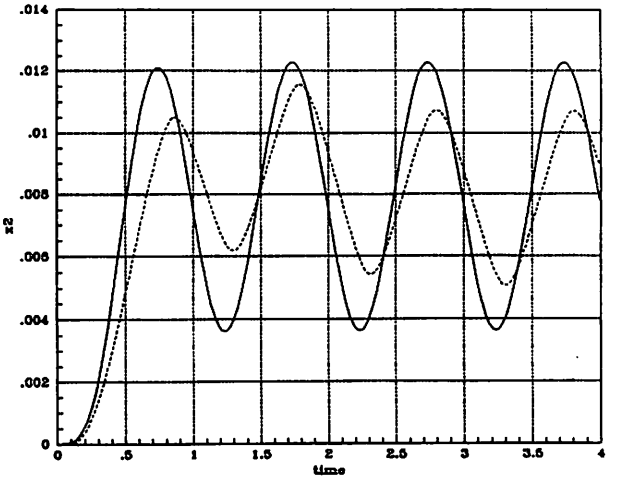

Figure 5.4: Modified exact linearization controller

close, less than the dead-zone region, to tracking the desired position (the solid line).

5.6.2 Simple Two Dimensional System

Consider the system

$$\begin{aligned} \dot{x_1} &= \sin(x_2) \\ \dot{x_2} &= u \\ y &= x_1 \end{aligned} \tag{5.1}$$

Let us run the system through the system approximator with a trajectory about the origin. We retrieve the third order polynomial system

$$\begin{aligned} \dot{x_1} &= x_2 - \frac{{x_2}^3}{6} \\ \dot{x_2} &= u \\ y &= x_1 \end{aligned} \tag{5.2}$$

Note that this system is the same as the system one would get by simply replacing $\sin(x_2)$ with the first two terms in the Taylor series expansion for $\sin(\cdot)$.

If we proceed through the exact linearization calculations on our actual system we get

$$\dot{y} = \sin(x_2)$$

$$\begin{aligned} \ddot{y} &= \cos(x_2)u \\ u &= \frac{v}{\cos(x_2)} \end{aligned} \tag{5.3}$$

It should be noted that this control law has a singularity at $x_2 = \pm n\frac{\pi}{2}$. Now if we proceed through the exact linearization algorithm on our polynomial system we get

$$\begin{aligned} \dot{y} &= x_2 - \frac{{x_2}^3}{6} \\ \ddot{y} &= u\left(1 - \frac{{x_2}^2}{2}\right) \\ u &= \frac{1}{\left(1 - \frac{{x_2}^2}{2}\right)} \end{aligned} \tag{5.4}$$

This control law has a singularity at $x_2 = \pm\sqrt{2}$. The higher order linearization methodology will give us

$$\begin{aligned} \phi^{[3]}(z) &= \begin{bmatrix} 0 \\ \frac{{z_2}^3}{6} \end{bmatrix} \\ \alpha^{[3]}(z) &= 0 \\ \beta^{[2]}(z) &= 0 \end{aligned} \tag{5.5}$$

where

$$x = z + \phi^{[3]}(z) \tag{5.6}$$

$$u = \alpha^{[3]}(x) + \left(I + \beta^{[2]}(x)\right)v \tag{5.7}$$

This coordinate change yields

$$\dot{z} = \begin{bmatrix} z_2 + \frac{{z_2}^5}{12} - \frac{{z_2}^7}{72} + \frac{{z_2}^9}{1296} \\ \\ \left(1 + \frac{{z_2}^3}{2} - \frac{{z_2}^5}{4} + \frac{{z_2}^7}{24} - \frac{{z_2}^9}{512}\right)v \end{bmatrix} \tag{5.8}$$

which has only $O(z,v)^4$ terms and higher. If we then close the loop with a linear control law on both nonlinear controllers based on the approximation and simulate the step response we get very similar results as seen in figure 5.5.

This last example shows two different nonlinear control approaches to a problem with about the same results. One can envision a case where perhaps the exact linearization

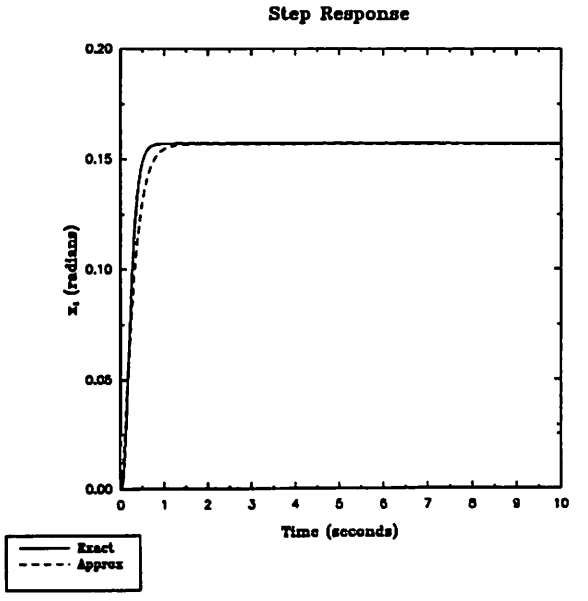

Figure 5.5: Step response of both nonlinear controllers

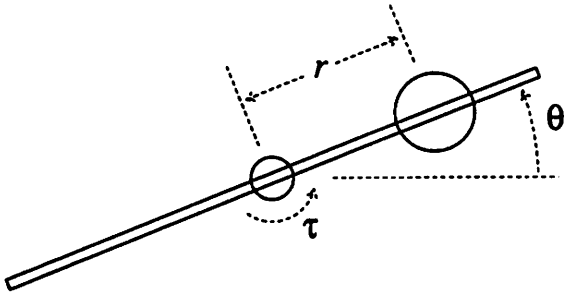

Figure 5.6: Ball and Beam

control law had an unavoidable singularity and the higher order linear approximation control scheme was stable in this neighborhood. The moral is that the design engineer must have options to turn to since there is not currently one omni-powerful methodology for nonlinear systems.

5.6.3 Ball and Beam Example

Figure 5.6 represents the so called ball and beam system which is comprised of a ball riding on a track. The control input is the torque of a motor at the center of the track which rotates the beam causing the ball to move accordingly. The equations for the system may be written (after a redefinition of the input) as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_1 x_4^2 - g \sin x_3 \\ x_4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u \qquad (5.9)$$

$$y = x_1$$

where $x = (x_1, x_2, x_3, x_4)^T := (r, \dot{r}, \theta, \dot{\theta})^T$

This system may be approximated by *AP_LIN* as:

System Spline Model

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

and

$$y = h(x)$$

where $x \in \mathbf{R}^4$, $y \in \mathbf{R}$, and $u \in \mathbf{R}$. With

$$f(x) = \begin{bmatrix} x_2 \\ -9.8x_3 + 1.6{x_3}^3 + x_1{x_4}^2 \\ x_4 \\ 0 \end{bmatrix}$$

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

$$h(x) = \begin{bmatrix} x_1 \end{bmatrix}$$

The $x_1{x_4}^2$ term causes the system to have relative degree 3, hence we would have unobservable dynamics if we used an exact linearization control law on this model. Unfortunately these dynamics are not minimum phase, thus we may not achieve asymptotic tracking. If we, however, ignore this term in the computation of the linearizing control law,

then we would have a relative degree 4 system, and hence no zero dynamics. This further approximation is valid if x_4 remains small.

It is quite easy to make this additional approximation (just set the term to zero) and have *AP_ LIN* churn through the calculations to create the linearizing control law. The calculation took less than 20 *msec* on a SparcStation 1. The control law as created by *AP_ LIN* is given below.

Input-Output Linearizing Controller

$$x \in \mathbf{R}^4,\ y \in \mathbf{R}, \text{ and } u \in \mathbf{R},$$

And the system has relative degree of 4

$$u = -A^{-1}(x)B(x) + A^{-1}(x)v$$

Where

$$A(x) = \begin{bmatrix} -9.8 + 4.9{x_3}^2 \end{bmatrix}$$

$$B(x) = \begin{bmatrix} 9.8x_3{x_4}^2 \end{bmatrix}$$

And the diffeomorphism:

$$\begin{bmatrix} y_1 \\ \dot{y_1} \\ \ddot{y_1} \\ y_1^{(3)} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ -9.8x_3 + 1.6{x_3}^3 \\ -9.8x_4 + 4.9{x_3}^2x_4 \end{bmatrix}$$

Defines the linearizing change of coordinates.

