Discrete Function Approximation: Numerical Tools for Nonlinear Control

Muruhan Rathinam * and Richard M. Murray
Division of Engineering and Applied Science
California Institute of Technology
Pasadena, CA 91125
muruhan@cds.caltech.edu, murray@indra.caltech.edu
Fax: 626-796-8914, Ph: 626-395-4882

Technical Memorandum : CDS 98-009
31 August 1998

Abstract

We describe a method for discrete representation of continuous functions and show how this may be used for typical computations in nonlinear control design. The method involves representing functions by their values and finitely many derivatives at discrete set of points on the domain. We propose a grid structure based on a hierarchy of rectangular boxes that provides flexibility in placing grid points densely in some regions and sparsely in the other. The grids possess enough structure to facilitate easy interpolation schemes based on piecewise polynomials. We illustrate the method using a simple example where we compute the feedback linearizing output of a system.

1 Introduction

Much of the theory developed for nonlinear control has traditionally been applied to systems that have a symbolic description and computations are carried out analytically. For example, checking whether a given control affine system is static feedback linearizable involves computing certain distributions obtained by repeated Lie bracket calculations of the control vector fields with the drift vector field, and checking involutivity of these distributions. These computations involve differentiation and linear algebra, and can be carried out analytically.

Unfortunately, many results in nonlinear control cannot be applied to applications because the operations required to compute nonlinear control laws cannot be carried out in closed form. For example, the task of finding the

*Research supported in part by AFOSR Grant F49620-95-1-0409.

linearizing outputs for a feedback linearizable systems involves solving a set of partial differential equations, which can rarely be done in closed form. In principle, one can resort to discretization and numerical methods, but this is intractable for systems with moderate state dimension and the Lie bracket calculations used to obtain the PDEs can lead to growth of expressions which may not be in an optimal form for discretization.

A second problem with applying many nonlinear tools is that symbolic descriptions for the dynamics may not be available in the first place. In many situations the analytical model of a system may need to be combined with experimental data obtained at certain points in the state space. Lift, drag, and moment coefficients in flight dynamics are one example. Hence it may be more efficient to start with a discrete representation of the system and carry out all the computations in the discrete domain.

In this paper we describe a general methodology for discrete computations in nonlinear control as well as a preliminary software implementation. We refer to both the software as well as the general methodology as *Discrete Function Approximation* (DFA). DFA provides a set of numerical tools which enables one to carry out the the kind of computations typically encountered in nonlinear control. DFA is implemented in MATLAB 5.0, taking advantage of the object oriented features and the multidimensional arrays.

There have been several efforts in generating nonlinear control toolboxes that do not rely on symbolic calculations. Krener and co-workers have written the Nonlinear System Toolbox (NST). NST uses local Taylor series expansions to represent vector fields, resulting in a technique which is potentially restricted to a smaller domain. A similar approach was studied by Kadiyala [3]. Bortoff has developed a spline toolbox for nonlinear control and applied it to the approximate feedback linearization [1]. The DFA technique proposed here is intended to be more global and more tied to the types of data that one obtains in experiments (e.g., stability derivatives and control derivatives at different points in the operating envelope of a flight vehicle).

2 Discrete Representation

In nonlinear control one typically starts with a set of nonlinear objects such as functions, vector fields, differential forms etc. on a manifold, which model a control system. A DFA user usually proceeds by discretizing above continuous objects to obtain discrete objects on which further computations may be carried out. DFA uses a discrete representation of functions, vector fields etc., that consists of a discrete set of points of the domain and the value and some finitely many derivatives of the function (or vector field etc.) at these points. The total number of derivatives to be represented is left to the user. This representation may be considered as a generalization of finite differencing where one uses a uniform grid of points at which a functions values are stored. See Figure 1.

