AP_LIN: A TOOL BOX FOR APPROXIMATE LINEARIZATION OF NONLINEAR SYSTEMS

by

Raja R. Kadiyala

Memorandum No. UCB/ERL M92/22

1 March 1992

AP_LIN: A TOOL BOX FOR APPROXIMATE LINEARIZATION OF NONLINEAR SYSTEMS

by

Raja R. Kadiyala

Memorandum No. UCB/ERL M92/22

1 March 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

AP_LIN: A Tool Box for Approximate Linearization of Nonlinear Systems *

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

March 1, 1992

Abstract

A toolbox for nonlinear control system design is presented. This package contains modules to approximate systems to polynomials systems of arbitrary order and then render them input-output linear or input-state linear with arbitrary order error terms. We also discuss possibilities for real-time control.

*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

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. 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) tool which handles feedback linearization of nonlinear systems. In turn, the nonlinear CAD tool development has been hampered in the past since the calculations necessary to formulate the feedback law are symbolic in nature.

Restricting 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 $x \in \mathbf{R}^n, u \in \mathbf{R}^m, y \in \mathbf{R}^l$, allows us 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.

The idea of using an approximate system 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. Some of the features of the *AP_ LIN* were taken from cues set by this flight control package while the author was at the NASA Ames Research Center.

We are currently in the midst of constructing the *AP_ LIN* nonlinear control CAD tool package which will implement the necessary operations for feedback linearization on polynomials systems. The present status of 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 [Krener *et al.*, 1987]).

This paper starts with a brief review of exact linearization and higher order linear approximations which are the two control schemes currently implemented in the *AP_ LIN* toolbox. The toolbox itself is discussed in section 3. Finally to show some of the features of the package we give a few examples in section 5 and a possible scenario for realtime control is given in section 6.

2 Review of Nonlinear Techniques

In this section we review two popular nonlinear techniques for controlling systems with nonlinear dynamics, namely exact input-output linearization and the approximate reduction of nonlinear systems to linear systems.

2.1 Exact Linearization

We review, following the notation set in [Isidori, 1989], the basic linearizing theory for the single-input single-output case. The multiple-input multiple-output theory is similar, but more involved. Consider the SISO system

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{1}$$

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

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

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

$$u = \frac{1}{L_g h}(-L_f h + v) \tag{3}$$

yields the linear system

$$\dot{y} = v. \tag{4}$$

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

$$y^{(i)} = L_f^i h(x) + L_g L_f^{i-1} h(x) u \quad i = 1, 2, \ldots \tag{5}$$

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

$$u = \frac{1}{L_g L_f^{\gamma-1} h(x)} (-L_f^\gamma h(x) + v) \tag{6}$$

yields

$$y^{(\gamma)} = v. \tag{7}$$

The integer γ is called the *strong relative degree* of system (1).

For a system with a strong 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

$$\begin{array}{rcl} \Phi_1(x) = \xi_1 = h(x) \\ \Phi_2(x) = \xi_2 = L_f h(x) \\ \vdots \\ \Phi_\gamma(x) = \xi_\gamma = L_f^{\gamma-1} h(x) \end{array} \tag{8}$$

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

$$\begin{array}{rcl} \dot{\xi}_1 = \xi_2 \\ \vdots \\ \dot{\xi}_{\gamma-1} = \xi_\gamma \\ \dot{\xi}_\gamma = b(\xi, \eta) + a(\xi, \eta) u \\ \dot{\eta} = q(\xi, \eta) \\ y = \xi_1. \end{array} \tag{9}$$

In equation (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

$$\dot{\eta} = q(0, \eta) \tag{10}$$

are referred to as the *zero-dynamics* (see [Isidori, 1989] section 4.3 for details). The nonlinear system (1) is said to be minimum phase if the zero-dynamics are asymptotically stable.

