# CAD FOR NONLINEAR CONTROL SYSTEMS

by

Andrew Teel

Memorandum No. UCB/ERL M89/135

11 December 1989

# CAD FOR NONLINEAR CONTROL SYSTEMS

by

Andrew Teel

Memorandum No. UCB/ERL M89/135

11 December 1989

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering
University of California, Berkeley
94720

# CAD FOR NONLINEAR CONTROL SYSTEMS

by

Andrew Teel

Memorandum No. UCB/ERL M89/135

11 December 1989

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering
University of California, Berkeley
94720

# CAD for Nonlinear Control Systems *

Andrew Teel
Department of Electrical Engineering
and Computer Science
261M Cory Hall
University of California
Berkeley, CA 94720

December 11, 1989

**Abstract**

A computer software package has been developed to assist in the control design for a certain subclass of nonlinear systems. Much progress has been made in the area of nonlinear control recently. The design procedure that is a result of these advances has been automated in this software package. Both single-input single-output (SISO) and multi-input multi-output (MIMO) systems are handled.

## 1 Introduction

The first part of this report documents a computer software package that has been developed to assist in the control design for a certain subclass of nonlinear systems. Control design theory for nonlinear systems of the form

$$\begin{aligned} \dot{x} &= f(x) + g_1(x)u_1 + \cdots + g_m(x)u_m \\ y_i &= h_i(x) \end{aligned} \tag{1}$$

lends itself readily to concise algorithms that can be easily executed using symbolic manipulator software. Our package is written to run in a MACSYMA environment.

*Research supported in part by the Army under grant ARO DAAL-88-K0572, NASA under grant NAG2-243, and NSF under grant ECS 87-15811.

After the user has specified the dynamical equations of a system and certain control criteria, our package will specify the appropriate control law (under proper conditions) to achieve the desired performance. Also, all necessary information will be converted to fortran code which can then be easily used within simulation software (eg. MATRIXx) to test the performance of the chosen design.

Section 2 will detail the theory and differential geometric tools that serve as a basis for the design algorithms. Both the single input and multi input theory will be covered. This section will serve to summarize the major results of the theory. For a more detailed development, the reader should consult [1].

Section 3 documents the specifics of our software package. The general flow of the program will be explained along with what is expected from the user and what is generated by the program. Also, the purpose of each function and how it relates to the theory will be explained.

Section 4 will contain the documented MACSYMA code of all the machinery necessary to produce the nonlinear control design results.

In section 5 we will present a brief summary of this work along with the existing limitations of the package. Further, we will comment on features that might be appropriate to add in the future.

# 2 Basic Theory

Our design package for nonlinear control systems implements basic linearizing theory which we will review here (for more details see [1]).

## 2.1 SISO Systems

Consider a single-input single-output system

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{2}$$

with $x \in \Re^n$,$u \in \Re$ and $f$, $g$, $h$ smooth. Differentiating $y$ with respect to time, one obtains

$$\dot{y} = L_f h + L_g h u \tag{3}$$

Here $L_f h$, $L_g h$ stand for the Lie derivatives of $h$ with respect to $f$, $g$ respectively. If $L_g h(x) \neq 0 \ \forall x \in \Re^n$ then the control law

$$u = \frac{1}{L_g h}(-L_f h + v) \tag{4}$$

yields the linear system

$$\dot{y} = v. \tag{5}$$

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

$$\dot{y} = L_f^i h + L_g L_f^{i-1} h u \tag{6}$$

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

$$u = \frac{1}{L_g L_f^{\gamma-1} h(x)} (-L_f^\gamma h(x) + v) \tag{7}$$

yields

$$y^{(\gamma)} = v. \tag{8}$$

The preceding discussion assumed that the linearization conditions held in all of $\Re^n$. Some completeness conditions on vector fields involving $f$,$g$ are sufficient for this (for details see [1]).

We can use the discussion above to define $\gamma$ as the *strong relative degree* of system (2). (Cases where the relative degree is not defined, namely where $L_g L_f^{\gamma-1} h(x) = 0$ for some values of $x$, are not handled very elegantly in the theory and consequently are not dealt with in our package.)

For a system with a strong relative degree $\gamma$, it is easy to verify that at each $x^0 \in \Re^n$ there exists a neighborhood $U^0$ of $x^0$ such that the mapping

$$\Phi : U^0 \longrightarrow \Re^n$$

defined as

$$\begin{array}{rcl} \Phi_1(x) = \xi_1 = h(x) \\ \Phi_2(x) = \xi_2 = L_f h(x) \\ \vdots \\ \Phi_\gamma(x) = \xi_\gamma = L_f^{\gamma-1} h(x) \end{array} \tag{9}$$

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* ([1]) as

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 \\ &\vdots \\ \dot{\xi}_{\gamma-1} &= \xi_\gamma \\ \dot{\xi}_\gamma &= b(\xi,\eta) + a(\xi,\eta)u \\ \dot{\eta} &= q(\xi,\eta) \\ y &= \xi_1. \end{aligned} \tag{10}$$

In equation (10), $b(\xi,\eta)$ represents the quantity $L_f^\gamma h(x)$ and $a(\xi,\eta)$ represents $L_g L_f^{\gamma-1} h(x)$. We assume $x = 0$ is an equilibrium point of the system (ie. $f(0) = 0$) and we assume $h(0) = 0$. Then the dynamics

$$\dot{\eta} = q(0,\eta) \tag{11}$$

are referred to as the *zero-dynamics* (see [1] for details). The nonlinear system (2) is said to be minimum phase if the zero-dynamics are asymptotically stable. It is important to note that the condition

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

is not a necessary one for the diffeomorphism to be valid. It is merely required that the Jacobian matrix of the transformation, $d\Phi/dx$, is nonsingular at a point $x = x^0$. In general, this requirement will generate equations of the form

$$\begin{aligned} \dot{\xi}_1 &= \xi_2 \\ &\vdots \\ \dot{\xi}_{\gamma-1} &= \xi_\gamma \\ \dot{\xi}_\gamma &= b(\xi,\eta) + a(\xi,\eta)u \\ \dot{\eta} &= q(\xi,\eta) + p(\xi,\eta)u. \end{aligned} \tag{12}$$

Imposing the condition that the output remains at zero we find that

$$\xi(t) = 0 \quad \forall t$$

$$0 = b(0,\eta(t)) + a(0,\eta(t))u. \tag{13}$$

If we solve for $u(t)$ in (13) and substitute into the last equation of system (12) we have

$$\dot{\eta} = q(0,\eta) - p(0,\eta)\frac{b(0,\eta)}{a(0,\eta)} \tag{14}$$

which describes the zero-dynamics in the new coordinates chosen.

Because of the nature of a diffeomorphism, the stability characteristics of the zero-dynamics will be invariant with respect to the multitude of ways the diffeomorphism can be completed.

### 2.1.1 SISO Stabilization

We now apply the system of equations given by (12) and the minimum phase property to the stabilization problem. Suppose we choose the control law as

$$u = \frac{1}{a(\xi,\eta)}(-b(\xi,\eta) - c_0\xi_1 - c_1\xi_2 - \ldots - c_{\gamma-1}\xi_\gamma). \tag{15}$$

Under this control, system (12) becomes

$$\begin{aligned} \dot{\xi} &= A\xi \\ \dot{\eta} &= q(\xi,\eta) + \tfrac{p(\xi,\eta)}{a(\xi,\eta)}(-b(\xi,\eta) - c_0\xi_1 - c_1\xi_2 - \ldots - c_{\gamma-1}\xi_\gamma) \end{aligned} \tag{16}$$

with

$$A = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ -c_0 & -c_1 & -c_2 & \ldots & -c_{r-1} \end{bmatrix}$$

so that $A$ has the characteristic polynomial

$$p(s) = c_0 + c_1 s + \ldots + c_{\gamma-1}s^{\gamma-1} + s^\gamma \tag{17}$$

If (17) is chosen to be a Hurwitz polynomial then $\xi \to 0$ asymptotically. We know that

$$\dot{\eta} = q(0,\eta) - p(0,\eta)\frac{b(0,\eta)}{a(0,\eta)} \tag{18}$$

represents the zero-dynamics of the system. Consequently, we can conclude that if the zero-dynamics are asymptotically stable and (17) is a Hurwitz polynomial then (15) asymptotically stabilizes the system.