If we now ask *AP_ LIN* to generate a *C* subroutine for the controller so we may simulate the system, then we get the results shown in figure 5.7 for the ball position (x_1) tracking a sinusoid, (the solid line and the dashed line, respectively).

The *AP_ LIN* toolbox gives us a platform to rapidly include nonlinear control schemes since we restrict ourselves in the end to polynomial systems. The data structures and manipulation routines have all been written to handle the multivariate polynomials. Hence we may easily implement other nonlinear algorithms such as the nonlinear regulator

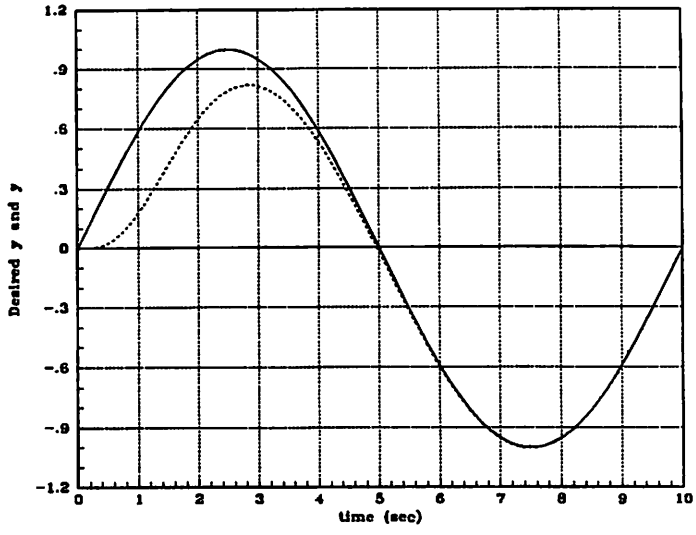

Figure 5.7: Tracking Results for the Ball and Beam

([Byrnes and Isidori, 1990]), or the approximate control methods discussed in [Hauser *et al.*, 1988], or even some adaptive schemes such as [Sastry and Isidori, 1987] and [Teel *et al.*, 1991]. Thus we can give the control designer the options and flexibility necessary.

5.7 Real-time Control

In table 5.2 we have summarized the time it takes for the computation of the approximation and to compute the higher order linearization control law on various computer systems for different problem sizes. It should be noted that these are all UNIX workstations running in a multi-user environment. In a real time setting the times would be even smaller. The code was compiled using the standard *C* compiler provided by the computer manufacturer and with the default level of optimization. The machine labeled *SS2-i860* is a Sun SparcStation 2 with an Intel i860 array processor connected to it. For this set up the singular value decomposition was ported to the array processor and was solely executed on it. For the smaller problems the overhead involved in setting up the shared memory and passing the data dominated the timing figures, but as the problem became larger the speed of the i860 dominated.

The computation of the input output linearizing controller took 10 *msec* on all

System with 2 states, one input and output					
Approximation					
		Time (msec)			
Order	Par.	DEC 5000	SUN 4/370	SS2	SS2-i860
3	10	35	60	40	60
4	15	98	220	110	110
5	21	230	510	240	230
13	105	46981	47270	32160	21860

Controller Computation				
	Time (msec)			
Order	DEC 5000	SUN 4/370	SS2	SS2-i860
3	32	70	50	70
4	74	160	90	90
5	133	260	150	140

Table 5.2: Computation time for various processors

platforms. The speed of execution is due to the nature of the computation (mainly multiply and additions). So if on were to use the exact linearization control law then the main computational load will be in the system approximation which can be made much more manageable if we use our knowledge of the system and do not take a *black box* approach.

It is quite clear that for small problems we could currently do the approximation and create a control law in real-time. With more optimization and faster (perhaps parallel) processing power one will be able to handle even larger problems.

So one may envision a scenario as depicted in figure 5.8 where we have a controller running at some fixed rate and at a slower time scale we have the system approximator gathering the inputs, states, and outputs to create an approximation of the system in its current operating region. This approximation is then feed to a another processor which will compute a new control law and update_the controller.

Another possibility would be to compute several approximations at various operating points and then apply a *nonlinear gain scheduling* algorithm to switch between the corresponding controllers.

The code generated by *AP_LIN* is minimal in size and has the benefit of having all floating point operations restricted to multiplications and additions. This *feature* is realized from the fact that we are restricting our systems to polynomial systems. Furthermore the

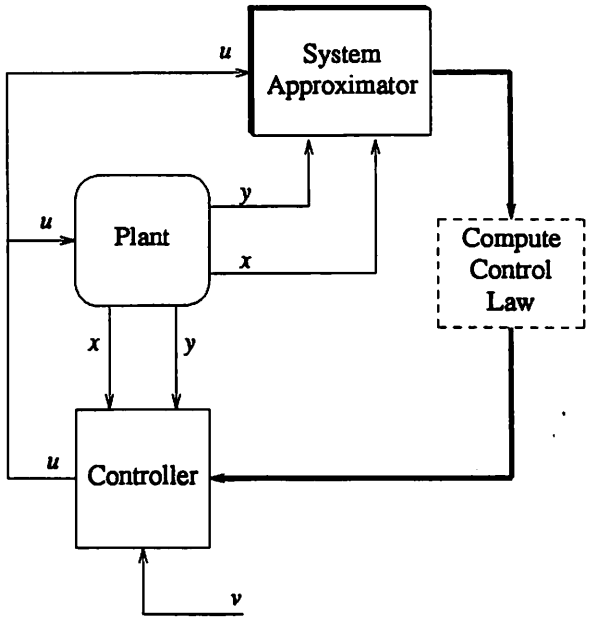

Figure 5.8: Real-time control diagram

nature of these calculations lend themselves ideally to digital signal processors (DSP) chips which currently have as much as 100 Mflops (millions of floating point operations) and in the future will span in the giga-flop speed.

One interesting thing to note is that the above scheme is very similar to adaptive control in that we have:

- A parametrization of the plant (polynomial).
- An identifier to estimate these parameters.
- A certainty equivalence control law based on the updated parameters.

One would expect that certain richness conditions on the plant must also be meet. It would also be easy to add another identifier running on the outer loop which would be able to handle parametric uncertainty in the plant which we are approximating. Hence many of the same problems/solutions which come up in techniques such as the indirect scheme from 3 may be applied here.

5.8 Conclusion

We have presented a toolbox for nonlinear control system design. The *AP_LIN* toolbox can currently approximate a system to a polynomial system and then carry through the computations to input-output linearize a class of systems or compute another control law which renders a system linear up to arbitrary order error terms. New modules can be easily incorporated and will allow the control designer the flexibility to choose amongst them as more design schemes are added.

We have also shown that for small size problems it is currently feasible to implement a system approximator and routines to compute control laws in a real-time setting. As processor speeds continue to climb we will be able to handle larger and larger size problems.

Chapter 6

Sys_View

6.1 Introduction

We discuss a visualization tool which allows one to view the stability characteristics of nonlinear ordinary differential equations in three dimensions. We find that these computations may be carried out in parallel and present an algorithm for multiple networked workstations. We also discuss various viewing alternatives for the visualization of these dynamics.

We start with some simple definitions and an overview of the types of systems we can handle in section 6.2 and discuss the parallel algorithm in section 6.3, and lastly in section 6.4 we look at the visualization methods developed to survey all the information inherent in these plots.

6.2 Extension into Three Dimensions

Sys_View was motivated by a discrete time version which looked at extended two dimensional Mandelbrot sets into three dimensions. We first discuss the discrete time case.