In control problems the dimensionality of the domain (which is usually the dimensionality of the state space) can be in the order of tens. Assuming we need

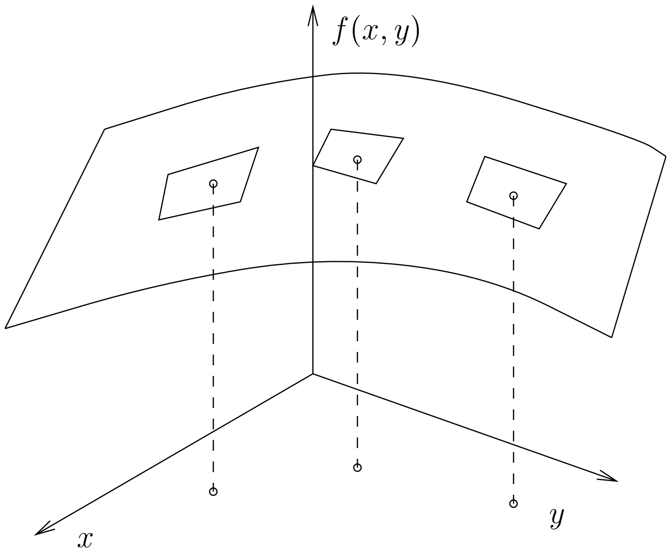

Figure 1: Discrete Representation.

at least two points per states, committing to a uniform grid requires at least 2^d number of points where d is the domain dimension. Even with single precision the storage requirements exceed 1GB for a scalar function when $d > 16$. The grid structure (we shall refer to domain points at which data are given as grid points even though they may not necessarily form a lattice) proposed in DFA is more general than the uniform grid and allows one to choose points closely in regions of state space where changes are rapid and sparsely in regions where changes occur slowly. See Section 4 for more details.

Storing exact derivatives at grid points minimizes errors in computing derivatives. Especially the repeated Lie bracket calculations involve taking several derivatives. If the derivatives of the vector fields are not known at the grid points then one needs to numerically differentiate them. Since numerical differentiation is highly sensitive to error propagation one may avoid this by storing sufficiently many derivatives of the vector fields at the grid points which have been obtained from analytical formulae for the derivatives. Since the Lie bracket of two vector fields at a given point only depends on their values and derivatives at that point, if the original derivatives are accurate then the Lie bracket computations will be accurate as well.

In order to represent an intrinsic geometric object such as a map Φ from one manifold to another (this includes functions and vector fields on manifolds) by a set of numbers one needs to choose coordinate systems on the domain and range of the map. Once this is done we have a map $\phi : \mathbb{R}^n \to \mathbb{R}^m$ which is the coordinate representation of Φ. But often the range manifold may have a bundle structure and in that event we might prefer to use a coordinate representation of Φ given by a map $\phi : \mathbb{R}^n \to \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \cdots \times \mathbb{R}^{m_k}$. For instance Φ may be tensor of rank k on the domain manifold. In that case we have $m_i = n$ for $i = 1, \ldots, k$. Thus the most general continuous object to be discretized by DFA is a map $\phi : \mathbb{R}^n \to \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \cdots \times \mathbb{R}^{m_k}$. Representing the value of such a map requires a multidimensional array of dimensions $m_1 \times m_2 \cdots \times m_k$.

Remark 1 It is important to note that a given map $\phi : \mathbb{R}^n \to \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \cdots \times \mathbb{R}^{m_k}$ in coordinates may correspond to different kinds of objects. For instance when $n = 2$ and $k = 2, m_1 = 2$ and $m_2 = 2$, ϕ might represent a tensor of contravariant rank 2, a tensor of covariant rank 2 or a tensor of covariant rank 1 and contravariant rank 1.

3 Numerical Issues and Basic Operations

Two important desirable properties of any such general purpose software are good numerical accuracy and adaptability of software for a reasonably wide class of computations. In general it is hard to satisfy both criteria and one needs to find a compromise. Typically for best numerical accuracy one may have to exploit special knowledge of the system in writing the integration routines or interpolation schemes etc. Such a code may not be useful for another application. On the other hand a general purpose software such as DFA should be made up of some basic functions which could be put together by the user in different ways depending on the application. While this provides flexibility, the chances of propagation of round off errors is higher.

As in any numerical scheme there are typically two sources of error. One is the round off error due to finite precision arithmetic. The other is the error due to approximation of functions by certain simple classes of functions. When one computes the Lie bracket of two vector fields whose values and first derivatives are given at certain grid points, the only error in the computed value of the result at these grid points is due to finite precision arithmetic. But if one needs the Lie bracket at some point in between the given grid points or if one didn't have derivative information for the original vector fields then some kind of approximation needs to be made. This would typically involve interpolation and the error in the computation includes both types mentioned above.

The above example high lights two important problems: one of interpolation and the other of numerical differentiation. The latter may be thought of as an interpolation problem too. Interpolation of functions with derivative information on arbitrary multidimensional grids is still a subject of research. The DFA approach is to use grids that are practically reasonable and hence more general than uniform grids but yet possess sufficient structure to make the interpolation problem tractable. This is explained in Section 4.