### 2.1.2 SISO Bounded Tracking

We now examine the tracking problem in a similar fashion. We desire to have $y(t)$ track $y_M(t)$. We start by choosing

$$u = \frac{1}{a(\xi,\eta)}(-b(\xi,\eta)+c_0(y_M-y)+c_1(\dot{y}_M-\dot{y})+\ldots+c_{\gamma-1}(y_M^{(\gamma-1)}-y^{(\gamma-1)})+y_M^{(\gamma)}) \tag{19}$$

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

$$\begin{aligned}
\dot{e} &= Ae \\
\dot{\eta} &= q(\xi,\eta) \\
&\quad + \tfrac{p(\xi,\eta)}{a(\xi,\eta)}(-b(\xi,\eta) + c_0(y_M - y) + \ldots + c_{\gamma-1}(y_M^{(\gamma-1)} - y^{(\gamma-1)}) + y_M^{(\gamma)}) \\
&= \tilde{q}(\xi,\eta) \\
\xi_i &= e_i + y_M^{(i-1)}
\end{aligned} \tag{20}$$

It is easy to see that this control results in asymptotic tracking and bounded state $\xi$ provided $y_M, \dot{y}_M, \ldots, y_M^{(\gamma-1)}$ are bounded. It can be proved further, using a converse Lyapunov approach, that $\eta$ remains bounded as well, assuming exponentially stable zero-dynamics and $\tilde{q}(\xi,\eta)$ is Lipschitz in $\xi, \eta$. Thus, under these conditions, this control yields bounded tracking. (See [2])

## 2.2 MIMO Systems

Multi-input multi-output systems fall into two categories. (The following discussion is parallel to the discussion found in [1].)

### 2.2.1 MIMO - Nonsingular Decoupling Matrix

For a certain subclass of MIMO systems, the control design procedure is a simple extension of the SISO procedure. This subclass is characterized by the following definition.

A multivariable nonlinear system has a *(vector) relative degree* $(r_1, \ldots, r_m)$ at $x^0$ if:

(i)

$$L_{g_j} L_f^k h_i(x) = 0$$

for all $1 \leq j \leq m$, for all $1 \leq i \leq m$, for all $k < r_i - 1$, and for all $x$ in a neighborhood of $x^0$,

(ii) the $m \times m$ matrix:

$$A(x) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1(x) & L_{g_2} L_f^{r_1-1} h_1(x) & \ldots & L_{g_m} L_f^{r_1-1} h_1(x) \\ L_{g_1} L_f^{r_2-1} h_2(x) & L_{g_2} L_f^{r_2-1} h_2(x) & \ldots & L_{g_m} L_f^{r_2-1} h_2(x) \\ \vdots & \vdots & \ldots & \vdots \\ L_{g_1} L_f^{r_m-1} h_m(x) & L_{g_2} L_f^{r_m-1} h_m(x) & \ldots & L_{g_m} L_f^{r_m-1} h_m(x) \end{bmatrix}$$

is nonsingular at $x = x^0$.

Observe that $r_i$ is equal to the number of times one must differentiate $y_i(t)$ before a component of the input appears explicitly. Consequently, $r_i$ can be thought of as the relative degree of the $i$th output of the system.

The assumption that $A(x)$ is nonsingular is a fairly restrictive one, but it allows us to proceed naturally to the normal form and zero-dynamics. If we assume the system has vector relative degree $(r_i, \ldots, r_m)$ and $r = r_1 + \ldots + r_m \leq n$, then it is easy to verify that at each $x^0 \in \Re^n$ there exists a neighborhood $U^0$ of $x^0$ such that the mapping

$$\Phi : U^0 \longrightarrow \Re^n$$

defined as

$$\Phi(x) = col[\phi_1(x), \ldots, \phi_{r_1}(x), \psi_1(x), \ldots, \psi_{r_2}(x), \ldots, \eta_1(x), \ldots, \eta_{n-r}(x)]$$

with

$$\begin{array}{rcl} \phi_1(x) & = & h_1(x) \\ \phi_2(x) & = & L_f h_1(x) \\ & \vdots & \\ \phi_{r_1} & = & L_f^{r_1-1} h_1(x) \\ \psi_1(x) & = & h_2(x) \\ \psi_2(x) & = & L_f h_2(x) \\ & \vdots & \\ \psi_{r_2} & = & L_f^{r_2-1} h_2(x) \\ & \vdots & \end{array} \tag{21}$$

and $\eta_i$ chosen so that $\Phi$ has a nonsingular Jacobian matrix at $x^0$.

To derive the normal form we will consider a system with two inputs and outputs. First we define

$$z = col[\phi_1, \phi_2, \ldots, \psi_1, \psi_2, \ldots, \eta_1, \ldots, \eta_{n-r}].$$

Then observe that

$$\begin{array}{rcl} \dot{\phi}_1 & = & \phi_2 \\ & \vdots & \\ \dot{\phi}_{r_1-1} & = & \phi_{r_1-1} \\ \dot{\phi}_{r_1} & = & L_f^{r_1} h_1(x) + L_{g_1} L_f^{r_1-1} h_1(x(t)) u_1(t) + L_{g_2} L_f^{r_2-1} h_1(x(t)) u_2(t) \end{array} \tag{22}$$

and similar expressions are obtained for $\dot{\psi}_1, \dot{\psi}_2, \ldots$. Note that the terms multiplying the inputs are the entries in the first row of the decoupling matrix. Since $x(t) = \Phi^{-1}(z(t))$ we have

$$A(z) = \begin{bmatrix} a_{11}(z) & a_{12}(z) \\ a_{21}(z) & a_{22}(z) \end{bmatrix} = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1(\Phi^{-1}(z)) & L_{g_2} L_f^{r_1-1} h_1(\Phi^{-1}(z)) \\ L_{g_1} L_f^{r_2-1} h_2(\Phi^{-1}(z)) & L_{g_2} L_f^{r_2-1} h_2(\Phi^{-1}(z)) \end{bmatrix} \tag{23}$$

and

$$b(z) = \begin{bmatrix} b_1(z) \\ b_2(z) \end{bmatrix} = \begin{bmatrix} L_f^{r_1} h_1(\Phi^{-1}(z)) \\ L_f^{r_2} h_2(\Phi^{-1}(z)) \end{bmatrix} \tag{24}$$

Then, in the normal form, the equations appear as:

$$\begin{aligned}
\dot{\phi}_1 &= \phi_2 \\
\dot{\phi}_2 &= \phi_3 \\
&\vdots \\
\dot{\phi}_{r_1-1} &= \phi_{r_1-1} \\
\dot{\phi}_{r_1} &= b_1(z) + a_{11}(z)u_1 + a_{12}(z)u_2 \\
\dot{\psi}_1 &= \psi_2 \\
\dot{\psi}_2 &= \psi_3 \\
&\vdots \\
\dot{\psi}_{r_2-1} &= \psi_{r_2-1} \\
\dot{\psi}_{r_2} &= b_2(z) + a_{21}(z)u_1 + a_{22}(z)u_2 \\
\dot{\eta} &= q(z) + p_1(z)u_1 + p_2(z)u_2 \\
y_1 &= \phi_1 \\
y_2 &= \psi_1.
\end{aligned} \tag{25}$$

Normal forms for systems with more inputs and outputs have the same structure.