6.2.1 Discrete Time

The defining equation for a Mandelbrot set is given by:

$$x_{k+1} = f(x_k) + R \tag{6.1}$$

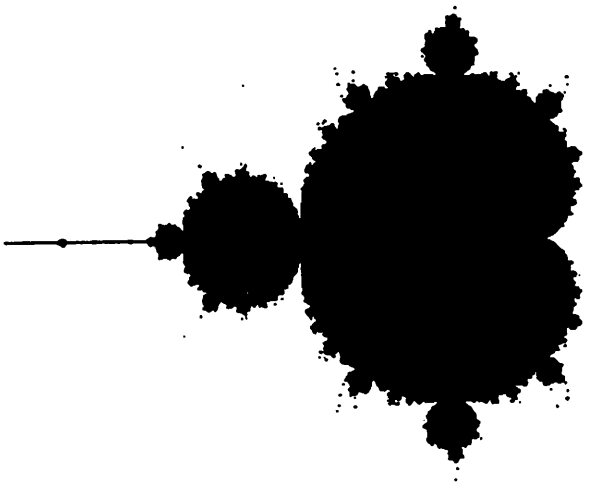

Figure 6.1: Unenhanced Mandelbrot Set

where $x, R \in \mathbb{C}$, $f(\cdot) : \mathbb{C} \rightarrow \mathbb{C}$. This gives rise to the familiar *vanilla* mandelbrot set shown in figure 6.1, where one plots all the points which converge (a discrete version of the region of attraction).

These sets may be enhanced by color to give a more artistic flair and display more information about the system dynamics. A typical mapping is to assign each point a color according to how many iterations it takes to escape some neighborhood of the start point (the so called *escape time*).

Equation (6.1) is very similar to a general class of nonlinear discrete time systems which is affine in the control input. These systems may be described by:

$$x_{k+1} = f(x_k) + g(x_k)u_k \tag{6.2}$$

where $x \in \mathbf{R}^n, u \in \mathbf{R}$.

It is quite easy to see that the Mandelbrot sets are a subclass of the above system definition. Hence we have a simple natural extension into three dimensions. Interestingly enough, it was found that systems such as

$$\begin{aligned} x_1(k+1) &= e^{\sin(x_1(k))*x_1(k)} \\ x_2(k+1) &= e^{\sin(x_2(k))*x_2(k)} \\ x_3(k+1) &= e^{\sin(x_3(k))*x_3(k)} \end{aligned} \tag{6.3}$$

and other similar equations did indeed create Mandelbrot like subsets filling three space.

6.2.2 Continuous Time

The continuous time situation is similar to discrete time, except that we have the defining equation to be

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i \tag{6.4}$$

We may broaden this class to include systems which are not affine in the control input u such as:

$$\dot{x} = f(x, u) \tag{6.5}$$

where we now have $u \in \mathbb{R}^m$.

The evaluation of (6.5) is not as straight forward as (6.2), and requires numerical integration to achieve a solution for x. Simple iteration would achieve a solution for x_k in the discrete time case.

The inherent complexity in integrating differential equations causes the computation time to naturally rise and also leads to problems under certain conditions. One in particular is the integration of *stiff* systems.

Definition 6.1 Stiff System

A differential equation is said to be stiff *if for some* i, j $\|\dot{x}_i\| \gg \|\dot{x}_j\|$.

A typical method of integrating stiff systems is to use the *slow* variable ($\dot{x}_j$) as a parameter in the equation for the fast dynamics ($\dot{x}_i$). Then one updates the fast dynamics at one rate and updates the slow dynamics at a slower rate.

The *Lsoda* ordinary differential integration package developed at the Lawrence Livermore Labs (see [Hindmarsh, 1983, Petzold, 1983]) is used. This package has automatic detection of stiff systems and switches integration algorithms accordingly.

6.2.3 Description of Dynamics

The user may describe the dynamics of his system by a simple *C* program which defines the state equations. A template is given below.

```
usr(init, x, xd, t, neq)

int
        init;
```

```
double
        x[], xd[], t;

int
        neq;                                                    10

{

register
        i;

        if (init)
          {
        / put initialization code here          /
          }                                                     20

        / compute xd here           /

        return;
}
```

The routine has five arguments. The first (*init*) is a flag variable that is *true* if it is the first call to the routine and allows the subroutine to initialize any variables. The second argument is the state vector x, which contains the current state of the system. xd is equivalent to $\dot{x}$ in (6.5) and t is the current time. Finally *neq* is the number of equations (currently this is fixed at three).

In summary the user is supplied with the current state, simulation time, and number of state equations and then must generate $\dot{x}$.

6.3 Computational aspects

The computation of the dynamics in three space can become quite large very quickly since we essentially grid up a cube of specified size (see figure 6.2) and compute the characteristics of each subcube based on the characteristics for a random point within the

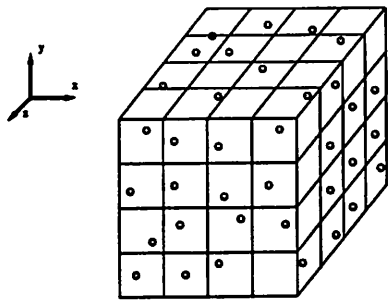

Figure 6.2: Gridding of the State Space

subcube (the necessity for the randomness will be explained later). Since the calculation of the dynamics in each subcube depends only on the subcube itself, we can carry on the computations in parallel.

With this in mind, a graphics server/computational client scheme was set up to offload the calculations. This also alleviated the need to write a full blown parser to gather the information necessary to define the system equations since some prewritten utilities could be used which would dynamically link in a subroutine.

These utilities exist for Sun workstations hence the logical choice of using the Sun as the computational server was made. The inherent graphics power of the Silicon Graphics series of workstation made it the obvious choice as the graphical server. The SGI workstation *talks* to the various Sun computational clients using the standard BSD[1] sockets library using the TCP[2] protocol over ethernet to disseminate information.

The steps to compute the stability characteristics may be listed as follows:

1. The server initiates communication to each client (max of 64).
2. The server downloads (to each client) a source file which describes the dynamics of the system.
3. Each client compiles and dynamically links in the routine and is ready to compute.
4. The server grids up the state space and divides it into *equal* portions for each client.
5. The clients integrate all initial conditions and assign to each a number representing whether a point is stable or unstable. If the point is unstable it is also assigned

[1]Berkeley Software Distribution, i.e. Unix with the Berkeley enhancements
[2]Internet Transmission Control Protocol

a number which represents how fast the trajectory is moving away from the initial condition.

6. Clients report back with results.

Note: if we are only changing the grid size/shape parameters and not the subroutine describing the system, then only steps 4 and 5 need be repeated.

One may argue that a better way to compute the region of attraction would be to start a cluster of points about an equilibrium point and integrate backwards in time. Then the boundary of the trajectories will eventually form the region of attraction. Since one must continually add points to fill in three space while the trajectories are dispersing, one may run into singularities in the decision tree on whether to branch or not. The computation time for this method is not necessarily faster and will vary widely depending on the system.

The problem of determining if a point is stable or not is not particularly well defined. For example, imagine a trajectory which initially leaves the region around the initial condition and then slowly comes back in. Currently *Sys_View* integrates the system for a user specified time and then checks to see if the trajectory at the end time is closer to the center point; the center point is a user defined point which defines the point to compute the stability characteristics about – the center point is assumed to be an equilibrium point. Thus the above problem may be alleviated if the end time was made large, but computational speed suffers as the end time is increased.

It would be easy to have *Sys_View* allow the user define the criterion for stability by having a *C* routine dynamically linked in to return whether a point is stable or not. Furthermore, the user could be allowed to increase the end time dynamically to alleviate the above problem in a more intelligent way.

6.3.1 Server-Client Communication

A simple language had to be defined for communication between the server and each client. These include commands to download code, receive a set of points and send back the results. Error and warning commands also had to be incorporated along with another set of commands to handle the computation of surfaces.

Extra effort was made to insure the connections be as robust as possible. Communication takes place asynchronously with a maximum latency response to a message of 1.5 seconds and is carried through at a maximum rate of 1 Mbit/sec. If a client for some reason

fails, the server is notified and redistributes the computation while closing the connection to the defunct client. In addition, if a client does not receive a command for some period of time (currently 30 minutes), the client reports to the server and terminates itself. This prevents *rogue* processes from running indefinitely. Furthermore, a client does not use any resources while it is waiting for a new set of computations.

Dynamic load scheduling is also used. This basically distributes the computational load *equally* amongst the clients. Prior to a computation each client is asked to return the time it took to compute a benchmark problem. This benchmark is included in each client as a subroutine and does not have to be sent nor is it linked in dynamically. This computation typically takes a couple of seconds and based on the returned result the server then prorates the number of points to be integrated. Hence if a machine is either inherently slow or is currently heavily loaded then its work will be accordingly scaled back.