2.1.1 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 (6), we choose

$$v = y_M^{(\gamma)} + \alpha_1(y_M^{(\gamma-1)} - y^{(\gamma-1)}) + \ldots + \alpha_\gamma(y_M - y) \tag{11}$$

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

$$s^\gamma + \alpha_1 s^{\gamma-1} + \ldots + \alpha_\gamma \tag{12}$$

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

$$\begin{aligned} \dot{e} &= Ae \\ \dot{\eta} &= q(\xi, \eta) \\ \xi_i &= e_i + y_M^{(i-1)} \end{aligned} \tag{13}$$

where A is the companion matrix associated with (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, (6) and (11) yield bounded tracking. (see [Sastry and Isidori, 1987]).

2.2 Approximate Reduction of Nonlinear Systems to Linear Systems

Consider the MIMO system defined below

$$\begin{aligned} \dot{x} &= f(x) + G(x)u \\ y &= h(x) \end{aligned} \tag{14}$$

with $x \in \mathbf{R}^n, u \in \mathbf{R}^m, y \in \mathbf{R}^l$. Furthermore $f(x)$ and the columns and rows of $G(x)$, $h(x)$ are smooth.

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 (14) 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 :

When can we transform

$$\dot{x} = Ax + f^{[2]}(x) + f^{[3]}(x) + \cdots \tag{15}$$

to the linear equation

$$\dot{s} = As \tag{16}$$

through a change of coordinates of the form

$$x = s + \phi^{[2]}(s) + \phi^{[3]}(s) + \cdots \tag{17}$$

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)?

Poincaré gave necessary conditions on the eigenvalues of A for such a transformation to result in a linear equation (see [Arnold, 1983] chapter 5). From a control standpoint this problem is not very interesting since we have the extra freedom to choose u.

So let us consider the MIMO control system defined in (14). Suppose that this system can be expanded as

$$\begin{aligned} \dot{x} &= Ax + Bu + f^{[2]}(x) + G^{[1]}(x)u + O(x^3, x^2u) \\ y &= Cx + h^{[2]}(x) + O(x^3) \end{aligned} \tag{18}$$

Apply quadratic changes in the state space and output space

$$z = x - \phi^{[2]}(x) \tag{19}$$

$$w = y - \gamma^{[2]}(y) \tag{20}$$

to obtain

$$\begin{aligned} \dot{z} &= Az + Bu + f^{[2]}(z) + G^{[1]}(z)u \\ &\quad - \left[Az + Bu, \phi^{[2]}(z)\right] + O(x^3, x^2u) \\ w &= Cz + h^{[2]}(z) + C\phi^{[2]}(z) \\ &\quad - \gamma^{[2]}(w) + O(x^3) \end{aligned} \tag{21}$$

where $\left[Az + Bu, \phi^{[2]}(z)\right]$ is the Lie bracket of $Az + Bu$ and $\phi^{[2]}(z)$ and $\left([f(z), g(z)] = \frac{\partial g}{\partial z} f(z) - \frac{\partial f}{\partial z} g(z)\right)$. 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

$$u = \alpha^{[2]}(x) + (I + \beta^{[1]}(x))v \tag{22}$$

to obtain

$$\begin{aligned} \dot{z} &= Az + Bv + R_1{}^{[2]}(z, u) + O(x^3, x^2u) \\ w &= Cz + R_2{}^{[2]}(z, w) + O(x^3) \end{aligned} \tag{23}$$

where

$$\begin{aligned} R_1{}^{[2]}(z, u) &= f^{[2]}(z) + G^{[1]}(z)u \\ &\quad - \left[Az + Bu, \phi^{[2]}(z)\right] \\ &\quad + B\left(\alpha^{[2]}(z) + \beta^{[1]}(z)u\right) \\ R_2{}^{[2]}(z, w) &= h^{[2]}(z) + C\phi^{[2]}(z) - \gamma^{[2]}(w) \end{aligned} \tag{24}$$

The equation

$$R_1{}^{[2]}(z, u) = 0 \tag{25}$$

is referred to as the *controller homological equation of degree 2.* If equation (25) can be solved then we get

$$\dot{z} = Az + Bv + O(z^3, z^2u) \tag{26}$$

Further, suppose this system can be expanded as

$$\begin{aligned} \dot{z} &= Az + Bv + f^{[3]}(z) \\ &\quad + G^{[2]}(z)v + O(z^4, z^3u) \end{aligned} \tag{27}$$

We could repeat the above computations with

$$z = x - \phi^{[3]}(x) \tag{28}$$

$$v = \alpha^{[3]}(x) + (I + \beta^{[2]}(x))r \tag{29}$$

to get a system which is linear up to $O(z^4, z^3u)$.