We now proceed with determining the internal dynamics consistent with the constraint that the output functions $y_i(t)$ are identically zero for all time. We define

$$\xi = col[\phi_1, \phi_2, \ldots, \psi_1, \psi_2, \ldots, \ldots]$$

and

$$\eta = col[\eta_1, \eta_2, \ldots, \eta_{n-r}]$$

so that

$$z = (\xi, \eta).$$

Observe that this constraint is then written as

$$\xi(t) = 0 \quad \forall t$$

$$0 = y_i^{(r_i)}(t) = b_i(0, \eta(t)) + \sum_{i=1}^{m} a_{ij}(0, \eta(t)) u_j(t) \tag{26}$$

$$1 \leq i \leq m.$$

In vector notation

$$0 = b(0, \eta(t)) + A(0, \eta(t)) u(t). \tag{27}$$

Since $A(x)$ is nonsingular at $x = x^0$ by definition, the matrix $A(\xi, \eta)$ is nonsingular at $(\xi, \eta) = (0, 0)$ and (27) can be solved for $u(t)$ if $\eta(t)$ is close to zero, yielding

$$u(t) = -[A(0, \eta(t))]^{-1} b(0, \eta(t)) \tag{28}$$

The zero-dynamics are then

$$\begin{aligned} \dot{\eta}(t) &= q(0, \eta(t)) - p(0, \eta(t))[A(0, \eta(t))]^{-1} b(0, \eta(t)) \\ &= q_0(\eta(t)). \end{aligned} \tag{29}$$

Observe that in the original coordinates $x$, the necessary input is given by

$$u(t) = -[A(x(t))]^{-1} b(x(t)) \tag{30}$$

where $A(x)$ is the decoupling matrix in the definition of (vector) relative degree and $b(x)$ is the m-vector:

$$b(x) = \begin{bmatrix} L_f^{r_1} h_1(x) \\ L_f^{r_2} h_2(x) \\ \vdots \\ L_f^{r_m} h_m(x) \end{bmatrix}$$

In this expression for $u$, $x(t)$ is constrained to evolve on the subset

$$M = \{x : h_i(x) = L_f h_i(x) = \ldots = L_f^{r_i - 1} h_i(x) = 0, 1 \leq i \leq m\} \tag{31}$$

on which the zero-dynamics is defined.

This discussion of normal forms and zero-dynamics will be used later when we discuss noninteracting control for MIMO systems.

### 2.2.2 MIMO - Singular Decoupling Matrix

There exist two basic design philosophies for handling multivariable systems that do not have a (vector) relative degree. They are known as "preprocessing" and "postprocessing". Both strive to produce an extended system which does have a (vector) relative degree. This package implements a preprocessing algorithm when extension is necessary.

The principle behind preprocessing is to add integrators on the appropriate input channels to delay their appearing in the output. This operation is also known as dynamic extension, and the linearizing and decoupling feedback law is referred to as a dynamic state feedback law, because an extra state is generated in the compensator by each integrator that is added to the system.

An efficient algorithm for completing the appropriate extension for a multivariable system without a (vector) relative degree is given in [1] and is repeated here:

*1) From the given system information* $\{f(x), g(x), h(x)\}$, *construct the decoupling matrix* $A(x)$. *If the decoupling matrix is nonsingular at* $x^0$, *the procedure terminates.*

*2) Suppose the decoupling matrix has constant rank,* $p$, *at all* $x$ *near* $x^0$. *Find a square and nonsingular matrix* $\beta(x)$ *such that* $A(x)\beta(x)$ *has the following form:*

$$A(x)\beta(x) = \begin{bmatrix} * & * & \dots & * & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ * & * & \dots & * & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ * & * & \dots & * & 0 & \dots & 0 \end{bmatrix}$$

*The* $n_i x n_i$ *matrix* $\beta(x)$ *is chosen to annihilate the last* $n_i - p$ *columns of* $A(x)\beta(x)$, *and to impose that, in the first* $p$ *columns, a set of* $p$ *rows (whose position is not uniquely specified) coincide with the rows of the identity matrix.*

*Let* $j_1, \dots, j_q$ *denote the indices of those columns of* $A(x)\beta(x)$ *in which at least two entries are not identically zero. Note that* $q > 0$ *because, by construction, all rows of* $A(x)$ *are not identically zero (by definition of relative degree) and so are the rows of* $A(x)\beta(x)$.

*3) Modify the system by first changing $g(x)$ into $g(x)\beta(x)$ and then adding exactly one integrator on each of the inputs indexed by $j_1, \ldots, j_q$. It is easy to realize that this corresponds to implementing a dynamic state-feedback having the following form:*

*i) the dynamics of the feedback has dimension $q$ and is characterized by:*

$$\frac{d}{dt}\begin{bmatrix} z_1(t) \\ \vdots \\ z_q(t) \end{bmatrix} = \begin{bmatrix} v_{j_1}(t) \\ \vdots \\ v_{j_q}(t) \end{bmatrix}$$

*i.e. by $q$ independent integration channels.*

*ii) the input $u$ to the system is expressed, as a function of the state $x$, of the additional state variables $z_1, \ldots, z_q$, and of the new inputs $v_1, \ldots, v_{n_i}$ by:*

$$u(t) = \sum_{k=1}^{q} \beta_{j_k}(x(t)) z_k(t) + \sum_{j \notin \{j_1, \ldots, j_q\}} \beta_j(x(t)) v_j(t)$$

*where $\beta_j(x)$ denotes the $j$th column of $\beta(x)$. The system obtained by composing the original system with the dynamic state feedback is denoted by:*

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

*where*

$$\bar{x} = (x_1, \ldots, x_n, z_1, \ldots, z_q)$$

*4) Replace $\{f(x), g(x), h(x)\}$ by $\{\bar{f}(\bar{x}), \bar{g}(\bar{x}), \bar{h}(\bar{x})\}$ and return to step 1.*

Two important points should be made about this algorithm. First, the algorithm yields, in no more than $n$ iterations, a system with a decoupling matrix nonsingular at $x^0$ or no other dynamic state feedback exists yielding a system with a nonsingular decoupling matrix. In this sense, the algorithm is exhaustive. Second, the original system and the extended system have the same zero-dynamics. This is due to the fact that the zero-dynamics of a system are invariant with respect to the operations in the dynamic extension algorithm, namely the replacement of $g(x)$ by $g(x)\beta(x)$ and the addition of integrators on input channels. This fact allows us to calculate the zero-dynamics after the system has been extended when we can proceed as in the previous section with a nonsingular decoupling matrix.

In summary, this algorithm gets us back to the point where we can proceed with the control design as if the system naturally had a nonsingular decoupling matrix.

### 2.2.3 Noninteracting Control

The noninteracting control problem is to find a state feedback control law so that in the closed loop, each input channel controls one and only one output channel.