Thus, with this setup we gained the ability to do parallel computation and had a simple way to specify the system equations (i.e. the equations could be described in a simple C or FORTRAN subroutine). In the current version there can be up to sixty four computational clients that would receive the subroutine, dynamically link it in and wait for commands to act on and report back to the server. A pictorial description of the setup is given in figure 6.3

A benchmark problem which required the integration of over 14,000 initial conditions on a system which was stiff and included trigonometric functions in its dynamics was completed in less than a minute on ten Sparc Station 1's as computational clients.

Since computations are essentially done in parallel we get an N times speed up, with hardly any overhead, and we again get the advantage of using dynamic linking to pull in the system description in the form of a C or FORTRAN subroutine.

6.4 Viewing the Results

One of the original ideas to view the data was to traverse along some axis and look at 2-D slices of the system (much like CAT scans). Multiple slices could be stored and in conjunction with transparency one could look past the initial slice to get more global information (i.e. we could store data slices of the $x - y$ plane for increasing z then if each plane is somewhat transparent we may see past the initial plane to the next few planes). This type of visualization may still be implemented but we feel that it would not be particularly

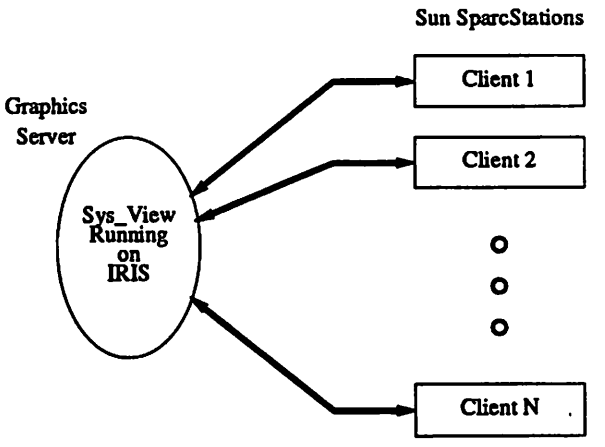

Figure 6.3: Graphics Server/Computational Client Set Up

easy to visually parse these images into a meaningful picture. that the global information lost would make this option not as attractive.

The route taken was to create a portrait cloud of points that was not fully dense (thus allowing us to see *through* things to view the behavior of the system at various points). This was accomplished by defining a random point constrained to be in a subcube of the gridded cube from figure 6.2. This allows us to see quite a bit of detail in the global aspect of the dynamics while still allowing us to view the local nature. Depth cueing is also available to give some depth perception in the visualization.

The randomness of the state space points was necessary to see the distant points. It should also be noted that it is much easier to view the data in an orthonormal projection then in a perspective projection.

A control panel was created as the main user interface (see figure 6.4). It allows the user to chose from the plethora of combinations given to view the data, and a complete description is given in the man pages for *sys-view*. A couple of features worth mentioning are the ability to record a sequence of button clicks in the control panel and then play them back for a sort of movie to traverse the escape portrait and secondly the ability to define your own color map for the escape portrait. Along with the control panel a plotting window to view the data is opened (see figure 6.5 for a black and white version).

The region of attraction for a system is the locus of initial conditions whose trajectories steer towards the origin and is valuable piece of information to compute for a system. The region of attraction may be simply plotted by using a predefined color map

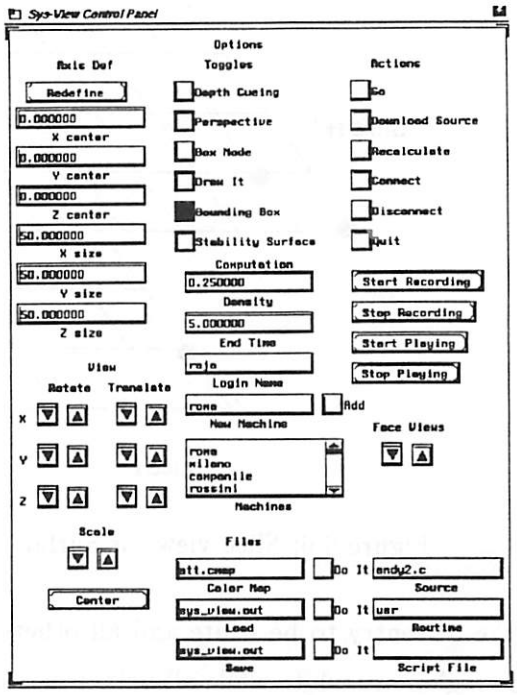

Figure 6.4: Sys-view Control Panel

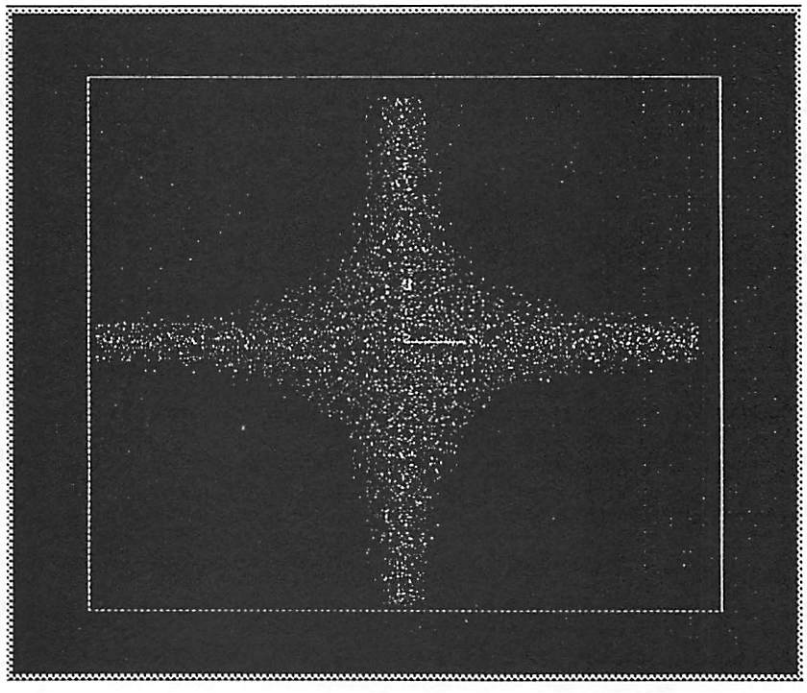

Figure 6.5: Sys-view Plotting Window

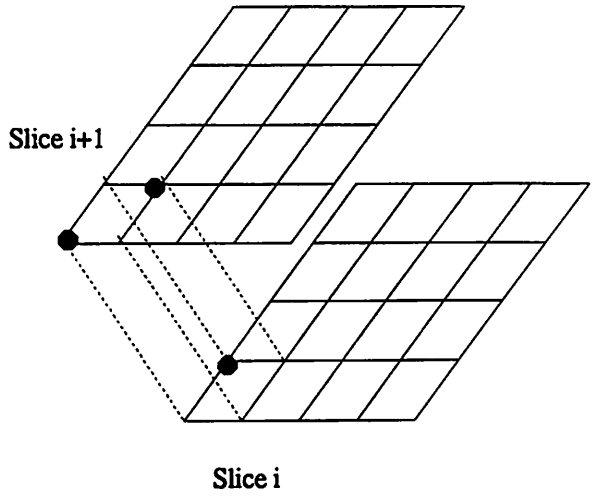

Figure 6.6: Slice view for Surface Reconstruction

which defines the 0^{th} entry to be white and all others to be black. Hence the points that do not leave the region are white while all points that do leave are black, thus giving us the region of attraction. One can play similar games for viewing all unstable points.

6.4.1 Creating Approximating Volume for the Region of Attraction

While the above solution for viewing the region attraction is quick, it is not the most visually pleasing. We would like to construct a a polyhedron which closely resembles the cloud of points.

Interestingly enough this problem is very similar to one investigated in the post processing of CAT, MRI, and SPECT scans in the medical information field. The problem facing them is to reconstruct surfaces based on density slices made through the body.

A CAT scan creates a two dimensional slice through the body with each slice containing a grid of numbers representing how *dense* that particular point was. Thus if we wanted to reconstruct a face or the bone structure of an individual we simply focus on the points which have the desired density.

This is virtually the same problem we face with the construction of a polyhedron which represents the region of attraction. We have two dimensional slices which contain data representing stability of a point and we wish to focus in on only the stable points.