In general we have

$$\begin{aligned} \dot{x} &= Ax + Bu + f^{[p]}(x) \\ &+ G^{[p-1]}(x)u + O(x^{p+1}, x^p u) \end{aligned} \tag{30}$$

with change of coordinates

$$z = x - \phi^{[p]}(x) \tag{31}$$

$$u = \alpha^{[p]}(x) + (I + \beta^{[p-1]}(x))v \tag{32}$$

which yields

$$\dot{z} = Az + Bv + R^{[p]}(z, u) + O(x^{p+1}, x^p u) \tag{33}$$

where

$$\begin{aligned} R^{[p]}(z,u) &= f^{[p]}(z) + G^{[p-1]}(z)u \\ &- \left[Az + Bu, \phi^{[p]}(z)\right] \\ &+ B\left(\alpha^{[p]}(z) + \beta^{[p-1]}(z)u\right) \end{aligned} \tag{34}$$

The equation

$$R^{[p]}(z,u) = 0 \tag{35}$$

is the *controller homological equation of degree p.*

In contrast to section 2.1, this method attempts to linearize the state space equations rather than the input-output map.

2.2.1 Solution of Controller Homological Equation

Equation (35) is a system of

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

linear equations in

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

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 1: Problem size for various systems

unknowns, 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 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 coefficients of the remainder term $R^{[p]}$, i.e.

- Underdetermined

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

- Overdetermined

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

2.2.2 Controller Implementation

In addition to linearization, one must also stabilize the plant. A subtle point that needs to be addressed is: at what point in the design should we place the eigenvalues of A with the standard state feedback control law

$$Fx + v$$

(where v is the redefined input)? There are two possibilities:

1. on the original system

2. on 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 can 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 most $O(x^p)$ terms and we have no way to remove this new term. So, 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:

$$u = Fx + v \tag{36}$$

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

$$\begin{aligned} \dot{x} &= Ax + Bv + \hat{f}^{[p]}(x) \\ &+ G^{[p-1]}(x)v + O(x^{p+1}, x^p u) \end{aligned} \tag{37}$$

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

3. Define the coordinate change and feedback

$$\begin{aligned} z &= x - \phi^{[p]}(x) \\ v &= \alpha^{[p]}(z) + (I + \beta^{[p-1]}(z))r \end{aligned} \tag{38}$$

and set

$$w = y - \gamma^{[p]}(y) \tag{39}$$

We can estimate z by

$$\dot{\hat{z}} = A\hat{z} + Bu + L(w - C\hat{z}) \tag{40}$$

3 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

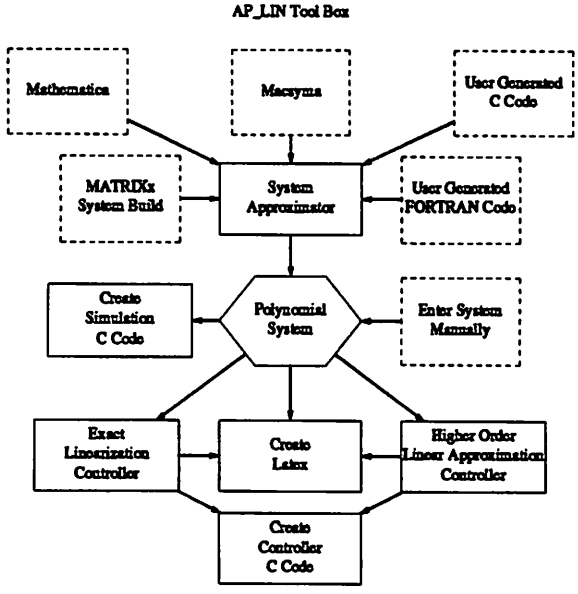

Figure 1: The *AP_ LIN* Toolbox Flow

[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 3.1. As seen in figure 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 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.