Consider a system with two inputs and two outputs and its normal form as given in (25). If the following feedback law is used:

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} a_{11}(z) & a_{12}(z) \\ a_{21}(z) & a_{22}(z) \end{bmatrix}^{-1} \begin{bmatrix} -b_1(z) + v_1 \\ -b_2(z) + v_2 \end{bmatrix} = A^{-1} \left[ -b(z) + \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \right] \tag{32}$$

the closed loop system assumes the form:

$$\begin{aligned}
\dot{\phi}_1 &= \phi_2 \\
\dot{\phi}_2 &= \phi_3 \\
&\vdots \\
\dot{\phi}_{r_1-1} &= \phi_{r_1-1} \\
\dot{\phi}_{r_1} &= v_1 \\
\dot{\psi}_1 &= \psi_2 \\
\dot{\psi}_2 &= \psi_3 \\
&\vdots \\
\dot{\psi}_{r_2-1} &= \psi_{r_2-1} \\
\dot{\psi}_{r_2} &= v_2 \\
\dot{\eta} &= q(z) - p(z)A^{-1}(z)b(z) + p_1(z)v_1 + p_2(z)v_2 \\
y_1 &= \phi_1 \\
y_2 &= \psi_1.
\end{aligned} \tag{33}$$

We see that the input $v_1$ controls only the output $y_1$ via a chain of $r_1$ integrators. Similarly, the input $v_1$ controls only the output $y_1$ via a chain of $r_2$ integrators. If $r_1 + r_2 < n$ the system contains an unobservable part. Namely, the $\eta$ states are effected by the inputs and all the states but do not affect the outputs. Regardless, the input/output behavior of the closed loop system has been made linear and is characterized by the following transfer

function matrix:

$$H(s) = \begin{bmatrix} \frac{1}{s^{r_1}} & 0 \\ 0 & \frac{1}{s^{r_2}} \end{bmatrix} \tag{34}$$

It is apparent that this discussion extends easily to systems with more that two inputs and two outputs.

### 2.2.4 MIMO Stabilization

We now examine the stability of a system that has been made noninteractive using state feedback. Suppose we choose our control inputs to impose the following additional feedback:

$$\begin{aligned} v_1 &= -c_0\phi_1 - c_1\phi_2 - \ldots - c_{r_1-1}\phi_{r_1} + w_1 \\ v_2 &= -d_0\psi_1 - d_1\psi_2 - \ldots - d_{r_2-1}\psi_{r_2} + w_2 \end{aligned} \tag{35}$$

Then the input/output behavior of the closed loop system is characterized by the transfer function matrix:

$$H(s) = \begin{bmatrix} \frac{1}{c_0+c_1 s+\ldots+c_{r_1-1}s^{r_1-1}+s^{r_1}} & 0 \\ 0 & \frac{1}{d_0+d_1 s+\ldots+d_{r_2-1}s^{r_2-1}+s^{r_2}} \end{bmatrix} \tag{36}$$

By choosing the $c_i's$ and $d_i's$ so that the corresponding polynomials are Hurwitz, one can render the input/output behavior stable.

The internal stability of the system, however, depends also on the behavior of the unobservable zero-dynamics. With $w_1$ and $w_2$ set to zero, the system reduces to a system of the form

$$\begin{aligned} \dot{\xi} &= \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \\ \dot{\eta} &= q(\xi,\eta) - p(\xi,\eta)A^{-1}(\xi,\eta)b(\xi,\eta) + p_1(\xi,\eta)v_1 + p_2(\xi,\eta)v_2 \end{aligned} \tag{37}$$

where $A_1$ and $A_2$ are Hurwitz matrices in canonical form. Consequently, $\xi \to 0$ as $t \to \infty$ and the second equation of (37) reduces to

$$\dot{\eta} = q(0,\eta) - p(0,\eta)A^{-1}(0,\eta)b(0,\eta) \tag{38}$$

which is exactly the zero-dynamics of the system. We see that, if the zero-dynamics are stable, the state feedback law has produced a linear noninteractive system that is internally stable.

### 2.2.5 MIMO Tracking

We now examine the MIMO tracking problem in a similar fashion. We desire to have $y_i(t)$ track $y_{iM}(t)$. We start by choosing in (32) the following additional feedback:

$$\begin{aligned} v_1 &= c_0(y_{1M} - y_1) + c_1(\dot{y}_{1M} - \dot{y}_1) + \ldots + c_{r_1-1}(y_{1M}^{(r_1-1)} - y_1^{(r_1-1)}) + y_{1M}^{r_1} \\ v_2 &= d_0(y_{2M} - y_2) + d_1(\dot{y}_{2M} - \dot{y}_2) + \ldots + d_{r_2-1}(y_{2M}^{(r_2-1)} - y_2^{(r_2-1)}) + y_{2M}^{r_2} \end{aligned} \tag{39}$$

Note that $y_1^{(i-1)} = \phi_i, y_2^{(i-1)} = \psi_i$. If we define $e_{1i} = y_1^{(i-1)} - y_{1M}^{(i-1)}$, $e_{2i} = y_2^{(i-1)} - y_{2M}^{(i-1)}$ then we have

$$\begin{aligned} \tfrac{d}{dt}\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} &= \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}\begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \\ \dot{\eta} &= q(\xi,\eta) - p(\xi,\eta)A^{-1}(\xi,\eta)b(\xi,\eta) + p_1(\xi,\eta)v_1 + p_2(\xi,\eta)v_2 \\ &= \tilde{q}(\xi,\eta) \\ \phi_i &= e_{1i} - y_{1M}^{(i-1)} \\ \psi_i &= e_{2i} - y_{2M}^{(i-1)} \\ \xi &= (\phi,\psi) \end{aligned} \tag{40}$$

It is easy to see from the equations above that this control results in asymptotic tracking and bounded state $\xi$ provided that the desired trajectories and their derivatives are bounded. It can be proved further, using a converse Lyapunov approach, that $\eta$ remains bounded as well, assuming exponentially stable zero-dynamics and $\tilde{q}(\xi,\eta)$ is Lipschitz in $\xi$,$\eta$. Thus, under these conditions, this control yields bounded tracking.

# 3 The Theory Automated

From our discussion of the basic linearizing theory, it is obvious that the control engineer has a powerful set of design tools for nonlinear systems affine in the input. However, it is apparent that the mathematical calculations involved in using such tools can be cumbersome. Consequently, it would be very useful to automate this machinery so the engineer can spend more time refining his design.

The purpose of our package is precisely to automate the determination of Lie derivatives, coordinate transformations, system relative degrees and zero-dynamics characteristics. Furthermore, this machinery has been coordinated into a module which progresses through the design procedure

systematically and requires minimal user input. Also, to save the design engineer more time, the details of programming the design result into Fortran code for simulation has been automated (but at the time of this writing, not yet tested.) Our package produces a Fortran program that can be easily plugged into simulation software such as MATRIXx to simulate the prescribed design. Because of the similarities in the SISO and MIMO system design theory, we were able to generalize one module to handle both cases.

## 3.1 Getting Started

To use this package, one must enter the MACSYMA environment, and load in the defined functions that make up our package. When first entering the MACSYMA environment, the system prompt will be "(c1)". (Subsequent system prompts will be in the same form with the number that follows "c" incremented each time.) All commands to MACSYMA and within the design module must end with a semi-colon. Loading the package is accomplished in MACSYMA by typing the following next to the system prompt:

**load("nlcontrol");**

After the package has finished loading, the complete design module can be accessed by typing:

**design();**

At this point the user will be interacting with the design module which will lead the user through the design procedure.

## 3.2 User Input

The first question the user will face is the following:

**Save a transcript of this session? ("filename;" or n;)**

> This give the user the option of saving everything that will appear on the screen as the design module is running. If a transcript is desired, the file name must be included in quotation marks and the line ended by a semi-colon. If a transcript is not desired, the user should type the letter *n* without quotation marks but followed by a semi-colon.

The design module begins by prompting the user for information concerning the uncontrolled nonlinear system. The module assumes the system takes the form

$$\begin{aligned} \dot{x} &= f(x) + g_1(x)u_1 + \cdots + g_m(x)u_m \\ y_i &= h_i(x) \end{aligned}$$

The user should be prepared to respond to the following prompts to provide the system the information necessary to begin:

**Type "y;" to retrieve previously saved equations. Otherwise type "n;"**

Each time the design module is run, the system of equations is saved in a file called "systemeq". If you wish to redesign your control for the last system you enter using different design parameters, you can avoid reentering the system information by typing "y;".