The approach taken by [Lorenson and Cline, 1987], which is the algorithm we choose, was the so called marching cubes algorithm. In short one basically looks at two

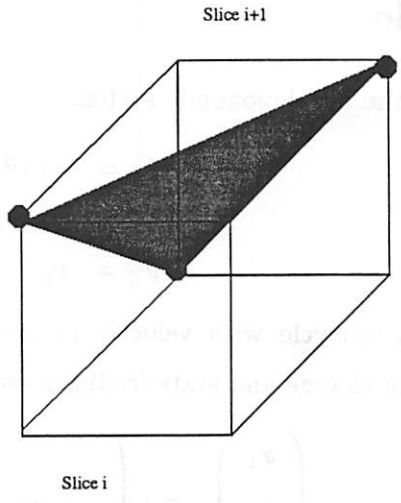

Figure 6.7: Marching Cube

slices of data at a time (see figure 6.6). We then restrict ourselves to look at eight vertices which form the so called *marching cube* as in figure 6.7. The three vertices define a face which constructs the local surface for the particular cube. We continue on repeating the process for each cube to construct the whole surface.

It turns out that if one factors in rotational symmetry and symmetry due to vertex complements (i.e. same configuration as case i except where there were points there are not and vice versa) there are only 14 unique configuration of the points within a cube. Thus we examine each cube in the data set and match it with one of the 14 basis cubes modulo rotational and complementary symmetry. Then retrieve the face structure for the basis cube and include it in the surface data structure with the rotation and complement operations applied.

It should be noted that these computations are carried out in a parallel fashion similar to the computation of the stability characteristics. The only drawback with this algorithm for our task at hand is that it may create an overabundance of triangles depending on the density of the point clouds. Alternatively, we do achieve an accurate approximation of the region of attraction.

6.5 Example

Consider the non-holonomic system

$$\begin{aligned} \dot{x} &= \cos(\theta)v_1 \\ \dot{y} &= \sin(\theta)v_1 \\ \dot{\theta} &= v_2 \end{aligned} \tag{6.6}$$

which represents a unicycle with velocity v_1 and angular velocity v_2. If we define the following coordinate change and state feedback (see [Teel *et al.*, 1992])

$$\begin{aligned} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} &= \begin{pmatrix} \theta \\ -x\cos(\theta) - y\sin(\theta) \\ -x\sin(\theta) + y\cos(\theta) \end{pmatrix} \\ \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} &= \begin{pmatrix} -u_2 + x_3 u_1 \\ u_1 \end{pmatrix} \end{aligned} \tag{6.7}$$

we get

$$\begin{aligned} \dot{x_1} &= u_1 \\ \dot{x_2} &= u_2 \\ \dot{x_3} &= x_2 u_1 \end{aligned} \tag{6.8}$$

We ask *Sys_View* to compute the region of attraction with the control being

$$\begin{aligned} u_1 &= -x_1 - {x_3}^2 \sin(t) \\ u_2 &= -x_2 + x_3 \cos(t). \end{aligned} \tag{6.9}$$

These set of equations may be included into a *C* subroutine with little effort as follows

```
#include <math.h>
#define x1      x[0]
#define x2      x[1]
#define x3      x[2]
#define xd1     xd[0]
#define xd2     xd[1]
#define xd3     xd[2]

usr(init, x, xd, t, neq)
int     init, neq;
double  x, xd, t;
```

```
{
register i;
double u1, u2;

        u1      = -x1 - x3x3sin(t);
        u2      = -x2 +x3cos(t);

        xd1     = u1;
        xd2     = u2;                                          20
        xd3     = x2u1;
}
```

On ten Sun SparcStations *Sys_View* took less than one minute to compute the stability characteristics for a cube spanning -50 to 50 in each dimension. With a density of 0.3 one obtains 27000 points to integrate to gather the stability characteristics. Once the calculations are made, one may compute the surface which approximates the region of attraction. This computation and the transfer of data from the computational client to the server was completed in less than 30 seconds and generated 5457 vertices, and 10910 triangular faces.

The results are shown in figure 6.8. The axis is defined by x_3 out of the page, x_2 pointing upwards, and x_1 pointing to the right. The view depicted in figure 6.8 is with a slight negative angle offset about the x_1 and x_2 axes. As one can see the region of attraction for this system is quite irregular, nonsymmetric and is definitely not what one would expect before hand.

One sees the ease at which *Sys_View* allows the the user to view the dynamics of a system. The only thing that needed to be provided is a simple *C* program which describes the dynamics.

6.6 Conclusions

A tool for viewing the dynamics of three dimensional continuous time systems has been developed in an interactive environment. With the addition of computational clients on remote machines the calculations necessary can be carried through relatively quickly. Possible additions for the future would be to include the CAT scan like slices

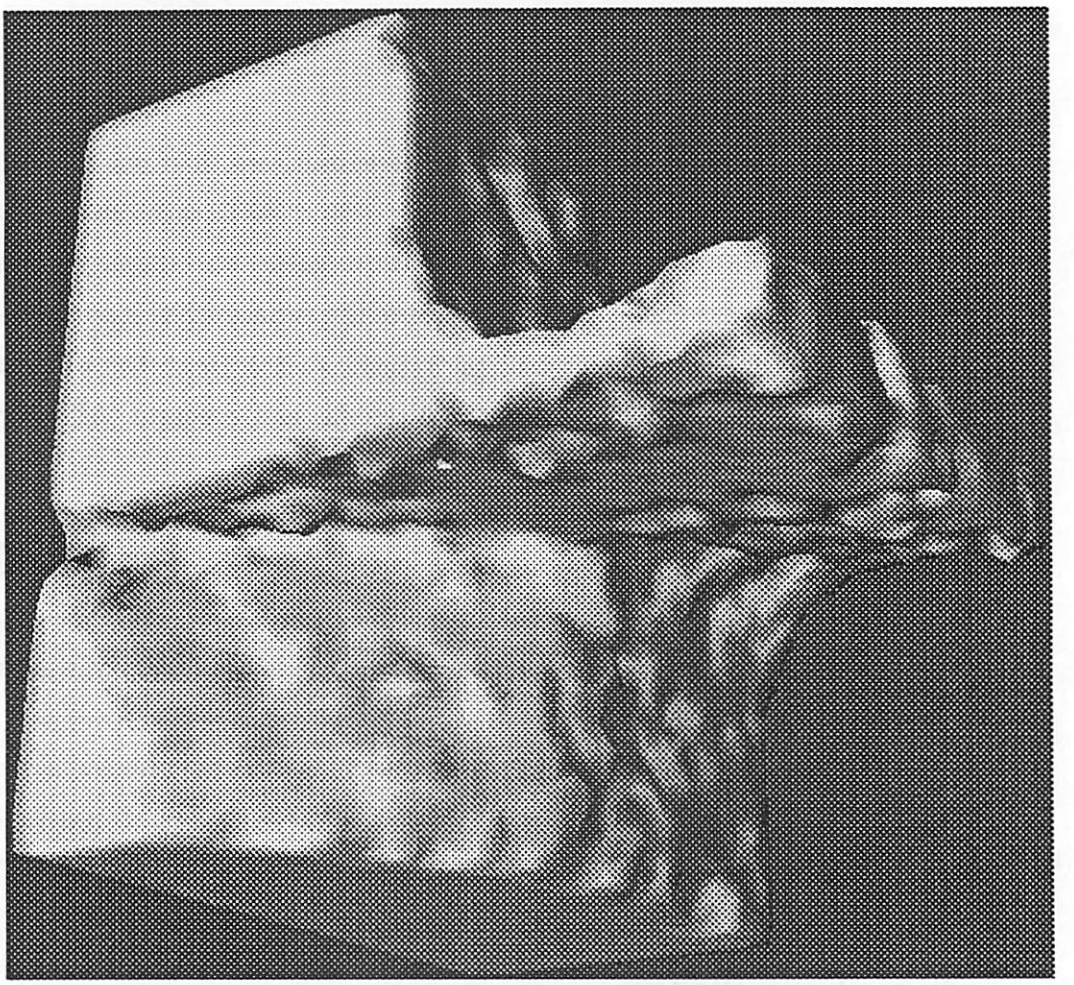

Figure 6.8: Region of Attraction

mentioned above as another possible viewing scheme. *Sys_View* combined with the *AP_LIN* package presented in Chapter 5 provides a powerful set of utilities for controlling and viewing nonlinear dynamics.

Bibliography

[Arnold, 1983] V. Arnold. *Geometrical Methods in the Theory of Ordinary Differential Equations.* Springer-Verlag, 1983.