3.1 Polynomial System Approximator

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

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

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.

3.1.1 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 can easily see from table 2 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.

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

Table 2: Number of parameters to be identified for various systems

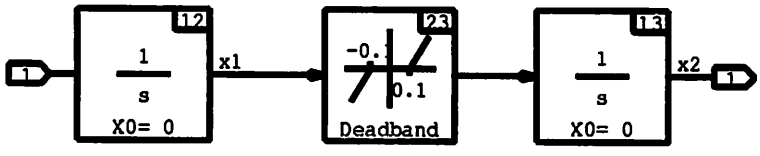

Figure 2: Backlash system from *SystemBuild*

4 Computation of Exact Linearization Control Law

The computation of the exact linearization control law is straight forward once the system has been approximated by a polynomial system. *AP_ LIN* basically caries through the calculations in section 2.1. *AP_ LIN* also does carry through the calculations for mimo systems.

In doing these computations several core routines that handle 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 Examples

In this section we give a couple of examples to show some the capabilities of the *AP_ LIN* toolbox. Since the toolbox is still in its infancy, we have not had a chance to work through an extensive number of systems. A more thorough undertaking on more realistic examples will be forthcoming. All system and controller equations we generated by *AP_ LIN* in LaTeXform.

5.1 Backlash Example

In figure 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

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

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

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

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 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

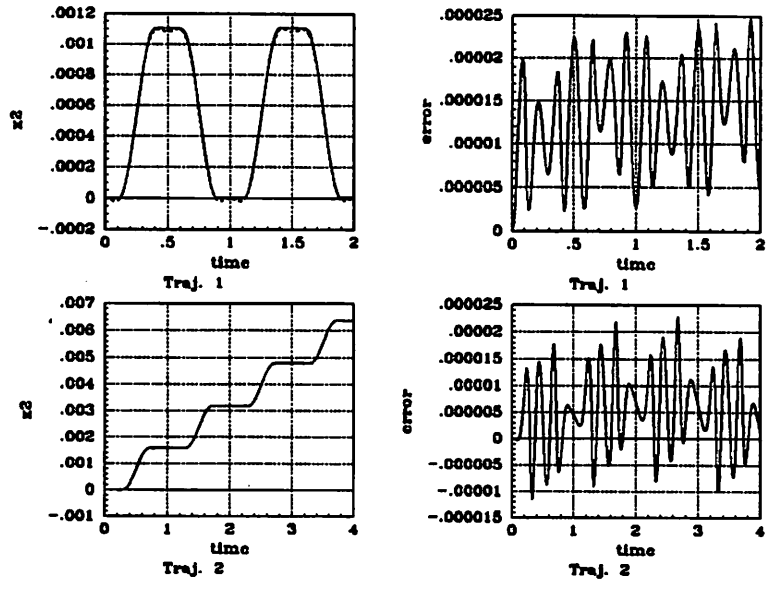

Figure 3: Third order approximation to the backlash system

it to continue through the example. Results of a simulation to track an offset sinusoid through the dead-zone region is given in figure 4. The actual position (the dashed line) comes reasonably close, less than the dead-zone region, to tracking the desired position (the solid line).

5.2 Simple Two Dimensional System

Consider the system

$$\begin{aligned} \dot{x_1} &= \sin(x_2) \\ \dot{x_2} &= u \\ y &= x_1 \end{aligned} \tag{41}$$

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

$$\begin{aligned} \dot{x_1} &= x_2 - \frac{{x_2}^3}{6} \\ \dot{x_2} &= u \\ y &= x_1 \end{aligned} \tag{42}$$