**A square plant is required (# of inputs = # of outputs)**
**enter the number of inputs/outputs:**

If the system is SISO, type "1;". Otherwise type the number of inputs followed by a semi-colon.

**enter the dimension of the state space:**

This is simply the number of state variables of the system.

**enter the column vector f as an (n x 1) matrix:**

You will be prompted for each entry individually. Enter the (1,1) element followed by a semi-colon. The module will then ask to the (2,1) entry. (Matrix representation makes internal manipulation more straightforward.) Note also that the program assumes you will use state variables specifically denoted as $x1, \ldots$.

**enter the column vectors g as an (n x ni) matrix:**

Again, you will be prompted for each entry individually.

**enter the scalar output h:**
**enter the output functions as an (no x 1) matrix:**

Only one of the two proceeding prompts will be printed depending on whether the system is SISO or MIMO.

**Type "y;" for an operating point other than 0. Otherwise type "n;"**

Linearizing design theory is a local theory and, therefore, requires an operating point and a corresponding neighborhood on which the theory is valid.

**enter the operating point x0 as an (n x 1) matrix starting with the value of x1....:**

This prompt will only be printed if the operating point is being changed from zero.

## 3.3 Program Output

Once equipped with the necessary information provided by the user, the module proceeds to call upon various functions to carry out the necessary algebraic manipulations.

The first thing that is determined is the system relative degree. If the system is SISO or MIMO with a nonsingular decoupling matrix, then the relative degree is well-defined. If the system is MIMO with a singular decoupling matrix, then a preprocessing algorithm is applied to the system to generate an extended system that has a well-defined (vector) relative degree.

Next, a valid diffeomorphism between the original states and the states of the normal form is obtained. No effort is applied to assure that the inputs do not enter explicitly in the expression for the dynamics of the $\eta$ coordinates. This would be time consuming and unnecessary. The $\eta$ components of the transformation can be chosen arbitrarily as long as the Jacobian matrix of the transformation is nonsingular at the operating point. This is most easily accomplished using the following algorithm:

*1) Produce an (r x n) matrix where the codistributions* $dh_i, dL_f h_i, \ldots, dL_f^{r_i} h_i$ *are the r rows of the matrix. (It is a well known fact that the rank of this matrix is r).*

*2) Set* $i = 1, j = 1$.

*3) Append to the matrix in step 1 a row vector containing a "1" in the jth column and "0" everywhere else.*

*4) Determine the rank of the augmented matrix.*

*5) If augmenting the row in step 3 increased the rank of the matrix, then set $\eta_i = x_i$ and increment $i$.*

*6) Increment $j$ and goto step 3.*

If the system is not fully state linearizable then the stability of the zero-dynamics must be checked. Because each of the $\eta$ coordinates was chosen equal to a different original state, the zero-dynamics corresponding to the $\eta$ variables can be found in the following way:

*1) Find the dynamic equation for the corresponding original state variable.*

*2) Substitute the unique input that holds the output(s) at zero into the dynamic equation found in step 1.*

*3) Determine the inverse transformation.*

*4) Replace the original state variables with equivalent expressions in the new state variables. These expressions follow from the inverse transformation.*

*5) Set to zero the output(s) and the corresponding $r_i - 1$ derivatives of the output(s). (ie. all of the new state variables except the $\eta_i$'s.)*

The stability of the zero-dynamics is then checked using a first order (Jacobian) approximation. If the eigenvalues of the linear approximation are all located in the left half plane (LHP) then the zero-dynamics are exponentially stable and the module proceeds. If the zero-dynamics are unstable or the stability cannot be readily determined because those eigenvalues not in the LHP lie on the imaginary axis, the module will want to stop.

**Terminate design procedure? (y; or override;)**

The user can override this decision by typing

**override;**

at this point. This may be done when the user has simulated the zero-dynamics elsewhere and knows that they are asymptotically stable and/or the user just wants to see simulation results under such conditions. In either case, the stability claims of the subsequent controller should be taken with a grain of salt and the simulation results should be carefully scrutinized. If the zero-dynamics are stable, the module will produce a general feedback law to render the input-output relationship linear.

If the system was fully state linearizable, the module will produce a general feedback law that, in conjunction with the state transformation, renders the entire system linear.

Next, the user is given the opportunity to specify the control law as an output tracking control law or as a stabilizing control law.

**Is the control objective to stabilize or track? (s; or t;)**

If the control objective is tracking, the desired trajectory will need to be specified by the user following the prompt:

**enter desired time trajectory for output # $i$**

In both the stabilizing and tracking cases the user will need to supply pole location information for the dynamics of the linear portion of the system. For a MIMO system, because the control law decouples the system as well, the user will be asked for the pole locations of each subsystem (associated with the different inputs) separately. As an example, the prompt may be:

**Enter the $r_i$ desired LHP pole location(s)**
**for the subsystem associated with the input $i$**
**in the form $[p_1, \ldots, p_{r_i}]$;**

From the pole locations, the module produces the coefficients of the corresponding Hurwitz polynomial and uses these coefficients to specify $v$ specifically.

Note that, especially for MIMO systems, the screen display can become quite cluttered when trying to display a lengthy control law. In this case it can be useful to be aware of the following system variables:

| | |
|---|---|
| 1) a_dc | decoupling matrix |
| 2) a_dcinv | inverse of decoupling matrix |
| 3) d_ | determinant of decoupling matrix |
| 4) b_ | vector of Lie derivatives used in linearizing control law |
| 5) u_cl | linearizing control law |
| 6) v_ | component of control law to stabilize or track a trajectory |

Any of these variables can be looked at independently, after the module has finished, by typing the variable name and a semi-colon after the system prompt. For example:

(c10) **a_dc;**

The individual entries of each of these variables (except for the scalar d_) can be assessed by referencing the variable as a matrix:

**(c10) a_dc[1,1];**

If an individual entry is unruly, the user can attempt to simplify it in an ad hoc manner.

Finally, if the user desires, the module will transfer all of the system and control information to a Fortran program that can be used with MATRIXx to simulate the design results.

**Translate results into Fortran code for simulation? (y; or n;)**

The user will have to supply the name of the file in which to store the Fortran code.

**Enter a filename for the Fortran code:**

Be sure to use a filename in the following form and include the filename in quotations:

**"filename.f";**

## 3.4 Using Individual Functions Separately

The functions that make up this package are not designed to be used independently, however, it may be useful to know what they do independently.

**set_eqs()**

Prompts the user to enter the system of equations or loads them from a file. Each time this function is called, the user specified system of equations is saved to a file called "systemeq". The operating point is automatically set to zero.

*Variables used:* none.

*Variables affected:* f_,g_,h_,no_,ni_,ns_,x_,z_,u_,v_,w_,x0_.

**op()**

Allows the operating point to be set to something other than zero.

*Variables used:* none.

*Variables affected:* x0_.

**v_reldeg()**

Determines the preliminary vector relative degree without checking for the nonsingularity of the decoupling matrix.
*Variables used:* f_,g_,h_.
*Variables affected:* r_.

**snglr_chk()**

Produces the decoupling matrix and determines whether it is nonsingular.
*Variables used:* f_,g_,h_,r_.
*Variables affected:* a_dc,d_.

**nonsing_dc()**

Produces the inverse of the decoupling matrix and affirms the relative degree.
*Variables used:* a_dc.
*Variables affected:* a_dcinv,b_.

**preprocess()**

Carries out the preprocessing algorithm to produce a dynamic compensator that can be used to linearize the system.
*Variables used:* a_dc,f_,g_,ni_,ns_.
*Variables affected:* f_,g_,ns_,x_,z_.