DFA consists of a set of elementary operations which may be put together by the user to carry out frequently used computations involved in nonlinear control. Some typical operations include addition, subtraction, multiplication (of matrix valued functions), taking partial derivatives of a function, given a matrix valued function finding the inverse matrix valued function, composition of functions, and integration of vector fields. All operations mentioned except the last two only involve round off error. However in partial differentiation the number of derivatives of the result is one fewer than the original function. In composition of functions as well as integration of vector fields interpolation is necessary. Hence errors involve both round off as well as error in interpolation.

To illustrate how some of these computations may be done, consider functions f and g on a one dimensional domain. When discretized, they will be represented by values f_i^0 and g_i^0 (where i is the index of a grid point) and derivatives f_i^j and g_i^j where $j = 1, \ldots, k$ with k being the total number of derivatives. Then addition $h = f + g$ is given by,

$$h_i^j = f_i^j + g_i^j, \quad j = 0, \ldots, k,$$

multiplication $h = fg$ is given by the recursion,

$$h_i^0 = f_i^0 g_i^0 \tag{1}$$

$$h_i^{j+1} = f_i^j g_i^{j+1} + f_i^{j+1} g_i^j, \quad j = 0, \ldots, k-1, \tag{2}$$

and derivative h of f is given by,

$$h_i^j = f_i^{j+1}, \quad j = 0, \ldots, k-1.$$

4 Interpolation and Grid Structure

Computation of values and derivatives of a function at points other than the grid points is needed in certain calculations. Typically this happens when the flow of a vector field represented in DFA needs to be computed. Flow computations are needed in feedback linearization and differential flatness for instance, see [5] and [4]. The step sizes taken to integrate the vector field are typically quite small compared to the spacing between the grid points on which the vector field has been sampled and also the integral curves typically will not be aligned with the grid. Hence interpolation is necessary. Also since the performance of integrators depends on the smoothness of the vector field, the interpolant should be as smooth as possible. It is desirable to have an interpolation scheme that only depends on local information. This reduces computational effort and also avoids some unpleasant artifacts such as ringing effects seen when global polynomial fits are used.

Interpolation of data on an arbitrary set of grid points in arbitrary dimensional spaces is very much a research problem. In 2 or 3 dimensions one may do triangulation or simplicial tesselation and use some finite element functions to do the interpolation. However, in higher dimensions simplicial tesselation is nontrivial. Hence we shall consider rectangular box type tesselation of the space and piecewise polynomial interpolation.

Piecewise polynomial interpolation of a function of one dimensional domain is trivial and the interpolant is C^k-continuous where k is the number of derivatives given at each grid point. In each interval bounded by two adjacent grid

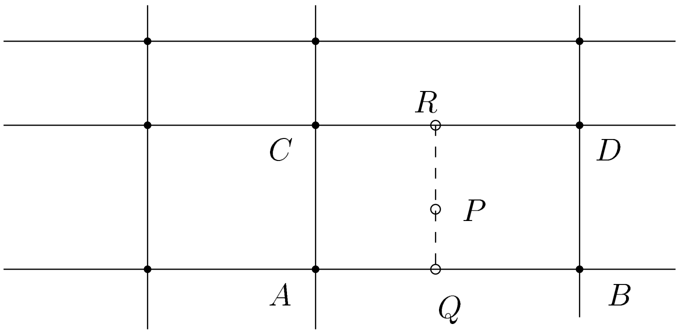

Figure 2: Rectangular Grid

points one interpolates using a polynomial of degree $2k+1$ to uniquely fit the set of $2(k+1)$ independent numbers for the values and derivatives at the grid points. By construction the interpolant will be C^k. However, extending this idea to higher dimensions, even in the case of rectangular grids is not so trivial. In higher dimensions C^k-continuity is not guaranteed, unless we choose the method wisely.