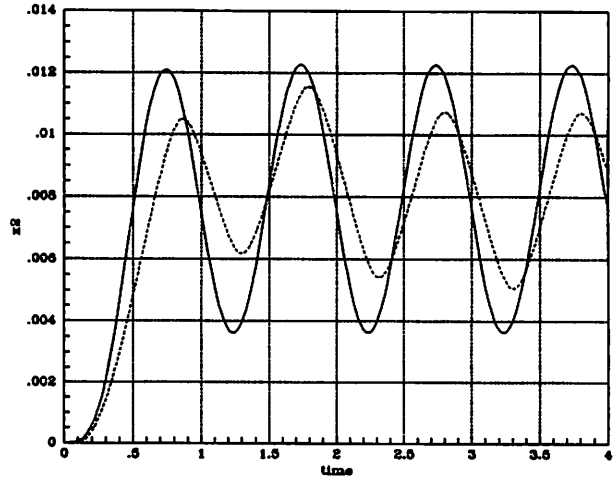

Figure 4: Modified exact linearization controller

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

$$\begin{aligned} \dot{y} &= \sin(x_2) \\ \ddot{y} &= \cos(x_2)u \\ u &= \frac{v}{\cos(x_2)} \end{aligned} \tag{43}$$

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

$$\begin{aligned} \dot{y} &= x_2 - \frac{{x_2}^3}{6} \\ \ddot{y} &= u\left(1 - \frac{{x_2}^2}{2}\right) \\ u &= \frac{1}{\left(1 - \frac{{x_2}^2}{2}\right)} \end{aligned} \tag{44}$$

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

$$\begin{aligned} \phi^{[3]}(z) &= \begin{bmatrix} 0 \\ \frac{z_2{}^3}{6} \end{bmatrix} \\ \alpha^{[3]}(z) &= 0 \\ \beta^{[2]}(z) &= 0 \end{aligned} \tag{45}$$

where

$$x = z + \phi^{[3]}(z) \tag{46}$$

$$u = \alpha^{[3]}(x) + \left(I + \beta^{[2]}(x)\right) v \tag{47}$$

This coordinate change yields

$$\dot{z} = \begin{bmatrix} z_2 + \frac{z_2{}^5}{12} - \frac{z_2{}^7}{72} + \frac{z_2{}^9}{1296} \\ \\ \left(1 + \frac{z_2{}^3}{2} - \frac{z_2{}^5}{4} + \frac{z_2{}^7}{24} - \frac{z_2{}^9}{512}\right) v \end{bmatrix} \tag{48}$$

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.

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 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.

5.3 Ball and Beam Example

Figure 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:

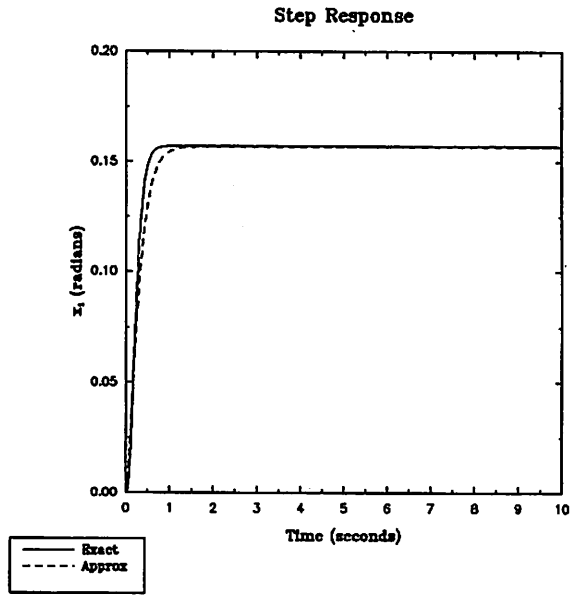

Figure 5: Step response of both nonlinear controller

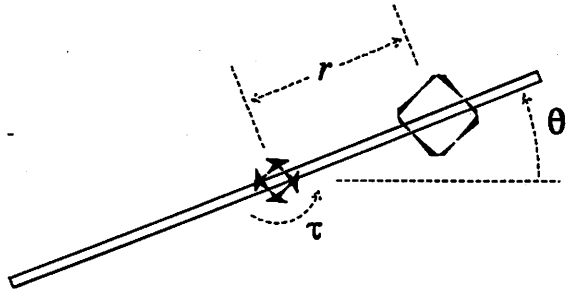

Figure 6: Ball and Beam

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_1 x_4^2 - g \sin x_3 \\ x_4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u \qquad (49)$$

$$y = x_1$$

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 Sparc-Station 1. The control law as created by *AP_ LIN* is given below.