**diffeo()**

Determines the diffeomorphic coordinates transformation.
*Variables used:* f_,h_,r_,z_,no_,ns_,x0_.
*Variables affected:* phi.

**control_law()**

Generates the general linearizing control law.
*Variables used:* a_dcinv,b_,v_,d_.
*Variables affected:* u_cl.

**check_zerodyn()**

Determines zero dynamics and analyzes zero dynamics in the first order approximation.

*Variables used:* f_,g_,h_,phi,rtotal_,a_dcinv,b_,ns_,x_,z_.

*Variables affected:* status.

**stabilize()**

Specifies components of the control law to allow for regulation to zero.

*Variables used:* no_,r_.

*Variables affected:* ym,w_

**track()**

Specifies components of the control law to allow for tracking.

*Variables used:* no_r_.

*Variables affected:* ym,w_

**poles()**

Compiles the specific control law from user specified control criteria.

*Variables used:* ni_,r_,ym,phi,w_.

*Variables affected:* v_

**make_fortran()**

Writes fortran code to a file to enable later simulation.

*Variables used:* ns_,ni_,no_,x_,h_,v_,u_cl,f_,g_,u_.

*Variables affected:* none.

**jacobian(v)**

Given any column vector with number of rows equal to the number of states, will determine the jacobian matrix of the vector.

*Variables used:* ns_,x_.

*Variables affected:* none.

**lderiv(v,h)**

Given a column vector and a scalar output, will determine the Lie derivative of the output with respect to the vector field specified by the column vector.

*Variables used:* ns_,x_.

*Variables affected:* none.

**klderiv(v,h,k)**

Given a column vector, a scalar output, and an integer, will determine the kth Lie derivative of the output with respect to the vector field specified by the column vector.

*Variables used:* ns_,x_.

*Variables affected:* none.