To illustrate the difficulties let us consider a set of rectangular grid points in the plane. The x and y spacing between adjacent points need not be constant. See Figure 2 where grid points are shown by small filled circles. Suppose the values and first order derivatives of a function (f, f_x and f_y) are given at the grid points. One may consider doing a polynomial interpolation inside each rectangle to fit the corner data. Since there are 12 independent numbers we need a 12 dimensional subspace of polynomials in two variables to find a unique solution. Since we are interested in arbitrary dimensional domains it is also desirable to think of an approach that recurses on the dimension of the domain. Hence in order to interpolate to the point P inside rectangle $ABDC$ we may first want to interpolate on the edges AB and CD to the points Q and R respectively, where Q and R are the projections of P onto AB and CD. Then we could interpolate on the line QR to P. It is easy to see such a method could be recursively implemented for interpolation inside multidimensional rectangular boxes.

Since we have f and f_x at both A and B, we need to fit a cubic (in x) along AB to find the value at Q. A similar procedure applies to edge CD and the point R. But we also need to interpolate f_y along the edges AB and CD. Since we don't have information on f_{yx} we have to do a linear interpolation. Then knowing f and f_y at Q and R we may do a cubic (in y) interpolation along QR to obtain the values at P. This gives a well defined interpolating polynomial inside the rectangle and repeating this for all rectangles we get a well defined piecewise polynomial interpolant.

It is easy to see that the interpolant in the above scheme is C^0-continuous across the edges and C^1-continuous across the horizontal edges when traversed in the vertical direction. But the interpolant is in general not C^1-continuous across the vertical edges. This is because we had to use linear interpolation for f_y along AB and CD since we did not know f_{xy}. In addition this method lacks another desirable property. The result depends on the order in which interpolation is done along different dimensions. We first interpolated along the x direction and then along the y direction. If we switched the order we get a different method which in general fails to be C^1 across the horizontal edges.

We get better results if we use the same idea of interpolation but use different choice of derivatives. In the above example if we use f, f_x, f_y and f_{xy} values then we will be using cubics for interpolation of both f and f_y along the horizontal edges and hence obtain C^1-continuity across all edges. In addition the method will give the same result if we swap the order of interpolation to y direction first and then the x. This is seen from the fact that the interpolating polynomial lies in the space of bicubic polynomials (i.e. polynomials spanned by the monomials $\{1, x, x^2, x^3, y, xy, x^2y, x^3y, y^2, \ldots, x^3y^2, y^3, \ldots, x^3y^3\}$) which is 16 dimensional. But the corner data also consists of 16 independent scalars (f, f_x, f_y and f_{xy} values). Hence the correspondence must be one to one and onto implying that it does not depend on the order in which different dimensions are interpolated. The earlier method resulted in choosing a 12 dimensional subspace of the above 16 dimensional bicubic space and this subspace depended on the order in which different dimensions were interpolated.

This method can be generalized to multidimensions and arbitrary number of derivatives. In d dimensions one must use information on all derivatives of order $(k_1, \ldots, k_d)$ where each k_i ranges from 0 to k. In other words one uses upto kth order derivatives in each coordinate direction independently. Hence at each grid point one has $(k+1)^d$ independent scalars (for a scalar valued function). Since a rectangular box in d dimensions has 2^d corners this amounts to $(2(k+1))^d$ independent scalars. The interpolating polynomial subspace will be spanned by the monomials ${x_1}^{l_1} \ldots {x_d}^{l_d}$ where each l_i ranges from 0 to $2k+1$. This space has dimension $(2(k+1))^d$ and hence we have all the desired properties as in the bicubic case mentioned above. Since in DFA the derivatives are only stored upto a total order k, in other words $k_1 + \cdots + k_d \leq k$, in the current implementation we set the higher derivatives for which $k_1 + \cdots + k_d > k$ to 0.

So far we have only considered rectangular grids. These suffer from the same drawbacks as uniform grids as mentioned in Section 2. Hence we need grid structures that allow the freedom to place grid points densely in some regions without having an exponential growth in the overall number of grid points. But yet we want to preserve the rectangular box type structure so that we can still apply the above interpolation scheme. This may be done by considering a hierarchy of rectangular boxes.

Basically one starts with a (rectangular) box (aligned with coordinate axes) representing a region of interest with grid points at the corners. Then the box may be partitioned in a regular manner into smaller boxes. For instance along coordinate direction x_i the box may be divided into n_i parts (the spacing need not be equal). This leads to $n = n_1 \times n_2 \cdots \times n_d$ sub boxes. These sub boxes may each in turn be divided into boxes in different ways. Some boxes may not be subdivided at all. The grid points are the corners of all these boxes.