Input-Ouput 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 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 ([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.

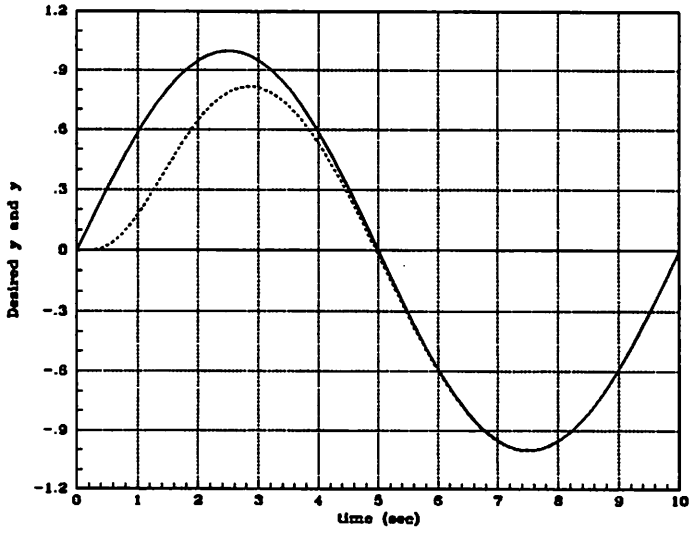

Figure 7: Tracking Results for the Ball and Beam

6 Real-time Control

In table 3 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 exact linearization controller on all platforms took 10 *msec*. 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.

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 3: Computation time for various processors

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 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.

7 Conclusion

In this paper, 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

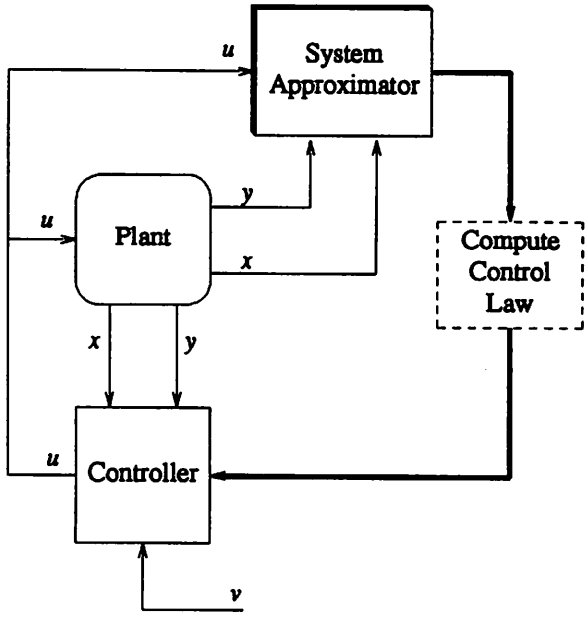

Figure 8: Real-time control diagram

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.

8 Acknowledgements

The author would like to thank Sunil Shah of Integrated Systems Inc. for his input, support, and encouragement to develop this package, Art Krener at UC Davis for his guidance and motivation, and George Meyer at the NASA Ames Research Center for allowing the author to spend one summer at Ames to work on a similar package for flight control.

References

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

[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.

[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.

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

[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.

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

[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.

[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.

[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.

A Manual Pages for the *AP_LIN* Package

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 **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, Dept. of EECS 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 **create_model(1), linearize(1), poincare(1), spline_hyper(1), spline_usr(1)** and 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, Dept. of EECS 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

$$x' = f(x) + G(x)u$$
$$y = h(x)$$

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

$$y = h(u)$$

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, Dept. of EECS 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, Dept. of EECS 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, Dept. of EECS 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* 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, Dept. of EECS 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

$$x' = f(x) + G(x)u$$
$$y = h(x)$$

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

$$y = h(u)$$

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, Dept. of EECS 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*] [-v] [-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

$$x' = f(x) + G(x)u$$
$$y = h(x)$$

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

$$y = h(u)$$

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.

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

-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, Dept. of EECS 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 ...)