[Bastin and Campion, 1989] G. Bastin and G. Campion. Indirect adaptive control of linearly parametrized nonlinear systems. In *3rd IFAC Symposium on Adaptive Systems in Control and Signal Processing, Glasgow, U.K.*, April 1989.

[Byrnes and Isidori, 1990] C. Byrnes and A. Isidori. Output regulation of nonlinear systems. *IEEE Transactions on Automatic Control*, 35, No.2:131–140, 1990.

[Campion and Bastin, 1989] G. Campion and G. Bastin. Indirect adaptive state feedback control of linearly parametrized nonlinear systems. Technical Report AP 89.09, Laboratoire D'Automatique, Dynamique et Analyse des Systèmes, Université Catholique de Louvain, 1989.

[Georgiou and Normand-Cyrot, 1989] G. Georgiou and D. Normand-Cyrot. De'couplage et linearisation adaptative d'un moteur à induction. Prepublication Report, CNRS-ESE, Paris, 1989.

[Hauser *et al.*, 1988] J. Hauser, S.S. Sastry, and G. Meyer. Nonlinear controller design for flight control systems. Technical Report UCB/ERL M88/76, Electronics Research Laboratory, University of California, Berkeley, 94720, 1988.

[Hindmarsh, 1983] A. C. Hindmarsh. Odepack, a systematized collection of ode solvers. In R. S. Stepleman et al., editor, *Scientific Computing*, pages 55–64. North-Holland, Amsterdam, 1983.

[Isidori, 1989] A. Isidori. *Nonlinear Control Systems: An Introduction.* Springer-Verlag, 1989.

[Kanellakopoulos *et al.*, 1989] I. Kanellakopoulos, P. V. Kokotovic, and R. Marino. Robustness of adaptive nonlinear control under an extended matching condition. In *Nonlinear Control Systems Design, Preprints of the IFAC Symposium, Capri, Italy*, June 1989.

[Krause, 1986] P.C. Krause. *Analysis of Electric Machinery.* McGraw-Hill, 1986.

[Kreisselmeier, 1977] G. Kreisselmeier. Adaptive observers with exponential rate of convergence. *IEEE Transactions on Automatic Control*, 22:2–8, 1977.

[Krener *et al.*, 1987] A. Krener, S. Karahan, M. Hubbard, and R. Frezza. Higher order linear approximations to nonlinear control systems. In *26th IEEE Conference on Decision and Control*, pages 519–523, December 1987.

[Krener *et al.*, 1991] A. Krener, M. Hubbard, S. Karahan, A. Phelps, and B. Maag. Poincaré 's linearization method applied to the design of nonlinear compensators. Technical report, Institute of Theoretical Dynamics, University of California at Davis, Davis, CA 95616, 1991.

[Krener, 1984] A. Krener. Approximate linearization by state feedback and coordinate change. *Systems and Control Letters*, 5, No. 3:181–185, December 1984.

[Krzeminski, 1987] Z. Krzeminski. Nonlinear control of induction motor. In *Proceedings of the 10th IFAC World Congress*, pages 349–354, December 1987.

[Kudva and Narendra, 1973] P. Kudva and K.S. Narendra. Synthesis of an adaptive observer using Lyapunov's direct method. *Int. Journal of Control*, 18:1201–1210, 1973.

[Lorenson and Cline, 1987] W. E. Lorenson and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Association of Computing Machinery, Computer Graphics*, 21, no. 4:163–169, 1987.

[Luca and Ulivi, 1987] A. De Luca and G. Ulivi. Full linearization of induction motors via nonlinear state-feedback. In *Proceedings of the 26th Conference on Decision and Control*, pages 1765–1770, December 1987.

[Luders and Narendra, 1973] G. Luders and K.S. Narendra. An adaptive observer and identifier for a linear system. *IEEE Transactions on Automatic Control*, 18:496–499, 1973.

[Marino *et al.*, 1990] R. Marino, S. Peresada, and P. Valigi. Adaptive partial feedback linearization of induction motors. In *Proceedings of the 29th Conference on Decision and Control*, pages 3313–3318, December 1990.

[Meyer, 1990] G. Meyer. Application of brunowsky forms in multi-mode flight control. In *1990 American Control Conference*, May 1990.

[Nath and Berg, 1981] G. Nath and G. Berg. Transient analysis of three-phase scr controlled induction motors. *IEEE Transactions on Industry Applications*, IA-17:133–142, 1981.

[Petzold, 1983] L. R. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *Siam Journal of Scientific Statistical Computing*, 4:136–148, 1983.

[Pomet and Praly, 1988] J. Pomet and L. Praly. Indirect adaptive nonlinear control. In *27th IEEE Conference on Decision and Control*, pages 2414–2415, December 1988.

[Sastry and Bodson, 1989] S.S. Sastry and M. Bodson. *Adaptive Control: Stability, Convergence, and Robustness.* Prentice Hall, 1989.

[Sastry and Isidori, 1987] S.S. Sastry and A. Isidori. Adaptive control of linearizable systems. Technical Report UCB/ERL M87/53, Electronics Research Laboratory, University of California, Berkeley, 94720, June 1987.

[Taylor *et al.*, 1989] D.G. Taylor, P.V. Kokotovic, R. Marino, and I. Kanellakopoulos. Adaptive regulation of nonlinear systems with unmodeled dynamics. *IEEE Transactions on Automatic Control*, 34:405–412, 1989.

[Teel *et al.*, 1991] A.R. Teel, R.R. Kadiyala, P.V. Kokotovic, and S.S. Sastry. Indirect techniques for adaptive input output linearization of nonlinear systems. *International Journal of Control*, 53, No. 1:193–222, 1991.

[Teel *et al.*, 1992] A.R. Teel, R.M. Murray, and G. Walsh. Nonholonomic control systems: Fromm steering to stabilization with sinusoids, 1992. To appear at the 31st Conference on Desision and Control.

[Verghese and Sanders, 1988] G. Verghese and S. Sanders. Observers for flux estimation in induction machines. *IEEE Transactions on Industrial Electronics*, pages 85–94, February 1988.

[Wette and Laub, 1986] M. Wette and A. Laub. Software practices in computer-aided control systems design: A need for tool-based systems. In *IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design*, pages 25–30, September 1986.

Appendix A

Manual Pages for the AP_LIN Package and Sys_View in Alphabetical Order

NAME

clean_param – clean up relatively small terms in polynomial strings in an *AP_LIN* configuration file

SYNOPSIS

clean_param [-c *cutoff* tolerance] [-f *file*]

DESCRIPTION

clean_param takes the output from the *AP_LIN* programs create_model(1), poincare(1), spline_hyper(1), spline_usr(1) and creates a new configuration file as described below.

OPTIONS

–c " cutoff tolerance;" Use *cutoff* tolerance as the tolerance to determine which variables are relatively small (compared to other elements in a single polynomial string) and should not be set to zero. The default is 1.0e-06.

–f " file;" Use the file named *file* as the file which contains the system configuration data created by create_model(1), poincare(1), spline_usr(1), spline_hyper(1)

SEE ALSO

create_model(1), poincare(1), spline_hyper(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

None known yet ...

NAME

config2latex – create latex from an *AP_LIN* configuration file

SYNOPSIS

config2latex [-c *cutoff* tolerance] [-i *input* file] [-o *output* file]

DESCRIPTION

config2latex takes the output from the *AP_LIN* programs create_model(1), linearize(1), poincare(1), spline_hyper(1), spline_usr(1) and then creates a latex file of the configuration file.

OPTIONS

–c " cutoff tolerance;" Use *cutoff* tolerance as the tolerance to determine which variables are relatively small (compared to other elements in a single polynomial string) and should not be printed. The default is 1.0e-06.

–i " input file;" Use the file named *input* file as the file which contains the system configuration data created by create_model(1), linearize(1), poincare(1), spline_usr(1), spline_hyper(1) The default is standard input.

–o " output file;" Use *output* file as the file to save to; the default is standard output.

SEE ALSO

create_model(1), linearize(1), poincare(1), spline_hyper(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

None known yet ...