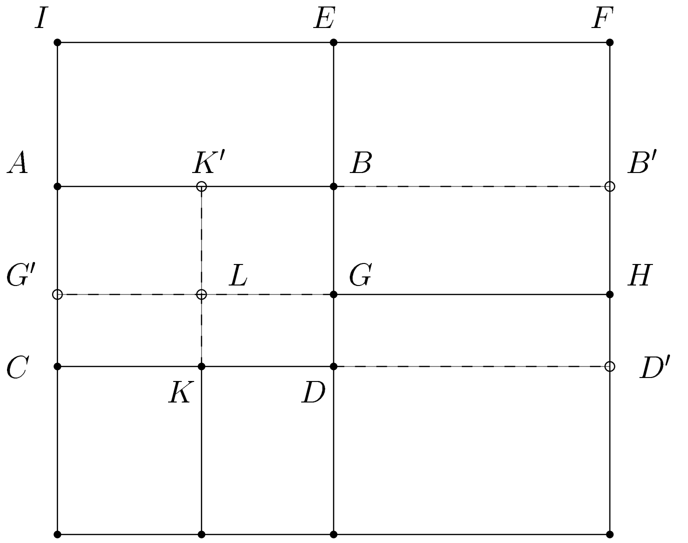

Figure 3: Rectangular Hierarchical Grid

This structure may be represented by a tree. See Figure 3 for an example, where the grid points are shown by filled small circles and box boundaries by solid lines. The interpolation scheme described for the regular rectangular grid may be applied to this hierarchical rectangular grid as well, but some extra considerations are necessary. It may be observed that some boxes have grid points not only at their corners but also on their edges. In higher dimensions these extra points can be on any of the lower dimensional boundaries. We illustrate using the example in Figure 3 how to deal with the boxes that have extra grid points on them.

In Figure 3 if we consider the box $GHFE$, it has an extra grid point B on one of its edges. Hence we need to split the box into two parts by dividing along line BB'. Since B' is not a grid point we need to assign value and derivatives to it to carry out the interpolation. B' may be regarded as a 'phantom grid point,' which is assigned value and derivatives in order to complete the interpolation process. All phantom grid points are shown by larger unfilled circles.

The interpolant will be C^k regardless of what value we assign to B', but however for the interpolation to be accurate we need a sensible way to do this. In fact the most natural thing to do is to interpolate on the line FH in the usual way using polynomials to obtain the value and derivatives at B'. The situation is more complicated with box $ABDC$. We need to split the box into four parts because we have points K and G on adjacent edges. As in the case of box $GHFE$ we could assign values and derivatives to the points K' and G' by polynomial interpolation on lines AB and AC. But however we also need to assign value and derivatives to the point L as well. There are two obvious ways of doing this: one is to interpolate on the line GG' and the other is to interpolate on the line KK'. Both are valid, but give different answers. We could make a decision based on whether L is closer to K and K' or G and G'. We still have C^k continuity but we no longer have the symmetry property (i.e. the interpolation has a preferred coordinate direction). We could also take the average and this maintains symmetry but increases computation. This idea has not been implemented in DFA yet and is currently being investigated.

Another important aspect to note is that this splitting of boxes does not propagate. Even though we introduced a phantom grid point K' on the edge of box $IEBA$, since the value and derivatives at K' are consistent with the information on the edge AB we do not have to split the box $IEBA$. Hence we do not have exponential growth of data points as in a regular rectangular grid.

5 Example: Feedback Linearization

We shall illustrate by a simple example of a feedback linearizable system how DFA can be used to compute the linearizing output. See [2] for an introduction to feedback linearization. Let us consider the system

$$\dot{y}_1 = y_2 \tag{3}$$
$$\dot{y}_2 = u. \tag{4}$$

Introduce the change of variables

$$y_1 = x_1 + \frac{1}{4}{x_2}^2 \tag{5}$$
$$y_2 = x_2 + \frac{1}{4}{x_1}^2. \tag{6}$$

This change of variables is a local diffeomorphism around the equilibrium point $(0, 0)$. In (x_1, x_2) coordinates the system is nonlinear but affine in control and has the form

$$\dot{x} = f(x) + g(x)u, \tag{7}$$

where

$$f = \frac{{x_1}^2 + 4x_2}{4 - x_1x_2}\frac{\partial}{\partial x_1} + \frac{x_1({x_1}^2 + 4x_2)}{2(x_1x_2 - 4)}\frac{\partial}{\partial x_2},$$

and

$$g = \frac{2x_2}{x_1x_2 - 4}\frac{\partial}{\partial x_1} + \frac{4}{4 - x_1x_2}\frac{\partial}{\partial x_2}.$$

The goal is to compute the linearizing output $y_1 = x_1 + \frac{1}{4}{x_2}^2$ on a set of grid points shown by filled small circles in Figure 4 (the grid includes the origin).