# 4 MACSYMA Code

```
/*********************************************************************/
/* Set flags and preload needed vaxima routines                      */

loadprint:false$
gcprint:false$
ratprint:false$
load("das/mstuff")$
load("share2/lrats")$
load("macrak/rpart")$
load("rat/result")$
resultant:subres$

/* ******************************************************************/
/* Global variables:                                                 */
/*   f_,g_,h_,f_old,g_old                                            */
/*   ni_,no_,ns_                                                     */
/*   x_,x0_,z_,u_,v_,w_                                              */
/*   r_,rtotal_                                                      */
/*   a_dc,a_dcinv,d_,b_,u_cl                                         */

design():=
  block(
    transcript:read("Save a transcript of this session? (\"filename\"\; or n\;)"),
    if transcript#n then
        apply(writefile,[transcript]),

    set_eqs(),
    op(),

    loop,
    v_reldeg(),
    check_s:snglr_chk(),
    if check_s=singular then
       (
        status:preprocess(),
        if status=failed then
```

```
      go(endflag)
    else
      go(loop)
    )
else
    nonsing_dc(),

diffeo(),
rtotal_:rd_sum(),

if rtotal_=ns_ then
  full_linear()
else
 (
  results:check_zerodyn(),
  if results#stable then
   (
    termd:read("Terminate design procedure? (y\; or override\;)"),
    if termd=override then
       print("")
    else
       go(endflag)
    ),
  io_linear()
  ),

obj:read("Is control objective to stabilize or track? (s\; or t\;)"),
if obj=s then
  stabilize()
else
  track(),
poles(),

simu:read("Translate results to Fortran for simulation? (y\; or n\;)"),
if simu=y then
  make_fortran(),

endflag,
if transcript#n and simu#y then
```

```
      apply(closefile,[transcript]),
    end
    )$

/* ********************************************************************/
/* Subroutine to establish system equations                           */

set_eqs():=
  block([i],
    oldeq:read("Type \"y\;\" to retrieve previously saved equations.  Otherwise ty
    print(""),
    if oldeq=y then
       loadfile(systemeq)
    else
      (
       print("A square plant is required (# of inputs = # of outputs)"),
       ni_:read("enter the number of inputs/outputs:"),
       no_:ni_,
       ns_:read("enter the dimension of the state space:"),
       print("enter the column vector f as an (",ns_," x 1) matrix:"),
       f_:entermatrix(ns_,1),
       print(""),
       print("enter the column vectors g as an (",ns_," x ",ni_,") matrix:"),
       g_:entermatrix(ns_,ni_),
       if ni_=1 then
         (
          htemp[1,1]:read("Enter the scalar output h:"),
          h_:genmatrix(htemp,1,1)
          )
       else
         (
          print("enter the output functions as an (",no_," x 1) matrix:"),
          h_:entermatrix(no_,1)
          ),
       save(systemeq,f_,g_,h_,no_,ni_,ns_)
       ),

    x_:transpose(makelist(concat('x,i),i,1,ns_)),
    z_:transpose(makelist(concat('z,i),i,1,ns_)),
```

```
    u_:transpose(makelist(concat('u,i),i,1,ni_)),
    v_:transpose(makelist(concat('v,i),i,1,ni_)),
    w_:transpose(makelist(concat('w,i),i,1,ni_)),

    x0_:zeromatrix(ns_,1),

    print(""),
    print("The system is described by:"),
    print(""),
    print('diff(x_,t),"=",f_,"+",g_,"*u"),
    print("y =",h_),
    print("")
    )$

/* ********************************************************************/
/* Subroutine to establish operating point                            */

op():=
  block([newop],
    newop:read("Type \"y\;\" for an operating point other than 0.  Otherwise type
    if newop=y then
       (
        print("enter the operating point x0 as an (",ns_," x 1) matrix,"),
        print("starting with the value of x1....  :"),
        x0_:entermatrix(ns_,1),
        print("")
        )
    )$

/* ********************************************************************/
/* Subroutines to determine (vector) relative degree                  */

v_reldeg():=
  block([i,rt],
    print("Determining the system relative regree..."),

    for i:1 thru no_ do
      (
```

```
      rt[1,i]:reldeg(f_,g_,h_[i,1]),
      print(" ")
      ),
    r_:genmatrix(rt,1,no_),
    print("")
    )$

reldeg(f_,g_,h_):=
  block([j,i,a,b],
    b:zeromatrix(1,ni_),
    j:0,
    while b = zeromatrix(1,ni_) do
      (
       j:j+1,
       for i:1 thru ni_ do
          a[1,i]:fullratsimp(lderiv(col(g_,i),klderiv(f_,h_,j-1))),
       b:genmatrix(a,1,ni_)
       ),
    return(j)
    )$

/* ********************************************************************/
/* Subroutines to assure that decoupling matrix is nonsingular    */
/* so that (vector) relative degree is defined.  This may         */
/* require preprocessing.                                         */

snglr_chk():=
  block(
    a_dc:decouple_m(f_,g_,h_,r_),
    d_:fullratsimp(determinant(a_dc)),
    if d_=0 then
      return(singular)
    else
      return(nonsingular)
    )$

nonsing_dc():=
  block([i,temp],
    a_dcinv:a_dc^^-1,
```

```
    for i:1 thru ni_ do
       temp[i,1]:klderiv(f_,h_[i,1],r_[1,i]),
    b_:genmatrix(temp,ni_,1),

    print("The relative degree of this system is ",r_),
    print("")
    )$

decouple_m(f_,g_,h_,r_):=
  block([i,j],
    for i:1 thru ni_ do
      for j:1 thru no_ do
        a[i,j]:fullratsimp(lderiv(col(g_,j),klderiv(f_,h_[i,1],r_[1,i]-1))),
    genmatrix(a,no_,ni_)
    )$

preprocess():=
  block([p,a_dc_aug,axbx,bx,i,ptest,bad,mflag,j,status,k,
         nonzero,jindex,f_bar,g_bar,newcolumn,m],
    print("The decoupling matrix is singular"),
    print("Determining dynamic extension..."),
    p:rank(a_dc),
    a_dc_aug:addcol(transpose(a_dc),ident(ni_)),
    axbx:transpose(echelon(a_dc_aug)),
    bx:axbx,
    for i:1 thru ni_ do
      (
       axbx:submatrix(ni_+1,axbx),
       bx:submatrix(1,bx)
       ),

    ptest:0,
    bad:0,
    for i:1 thru ni_ do
      (
       mflag:ZEROES,
       for j:1 thru ni_ do
          if j>p and axbx[i,j]#0 then
            bad:bad+1
```

```
      else
        if axbx[i,j]#0 and axbx[i,j]#1 then
          mflag:NOTID
        else
          if mflag=ZEROES and axbx[i,j]=1 then
            mflag:ONE
          else
            if mflag=ONE and axbx[i,j]#0 then
              mflag:NOTID,

   if mflag=ONE then ptest:ptest+1,
   if mflag=ZEROES then bad:bad+1
   ),

if ptest#p or bad#0 then
  (
   print("Algorithm didn't work"),
   status:failed,
   go(endflag2)
   ),

k:0,
for j:1 thru p do
  (
   nonzero:0,
   for i:1 thru ni_ do
     if axbx[i,j]#0 then nonzero:nonzero+1,
   if nonzero>1 then
     (
      k:k+1,
      jindex[k]:j
      )
   ),

if k=0 then
  (
   print("Algorithm didn't work"),
   status:failed,
   go(endflag2)
```

```
   ),

status:success,
x_:transpose(makelist(concat('x,i),i,1,ns_+k)),
z_:transpose(makelist(concat('z,i),i,1,ns_+k)),

f_bar:f_,
for i:1 thru k do
   f_bar:f_bar+(g_.col(bx,jindex[k]))*concat('x,ns_+i),
for i:1 thru k do
  (
   f_bar:addrow(f_bar,[0]),
   x0_:addrow(x0_,[0])
   ),

j:1,
g_bar:zeromatrix(ns_+k,1),
for i:1 thru ni_ do
  (
   if i=jindex[j] then
      (
       newcolumn:zeromatrix(ns_,1),
       for m:1 thru j-1 do
          newcolumn:addrow(newcolumn,[0]),
       newcolumn:addrow(newcolumn,[1]),
       for m:1 thru k-j do
          newcolumn:addrow(newcolumn,[0]),
       j:j+1
       )
   else
      (
       newcolumn:g_.col(bx,i),
       for m:1 thru k do
          newcolumn:addrow(newcolumn,[0])
       ),
   g_bar:addcol(g_bar,newcolumn)
   ),
g_bar:submatrix(g_bar,1),
```

```
    f_old:f_,
    g_old:g_,
    f_:f_bar,
    g_:g_bar,

    ns_:ns_+k,

    print(""),
    print("The extended system has",k,"additional states"),
    print("and is described by:"),
    print(""),
    print('diff(x_,t),"=",f_,"+",g_,"*u'"),
    print("y =",h_),
    print(""),
    print("This system is produced by setting u=",bx,"*u'"),
    print("and adding integrators on input channel(s):"),
    print(makelist(jindex[i],i,1,k)),
    endflag2,
    return(status),
    print("")

    )$

/* ****************************************************************/
/* Subroutines for diffeomorphic coordinates transformation       */

diffeo():=
  block(
    phi:basis(f_,h_,r_),
    print("The coordinates transformation is given by",z_=phi),
    print("")
    )$

basis(f_,h_,r_):=
  block([b,db,i,j,k,l,m],
    k:0,
    for i:1 thru no_ do
       for j:1 thru r_[1,i] do
         (
```

```
        k:k+1,
        b[k,1]:klderiv(f_,h_[i,1],j-1)
        ),

    for i:k+1 thru ns_ do
       b[i,1]:0,

    if k<ns_ then
      (
       db:jacobian(genmatrix(b,ns_,1)),
       db:evaluate(db,x0_),
       for i:1 thru ns_-k do
          db:submatrix(k+1,db),

       l:0,
       for m:k+1 thru ns_ do
         (
         i:1+l,
         for j:1+l while rank(addrow(db,ematrix(1,ns_,1,1,j)))=rank(db) do
            i:j+1,
         b[m,1]:concat('x,i),
         db:addrow(db,ematrix(1,ns_,1,1,i)),
         l:i
         )
       ),

    genmatrix(b,ns_,1)
    )$


rd_sum():=
  block([rtl,i],
    rtl:0,
    for i:1 thru no_ do
       rtl:rtl+r_[1,i],
    rtl
    )$
```

```
evaluate(m,x0_):=
  block([i,mtemp],
    mtemp:at(m,makelist(x_[i,1]=x0_[i,1],i,1,ns_))
    )$


/* ****************************************************************/
/* Subroutines to determine linearizing control law               */

full_linear():=
  block(
    print("The system can be made full state linear using the"),
    print("given change of coordinates and the following control law:"),
    control_law(),

    print("")
    )$

io_linear():=
  block(
    print("The system can be made input/output linear using the"),
    print("given change of coordinates and the following control law:"),
    control_law(),

    print("")
    )$

control_law():=
  block(
    u_cl:a_dcinv.(-b_+v_),
    if ni_=1 then
       u_cl:ematrix(1,1,u_cl,1,1),
    print("u=",u_cl),

    print("The region of validity for the proposed controller is "),
    print("defined away from the singularities of the"),
    if ni_>1 then
       (
        print("determinant of the decoupling matrix:"),
```

```
        print("det(A)=",factor(d_))
        )
    else
       (
        print("following expression:"),
        print(factor(d_))
        ),

    print("")
    )$

/* ************************************************************/
/* Subroutine to analyze zero dynamics and stabilize system   */
/* if zero dynamics are stable                                */

check_zerodyn():=
  block([status],
    zdz:zerodynamics(f_,g_,h_,phi),
    print("THE ZERO DYNAMICS: ",'diff(transpose(makelist(
                                concat('z,i),i,rtotal_+1,ns_)),t)=zdz),
    status:analysis(zdz),
    if status=stable then
       print("ARE STABLE.")
    else
       if status=unstable then
          print("ARE UNSTABLE.")
       else
          print("have eigenvalues on the imaginary axis in the first order approxi

    return(status)
    )$


zerodynamics(f_,g_,h_,phi):=
  block([u_zero,i,j,k,zd,zdx,zdz],

    scalarmatrixp:false,
    u_zero:-a_dcinv.b_,
    scalarmatrixp:true,
```

```
    for i:rtotal_+1 thru ns_ do
       (
        k:1,
        for j:1 while x_[j,1]#phi[i,1] do
           k:j+1,
        zd[i-rtotal_,1]:f_[k,1]+row(g_,k).u_zero
        ),
    zdx:genmatrix(zd,ns_-rtotal_,1),

    zdz:lratsubst(solve(makelist(z_[i,1]=phi[i,1],i,1,ns_),
                                makelist(x_[i,1],i,1,ns_))[1],zdx),
    lratsubst(makelist(z_[i,1]=0,i,1,rtotal_),zdz)
    )$

analysis(zdz):=
  block([i,j,temp,foa_zd,rootlist,realparts,maxrept,flag],

    for i:1 thru ns_-rtotal_ do
       for j:1 thru ns_-rtotal_ do
          temp[i,j]:diff(zdz[i,1],z_[j+rtotal_,1]),
    foa_zd:at(genmatrix(temp,ns_-rtotal_,ns_-rtotal_),
                                makelist(z_[i,1]=0,i,rtotal_+1,ns_)),

    rootlist:solve(charpoly(foa_zd,lambda)),
    realparts:makelist(realpart(rhs(rootlist[i])),i,1,length(rootlist)),
    maxrept:apply(max,realparts),
    if maxrept>0 then
       return(unstable)
    else
       if maxrept=0 then
          return(indeterminate)
       else
          return(stable)

    )$


/* ***************************************************************/
/* Subrountine to incorporate stabilizing or tracking feedback    */
```

```
stabilize():=
  block([i,j,wtemp],
    for i:1 thru no_ do
      (
       for j:1 thru r_[1,i] do
          ym[i,j]:0,

       wtemp[i,1]:0
       ),

    w_:genmatrix(wtemp,no_,1),
    print("")
    )$

track():=
  block([i,j,wtemp],
    for i:1 thru no_ do
       (
        ym[i,1]:read("Enter desired time trajectory for output #",i,":"),
        for j:2 thru r_[1,i] do
           ym[i,j]:diff(ym[i,j-1],t),

        wtemp[i,1]:diff(ym[i,r_[1,i]],t)
        ),

     w_:genmatrix(wtemp,no_,1),
     print("")
     )$

poles():=
  block([plist,list1,i,j,c_eqn,c,vtemp],
    for i:1 thru ni_ do
      (
       print("Enter the ",r_[1,i]," desired LHP pole location(s)"),
       if ni_=1 then
          print("for the linearized portion of the system")
       else
          print("for the subsystem associated with input #",i),
```

```
      plist:read("in the form: \"",makelist(concat('p,i),i,1,r_[1,i]),"\;\""),
      if length(plist)<r_[1,i] then
         (
          list1:makelist(plist(length(plist)),i,length(plist)+1,r_[1,i]),
          plist:append(plist,list1)
          ),

      c_eqn:1,
      for j:1 thru r_[1,i] do
         c_eqn:c_eqn*('s-plist[j]),
      c_eqn:ratexpand(c_eqn),

      for j:0 thru r_[1,i]-1 do
         c[i,j]:coeff(c_eqn,s,j)
      ),

    k:0,
    for i:1 thru ni_ do
      (
       vtemp[i,1]:0,
       for j:0 thru r_[1,i]-1 do
          vtemp[i,1]:vtemp[i,1]+c[i,j]*(ym[i,j+1]-phi[j+k+1]),
       vtemp[i,1]:vtemp[i,1][1],
       k:r_[1,i]
       ),

    v_:genmatrix(vtemp,ni_,1),
    v_:v_+w_,

    print("The additional feedback required is given by:"),
    print("v=",v_),
    print("")
    )$

/* ****************************************************************/
/* Convert results to Fortran code for simulation purposes        */

make_fortran():=
  block([xlist,vlist,H_TEMP,V_TEMP,U_TEMP,XDOT_TEMP,ffile],
```

```
ffile:read("Enter a filename for the Fortran code:"),
apply(closefile,[transcript]),
apply(writefile,[ffile]),

print("c     Routine to generate closed loop plant dynamics"),
print("      SUBROUTINE USR01(INFO,X,XDOT,Y)"),
print("c "),
print("c ------------------------------------------------------------"),
print("c |   closed loop plant: ",ns_,"states,",ni_,"input(s)/output(s)   |"),
print("c ------------------------------------------------------------"),
print("      DOUBLE PRECISION X(*),XDOT(*),Y(*)"),
print("      DOUBLE PRECISION h_temp(*,*)"),
print("      DOUBLE PRECISION v(*),v_temp(*,*)"),
print("      DOUBLE PRECISION u(*),u_temp(*,*)"),
print("      DOUBLE PRECISION xdot_temp(*,*)"),
print("      INTEGER          INFO(4)"),
print("      INTEGER          ns_,ni_,no_,i"),
print("      LOGICAL          STATE, OUTPUT"),
print("c     "),
print("      STATE = INFO(3).NE.0"),
print("      OUTPUT = INFO(4).NE.0"),
print("c     "),

kill(x_),
xlist:makelist(concat('x,i)=X(i),i,1,ns_),

print(fortran(n:ns_)),
print(fortran(ni:ni_)),
print(fortran(no:no_)),

print("      if (OUTPUT) then"),

H_TEMP:subst(xlist,h_),
print(fortran(H_TEMP)),
print("      do 10 i=1,no"),
print("      Y(i)=h_temp(i,1)"),
print("10    continue "),

print("      endif"),
```

```
print("      if (STATE) then"),

V_TEMP:subst(xlist,v_),
print(fortran(V_TEMP)),
print("      do 20 i=1,ni"),
print("      v(i)=v_temp(i,1)"),
print("20    continue "),

kill(v_),
vlist:makelist(concat('v,i)=v(i),i,1,ni_),

U_TEMP:subst(xlist,u_cl),
U_TEMP:subst(vlist,U_TEMP),
print(fortran(U_TEMP)),
print("      do 30 i=1,ni"),
print("      u(i)=u_temp(i,1)"),
print("30    continue "),

XDOT_TEMP:f_+g_.u_,

kill(u_),
ulist:makelist(concat('u,i)=u(i),i,1,ni_),

XDOT_TEMP:subst(xlist,XDOT_TEMP),
XDOT_TEMP:subst(ulist,XDOT_TEMP),
print(fortran(XDOT_TEMP)),
print("      do 40 i=1,n"),
print("      XDOT(i)=xdot_temp(i,1)"),
print("40    continue"),

print("      endif"),
print("      RETURN"),
print("      END"),

apply(closefile,[ffile]),
print("")
)$
```

```
/* ***************************************************************/
/* Necessary machinery for linearization calculations            */

jacobian(v):=
  block([zz,i,j],
    for i:1 thru ns_ do
        for j:1 thru ns_ do
            zz[i,j]:diff(v[i,1],x_[j,1]),
    genmatrix(zz,ns_,ns_)
    )$

lderiv(v,h_):=
  block([zz,j],
    for j:1 thru ns_ do
        zz[1,j]:diff(h_,x_[j,1]),
    genmatrix(zz,1,ns_).v
    )$

klderiv(v,h_,k):=
  block(
    if k=0 then h_ else lderiv(v,klderiv(v,h_,k-1))
    )$
```