NAME

create_model – multivariate spline fitting front end script

SYNOPSIS

create_model *fortran* file [-a] [-d] [-i *input* file] [-n *number* of points] [-o *output* file] [-p *power*] [-t] [-v]

DESCRIPTION

create_model takes a nonlinear model described by a MATRIXx HyperBuild file and creates a polynomial approximation of arbitrary order to the following system

$$\mathbf{x' = f(x) + G(x)u}$$
$$\mathbf{y = h(x)}$$

where f(x) and h(x) are vectors of polynomials and G(x) is a matrix of polynomials. The approximation for f(x) will contain all terms of order p and below except for order 0 terms, while the approximation for G(x) will contain all order (p-1) and lower terms. This behavior may be changed by using the 'all terms' option (see below). The spline fit is about some prescribed trajectory defined by the data in the variables t and u in the MATRIXx fsave'd file *input* file. The system is first simulated with the input specified in the input file and then knot points are uniformly picked as the points to use for the least square fit. If the input-output spline option is picked then the system is approximated by

$$\mathbf{y = h(u)}$$

and we must have the variables t and u in the input file. The output is then saved in the file *output* file which is **system.config** by default. This file may then be run through filters config2latex(1) to create latex of the approximate system or create_usr(1) to create a usr code file which may be simulated to check the validity of the approximation. Controllers may be created by using poincare(1) or linearize(1). **create_model** is actually a front end script to the actual spline routine, **spline_hyper.** The typical user will almost always spline fit models using **create_model.**

OPTIONS

fortran file; Use the file named *fortran* file as the source code file which contains the HyperBuild file to be spline fit. This argument is required.

–a Turns on all terms mode which will calculate all possible terms for f(x) and G(x).

–d Turns on debug mode which will print out more verbose information on what **create_model** is doing. This argument is optional.

–i " input file;" Use the file named *input* file as the file which contains the MATRIXx stored data of the trajectory to spline fit about. This file is created by the fsave command within MATRIXx and will contain the variables t and u. This argument is optional.

–n " num pts;" Use *num* pts as the number of knot points to be used in the spline fit. This argument is optional.

–o " output file;" Use *output* file as the file to save to; the default is system.config. This argument is optional.

–p " power;" Use *power* as the order of the polynomial fit. This argument is optional.

–t Time the spline operation. This argument is optional.

–v Do an input-output spline fit as described above. This argument is optional.

FILES

/tmp/tmp* temporary files created

SEE ALSO

clean_param(1), config2latex(1), create_usr(1), linearize(1), poincare(1), spline_hyper(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

The error checking for improper data is poor (trust is put in the user to use the software properly ...)

NAME

create_usr – create a C usr code block from an AP_LIN configuration file

SYNOPSIS

create_usr [-i *input* file] [-o *output* file]

DESCRIPTION

create_usr takes an AP_LIN configuration file and creates a C subroutine, in standard MATRIXx usr code block format. The created code is a subroutine representation of the AP_LIN configuration file. This subroutine may then be simulated to test the validity of the approximation.

OPTIONS

–i " input file;" Use the file named *input* file as the file which contains the AP_LIN configuration file The default is standard input.

–o " output file;" Use *output* file as the file to save to; the default is standard output.

SEE ALSO

create_model(1), linearize(1), poincare(1), spline_hyper(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

None known yet ...

NAME

linearize – input output linearize a system described by an *AP_LIN* configuration file

SYNOPSIS

linearize [-c *cutoff* tolerance] [-i *input* file] [-o *output* file] [-t]

DESCRIPTION

linearize takes the output from create_model(1), spline_usr(1), spline_hyper(1) and creates a controller that yields the original system input output linear

OPTIONS

–c " cutoff tolerance;" Use *cutoff* tolerance as the tolerance to determine which variables are relatively small (compared to other elements in a single polynomial string) and should not be set to zero. The default is 1.0e-06.

–i " input file;" Use the file named *input* file as the file which contains the system configuration data created by create_model(1), spline_usr(1), spline_hyper(1) The default is standard input.

–o " output file;" Use *output* file as the file to save to; the default is standard output.

SEE ALSO

create_model(1), poincare(1), spline_hyper(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

None known yet ...

NAME

poincare – reduce a system described by an *AP_LIN* configuration file to linear system up to arbitrary order

SYNOPSIS

poincare [-i *input* file] [-l *linearization* level] [-o *output* file] [-t]

DESCRIPTION

poincare takes the output from **create_model(1),** spline_usr(1), spline_hyper(1) and creates a controller that yields the original system linear up to an arbitrary order (see Krener et al., 1987 *26th* IEEE Conference on *Decision* and Control pages 519-523) for a description of the theory.

OPTIONS

–i " input file;" Use the file named *input* file as the file which contains the system configuration data created by **create_model(1),** spline_usr(1), spline_hyper(1) The default is standard input.

–l " linearization level;" Use *linearization* level as the order of linearity for the system (i.e. if we had *poincare* -l 3 then our resulting system with the control generated would be linear up through order 3 terms. The default is 2.

–o " output file;" Use *output* file as the file to save to; the default is the file **control.config**

SEE ALSO

create_model(1), linearize(1), spline_hyper(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

None known yet ...

NAME

spline_hyper – multivariate spline fitting routine

SYNOPSIS

spline_hyper [-a] [-d] [-f *fortran* file] [-i *input* file] [-n *number* of points] [-o *output* file] [-p *power*] [-s *number* of states] [-t] [-v] [-z *number* of tmps]

DESCRIPTION

spline_hyper takes a nonlinear model described by a MATRIXx HyperBuild file and creates a polynomial approximation of arbitrary order to the following system

$$\mathbf{x' = f(x) + G(x)u}$$
$$\mathbf{y = h(x)}$$

where f(x) and h(x) are vectors of polynomials and G(x) is a matrix of polynomials. The approximation for f(x) will contain all terms of order p and below except for order 0 terms, while the approximation for G(x) will contain all order (p-1) and lower terms. This behavior may be changed by using the 'all terms' option (see below). The spline fit is about some prescribed trajectory defined by the data in the variables t and u in the MATRIXx fsave'd file *input* file. The system is first simulated with the input specified in the input file and then knot points are uniformly picked as the points to use for the least square fit. If the input-output spline option is picked then the system is approximated by

$$\mathbf{y = h(u)}$$

and we must have the variables t and u in the input file. The output is then saved in the file *output* file which is **system.config** by default. This file may then be run through filters config2latex(1) to create latex of the approximate system or create_usr(1) to create a usr code file which may be simulated to check the validity of the approximation. Controllers may be created by using poincare(1) or linearize(1). It should be noted that the typical user will **never** use **spline_hyper,** but would use instead the front end shell script **create_model** which calls **spline_hyper** with the correct options. This command is only on the SUN

version and is not available on the DEC version. DEC users should use **create_model** instead.

OPTIONS

-a Turns on all terms mode which will calculate all possible terms for f(x) and G(x).

-d Turns on debug mode which will print out more verbose information on what **spline_hyper** is doing. This argument is optional.

-f " fortran file;" Use the file named *fortran* file as the source code file which contains the HyperBuild file to be spline fit. This argument is required.

-i " input file;" Use the file named *input* file as the file which contains the MATRIXx stored data of the trajectory to spline fit about. This file is created by the fsave command within MATRIXx and will contain the variables t and u. This argument is optional.

-n " num pts;" Use *num* pts as the number of knot points to be used in the spline fit. This argument is optional.

-o " output file;" Use *output* file as the file to save to; the default is system.config. This argument is optional.

-p " power;" Use *power* as the order of the polynomial fit; the default is 2. This argument is optional.

-s " num_states;" Use *num_states* to set the number of states in the model. This argument is required.

-t Time the spline operation. This argument is optional.

-v Do an input-output spline fit as described above. This argument is optional.

-z " num_tmps;" Use *num_tmps* as the number of temporary variables used by the system. This argument is required.

FILES

/tmp/tmp* temporary files created

SEE ALSO

clean_param(1), config2latex(1), create_model(1), create_usr(1), linearize(1), poincare(1), spline_usr(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

The error checking for improper data is poor (trust is put in the user to use the software properly ...)