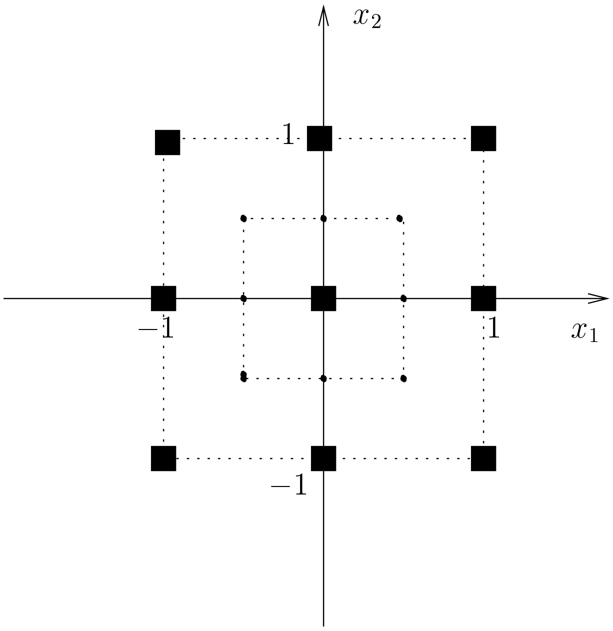

Figure 4: Grid points.

In order to do these we need to sample f and g on a larger domain. The grid consisting of 9 points on which f and g were sampled is shown by filled squares in Figure 4. The computations involve first finding a one-form α that annihilates g. See [5] for instance. Since g and $[f,g]$ span the whole space α could be taken to be the 2nd row of the inverse of the matrix whose columns are f and $[f,g]$. This could be computed using DFA. DFA would automatically compute as many derivatives of α as possible. Also analytically this can be computed to give $\alpha = -dx_1 - \frac{1}{2}x_2 dx_2$. Since the dx_1 component of α never vanishes we normalize α by multiplying by -1 to obtain $\bar{\alpha}$.

The next step is to compute a function $y_1(x_1,x_2)$ so that dy_1 is proportional to $\bar{\alpha}$. There are many such functions y_1 and they differ by a diffeomorphism. Since dx_1 component of α never vanishes, the level sets of y_1 are transversal to the x_1 axis and hence we may define the y_1 value at any point (x_1,x_2) as follows. Follow a curve that starts at (x_1,x_2), ends on the x_1 axis, lies on the same level set of y_1 (i.e. annihilates $\bar{\alpha}$) and has unit speed in the x_2 direction. Take the value at the x_1 axis intersection to be $y_1(x_1,x_2)$. This is computed by defining a parametrized curve $(F(t),z(t))$ that satisfies $(F(0),z(0)) = (x_1,x_2)$, $z(t) = (1-t)x_2$ and $(\frac{d}{dt}F(t), \frac{d}{dt}z(t))$ annihilates $\bar{\alpha}$. Hence we have the following ODE for $F(t)$.

$$\frac{d}{dt}F(t) = \beta(F(t), z(t))\frac{dz(t)}{dt}, \quad F(0) = x_1, \tag{8}$$

where β is defined by $\bar{\alpha} = dx_1 - \beta dx_2$. Then $y_1(x_1, x_2) = F(1)$. The right hand side of the ODE can be computed analytically to be $-\frac{1}{2}(t-1){x_2}^2$ and the ODE can be integrated analytically to obtain $F(1) = x_1 + \frac{1}{4}{x_2}^2$, which is the linearizing output.