# 5 Summary

We have documented a software package that carries out a complete control design procedure based on well-known linearization theory for a certain subclass of nonlinear systems. Programming the design algorithms using a symbolic manipulator (MACSYMA) proved to be straight forward. This package can be a useful tool to assist the design engineer in creating such control systems.

The current version of this package is programmed to run on a version of MACSYMA (known as "vaxima") that is apparently peculiar to UC-Berkeley and is not completely compatible with the commercial version of MACSYMA. Not much effort has been put into internal simplification and "pretty" formatting. The user may need to examine variable values independently particularly for MIMO systems where expressions become unruly. This may require a limited amount of MACSYMA knowledge. Also, for some MIMO systems where solving the inverse of the coordinates transformation is necessary to generate the zero-dynamics, the equations may prove too complicated for the internal functions of MACSYMA to handle. At the time of this writing, the fortran code output of this package has not been tested to see if it is compatible with such simulation software as MATRIXx.

One aspect of the linearization theory that has not been included is solving for the output (assuming one exists) that will produce a system without zero-dynamics after linearization. This was mainly a philosophical decision based on the assumption that the outputs are usually already prescribed for the engineer and he must design his control system accordingly.

## References


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

[2] 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.

[3] 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.

[4] A. De Luca and G. Ulivi. The design of linearizing outputs for induction motors. In *Nonlinear Control Systems Design, Preprints of the IFAC Symposium, Capri, Italy*, June 1989.

[5] Symbolics, Inc. *VAX UNIX MACSYMA Reference Manual*, 11th edition, November 1985.