NAME

spline_usr – multivariate spline fitting routine

SYNOPSIS

spline_usr [-a] [-c *code* file] [-d] [-i *input* file] [-k] [-n *number* of points] [-o *output* file] [-p *power*] [-r *routine* name] [-s *number* of states] [-t] [-u *number* of inputs] [-y *number* of outputs]

DESCRIPTION

spline_usr takes a nonlinear model described by a MATRIXx usr code file and creates a polynomial approximation of arbitrary order to the following system

$$\mathbf{x' = f(x) + G(x)u}$$
$$\mathbf{y = h(x)}$$

where f(x) and h(x) are vectors of polynomials and G(x) is a matrix of polynomials. The approximation for f(x) will contain all terms of order p and below except for order 0 terms, while the approximation for G(x) will contain all order (p-1) and lower terms. This behavior may be changed by using the 'all terms' option (see below). The spline fit is about some prescribed trajectory defined by the data in the variables t and u in the MATRIXx fsave'd file *input* file. The system is first simulated with the input specified in the input file and then knot points are uniformly picked as the points to use for the least square fit. If the input-output spline option is picked then the system is approximated by

$$\mathbf{y = h(u)}$$

and we must have the variables t and u in the input file. The output is then saved in the file *output* file which is **system.config** by default. If we are in data mode *(-d)* then the variables y and u must be in the input file and we simply spline fit the input output data without need for a code file.

The outfile system.config may then be run through filters config2latex(1) to create latex of the approximate system or create_usr(1) to create a usr code file which may be simulated to

check the validity of the approximation. Controllers may be created by using poincare(1) or linearize(1).

OPTIONS

–a Turns on all terms mode which will calculate all possible terms for f(x) and G(x).

–c " code file;" Use the file named *code* file as the source code file which contains the usr subroutine to be spline fit. This argument is required.

–d Turns on data mode creates an input output spline based on the data in the variables y and u in the input file. This argument is optional.

–i " input file;" Use the file named *input* file as the file which contains the MATRIXx stored data of the trajectory to spline fit about. This file is created by the fsave command within MATRIXx and will contain the variables t and u. This argument is optional.

–k Turns on state space fitting (i.e. specify which knot point to use in the least squares approximation. This argument is optional.

–n " num pts;" Use *num* pts as the number of knot points to be used in the spline fit. This argument is optional.

–o " output file;" Use *output* file as the file to save to; the default is system.config. This argument is optional.

–p " power;" Use *power* as the order of the polynomial fit; the default is 2. This argument is optional.

–s " num_states;" Use *num_states* to set the number of states in the model. This argument is required.

–t Time the spline operation. This argument is optional.

–u " num_inputs;" Use *num_inputs* as the number of inputs for the system. This argument is required.

–y " num_outputs;" Use *num_outputs* as the number of outputs for the system. This argument is required.

FILES

/tmp/tmp* temporary files created. $MATRIXX/src/usr01.c for template file for usr code subroutine.

SEE ALSO

clean_param(1), config2latex(1), create_model(1), create_usr(1), linearize(1), poincare(1), spline_hyper(1)

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

The error checking for improper data is poor (trust is put in the user to use the software properly ...)

NAME

sys_view – Interactively view a three dimensional nonlinear dynamic system

SYNOPSIS

sys_view [-b *box* mode] [-c *color* mapping] [-d *cloud* density] [-h *hush* mode] [-i -j -k *position* of bounding box] [-l *login* name to use on client machine] [-m *remote* machine] [-o *output* file] [-p *use* perspective projection] [-q *turn* on depth cueing] [-r *routine*] [-s *source_file*] [-t *end* time] [-x -y -z *dimensions* of bounding box]

DESCRIPTION

sys_view creates a three dimensional stability portrait of a dynamical system described by the subroutine in *source_file,* which may be either C or FORTRAN and is called by the name in *routine.* **sys_view** creates connections up to the machines specified through repetitive uses of the -m option (i.e. sys_view -m mach1 -m mach2 ...) and then downloads the file to the clients. The clients then dynamically link in this routine and compute equal portions of the trajectories. The results are then reported back to the server machine and a plot is created which the user may interactively view. Up to 64 machines may be used as clients.

OPTIONS

–b go into box mode which will draw cubes instead of pixels at the grid point. This is useful if a low density is used.

–c " color mapping;" Use the file contained in *color* mapping to define the colors to use in the escape portrait. This should be an ASCII file with each line containing four integers ranging from 0 to 255 specifying the index and the standard red, green, and blue (rgb) color values. (i.e. 0 12 87 39 would specify entry 0 to have a red value of 12, a green value of 87, and a blue value of 39). Typically entry 0 will be black and entry 255 will be white.

–d " density;" Use floating point value *density* as the parameter defining how dense of a pixel cloud to use when generating the escape portrait; valid range is 0 to 1, the default is 0.25 (anything larger than 0.6 will take more than a few minutes to run).

–h go into hush mode which limits the number of messages sent to the window which sys_view was initiated in.

–i –j –k " center of bounding box;" Use these three numbers to define the origin of the bounding box. This is useful for studying behavior away from the origin; the default is zero.

–l " login name;" If the account name on the remote machine is different than on the local machine then this option must be set accordingly. The remote machine should allow entry of the local account through an entry in .rhosts.

–m " machine;" Specifies which machines to run the calculàtions of the trajectories on. For a low density it is not necessary to specify too many machines. In fact this may hurt you since the overhead involved in the communication may slow things down. For large densities it will be well worth the effort to spread the computing across as many machines as possible. A maximum of 64 machines may be specified in the following fashion: sys_view -m mach -m mach2 ...

–o " output file;" Use *output* file as the file to save to; the default is sys_view.out.

–p " perspective projection;" This allows the user to specify a perspective projection be used as opposed to the default orthonormal projection.

–q " turn on depth cueing;" This will allow the user to gain some depth perception as points further away are darker than closer points. This will, however, slow down the redrawing of the plot in interactive mode.

–r " routine;" Use *routine* to tell **sys_view** the name of the routine to call to execute the system equations.

–s " source_file;" Use *source_file* as the file to dynamically link in to get the system equations.

–t " end time;" Use the floating point value *end* time as the length to numerically integrate the system before determining the stability characteristics; the default is 2.0 seconds.

–x –y –z " bounding box dimensions;" Use these three numbers to define the boundary of the plot. If not specified sys_view tries to set them automatically.

USER INTERFACE

The user interface may be divided into two sections. The first being the control panel and the second being the interactive or plot window. The control panel is partitioned into seven groups. The characteristics of the bounding box may be defined in the *Axis* Def group, while various attributes may be toggled on and off in the *Toggles* group. These attributes include depth cueing, perspective projection, box/cube mode, draw it, turn on and off the bounding box, and create a surface approximation for the region of attraction. This last toggle will create another graphics window with the surface displayed. The third subgroup is the *Actions* group which allows the user to start calculations, download a new source file, disconnect, reconnect to the clients and to quit. Note that this is the only way to quit sys_view. The *Computation* group allows the user to specify the density and end time and also allows the addition of a new machine to the client list which is also displayed. It is suggested that a disconnect occur before the addition of a new client followed by a connect. The next subgroup is the *Scripting* group which allows the user to record a series of button clicks thus creating a movie to be played back. Typically one records the actions from the *View* group which allows the user to rotate, translate, and scale the portrait in a precise manner. The *Face* Views section is a simple pair of up-down buttons which allows the user to view all six of the viewing cube's faces in an easy manner. The final group is the *Files* group which allows the user to specify the load/save files and the source and routine names.

The plot window allows for interactive viewing of the portrait much like the *View* group above, but in a less precise but faster manner. The left mouse button controls scaling, while the middle mouse button controls rotations and the right mouse button controls translations. All operations are made with the given mouse button down and moving the mouse on the pad in the appropriate direction. Z axis rotation and translation may be obtained by holding the shift key down and performing the normal rotation or translation. Pressing the c key will recenter the portrait.

Example C Program

The following is an example C program which will show the format necessary to be linked in and run by the remote computational server. The routine takes five arguments with the first being *init* which is true for the initialization call (ie. if init = 1 then the routine

should do any initialization it needs to do). The second argument is the state variable x which is of length *neq* (currently neq is three). The third argument is *xd* which is the return information for the system (ie. the user sets xd to the proper dynamics for the differential equation). The variable t represents the current simulation time.

```
#include <math.h>
usr(init, x, xd, t, neq)
int init, neq;
double *x, *xd;
{
if (init)
  {
    Initialize code here
  }
Compute dynamics here
}
```

AUTHOR

Raja R. Kadiyala, Department of Electrical Engineering and Computer Science, U.C. Berkeley. *email:* raja@robotics.berkeley.edu

BUGS

There is no error checking on the validity of the subroutine specified in source_file.