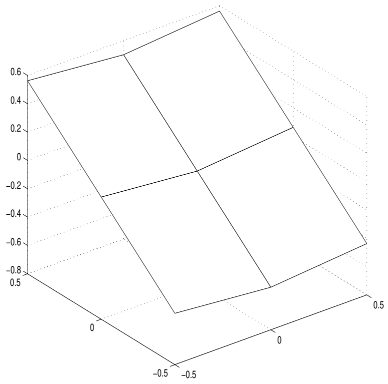

Figure 5: y_1: DFA

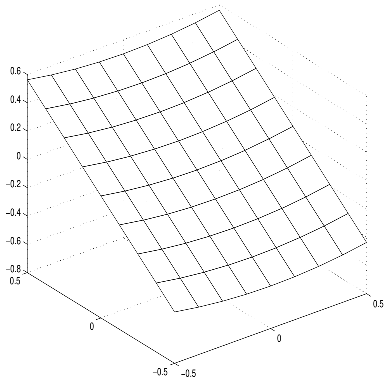

Figure 6: y_1: Analytical

The computations can be carried out in DFA and involve operations such as composition of maps, extraction of components, differentiation and computation of flows of ODEs. Furthermore by integrating the appropriate variations of the ODE we obtain the partial derivatives of y_1 upto second order. The flow computation in DFA computes the flow of a vector field and returns the values and

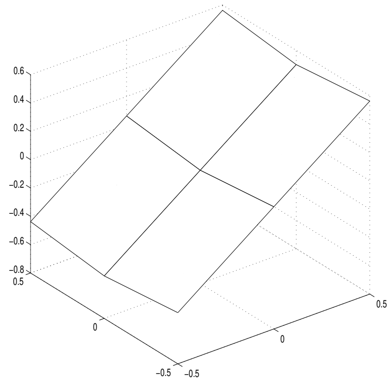

Figure 7: y_2: DFA

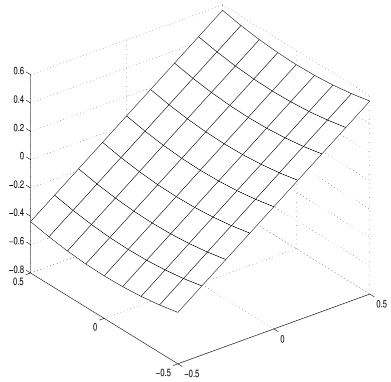

Figure 8: y_2: Analytical

derivatives of the flow on a given grid. Since we know y_1 upto two derivatives, we may compute y_2 and the feedback transformation $u = A(x_1, x_2) + B(x_1, x_2)v$ from

$$
\begin{aligned}
y_2 &= L_f y_1 \\
A &= \frac{-{L_f}^2 y_1}{L_g L_f y_1} \\
B &= \frac{1}{L_g L_f y_1}
\end{aligned}
$$

These may be computed analytically as well and the results are $y_2 = x_2 + \frac{1}{4}{x_1}^2$, $A = 0$ and $B = 1$ since ${L_f}^2 y_1 = 0$ and $L_g L_f y_1 = 1$.

The output y_1 as well as $y_2 = L_f y_1$, $L_g L_f y_1$ and ${L_f}^2 y_1$ were computed on the grid (filled circles) shown in Figure 4 and the results for y_1 and y_2 are shown in Figures 5 - 8. The pointwise error in computation was less than 1% for all computations.

6 Conclusions and Future Work

We presented a computational scheme for nonlinear control design, based on a discrete representation of functions by their values and finitely many derivatives on a discrete set of grid points. We described a hierarchical rectangular grid structure. This has the advantage that while it allows sufficient freedom in placing the grid points in a non-uniform manner the rectangular partitioning facilitates piecewise polynomial interpolation in arbitrary dimensional domains. We showed an example of feedback linearization where the linearizing output and the feedback transformations were computed with reasonable accuracy. Future work in DFA would include approximate feedback linearization, computation of flat outputs of differentially flat systems, and computation of control Lyapunov functions.

References

[1] Scott A. Bortoff. Advanced nonlinear robotic control using digital signal processing. *IEEE Transactions on Industrial Electronics*, 41(1).

[2] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag, 2nd edition, 1989.

[3] R. R. Kadiyala. *AP_LIN: A CAD Tool Box for Nonlinear Control Design*. PhD thesis, University of California, Berkeley, 1992.

[4] M. Rathinam and R.M. Murray. Configuration flatness of lagrangian systems underactuated by one control. *SIAM journal on control and optimization*, 36(1):164–179, January 1998. Also available as Technical Memorandum CIT-CDS 96-006, California Institute of Technology, Pasadena, CA 91125, March 96.

[5] W. Sluis, A. Banaszuk, J. Hauser, and R. M. Murray. A homotopy algorithm for approximating geometric distributions by integrable systems. *Systems & Control Letters*, 27:285–291, 1996.