Querying Databases of Trajectories of Differential Equations: Data Structures for Trajectories

Robert Grossman*

University of Illinois at Chicago

June, 1989

*This research is supported in part by NASA grant NAG2-513. Address: Department of Mathematics, Statistics, and Computer Science, m/c 249, University of Illinois at Chicago, Box 4348, Chicago, IL 60680, (312) 413-2164, U32964@UICVM.UIC.EDU.

Abstract

One approach to qualitative reasoning about dynamical systems is to extract qualitative information by searching or making queries on databases containing very large numbers of trajectories. The efficiency of such queries depends crucially upon finding an appropriate data structure for trajectories of dynamical systems.

Suppose that a large number of parameterized trajectories γ of a dynamical system evolving in $\mathbf{R}^N$ are stored in a database. Let $\eta \subset \mathbf{R}^N$ denote a parameterized path in Euclidean space, and let $||\cdot||$ denote a norm on the space of paths. In this paper, we define a data structure to represent trajectories of dynamical systems and sketch an algorithm which answers queries of the following form:

Query. Return the trajectory γ from the database which minimizes the norm

$$||\eta - \gamma||.$$

1 Introduction

1.1 Queries about Trajectories

Suppose that a large number of parameterized trajectories γ of a dynamical system evolving in $\mathbf{R}^N$ are stored in a database. Let $\eta \subset \mathbf{R}^N$ denote a parameterized path in Euclidean space, and let $||\cdot||$ denote a norm on the space of paths to be specified later. In this paper, we define a data structure to represent trajectories of dynamical systems and sketch algorithms to answer queries of the following forms:

Query 1. Return the trajectory γ from the database which minimizes the norm

$$||\eta - \gamma||.$$

Query 2. Fix $\epsilon > 0$. Return all trajectories γ from the database which satisfy

$$||\eta - \gamma|| \leq \epsilon.$$

Query 3. If η is a segment of trajectory stored in the database, return the entire trajectory γ.

Efficient algorithms to answer these type of queries should prove useful for a number of applications. As an example, consider the path–planning problem for a robotic arm. Suppose that a large number of feasible tra-

jectories of the robotic arm have been stored in a database. Let η be the desired path of the arm. It is not necessary that η be an actual feasible path. Query 1 would return the feasible trajectory γ of the arm which is closest to the desired path η.

An another example, assume the database contains trajectories not from just one dynamical system, but from a parameterized family of dynamical systems. Let γ denote a trajectory which is a periodic orbit. Then Query 2 could be used to gain information about those dynamical systems which also have a periodic orbit. In other words, queries such as these would be useful in extracting qualitative information about dynamical systems.

As a final example, consider a database containing control trajectories for an aircraft, and assume that auxiliary information is attached to each control trajectory in the database. For example, those trajectories which enter into an unstable control regime somewhere along their flight path could be tagged. Let η denote a measured portion of the flight path of the aircraft. Then Query 1 would return the nearest full control trajectory in the database, which would include information about the stability of the trajectory. In other words, the query could be used as part of a supervisory control system.

A database supporting these types of queries on trajectories needs an

efficient means of storing, accessing, searching, and comparing trajectories. In this paper, we describe a data structure for trajectories, a related family of index functions for trajectories, and sketch algorithms to answer the queries above. The work described in this paper is preliminary. For a more complete analysis of the issues raised here, a complexity analysis of the algorithms, and a discussion of implementation issues, see [9].

The data structure we describe is closely related to hashing methods for curves that have been used in computer vision; see [13], [16], and [11]. A related means of extracting qualitative information from dynamical systems is described in [1] and [2].

In Section 2 we review the relevant facts about trajectories of differential equations and define different data structures to store trajectories. In Section 3, we show how these data structures can be used to answer the queries above. Section 4 contains some concluding remarks. The remainder of this introduction discusses this work from the viewpoint of extensible databases.

1.2 Extensible Databases

Imagine an extensible relational database supporting the basic data type of trajectory, as well as the more traditional data types of string, integer, float, etc. The inclusion of this new data type also requires that new operators be

supported, new storage and access methods be developed, and new methods of optimizing queries be found.

For example, it is important that the database support the storage and retrieval of trajectories of an arbitrarily large size. It is also important that operators defined on the entire trajectory be supported, such as operators returning various temporal and spatial averages. Our viewpoint is to look for specific data structures and algorithms to handle the storage, access, and querying of trajectories. This is not the only viewpoint that has been proposed.

Another viewpoint is to change the relational model as little as possible and build a complete extensible DBMS on top of it. For example, Stonebreaker [14] describes a mechanism for a user to register new abstract data types into the University version of Ingres. Because of the built–in access methods used by Ingres, the new data types must occupy a fixed amount of space. Still another viewpoint is to provide a modular and modifiable system on top of which extensible databases for specific applications can be built. The Exodus DBMS described in [7] is an example of such an approach. See [6] for a recent survey covering some of these issues and detailing other approaches.

The queries above would not make very much sense for a traditional

relational database. On the other hand, queries such as these are natural queries for an extensible relational database of trajectories. They are also typical of the queries found in computational geometry; see [12]. Query 1 is an example of a nearest neighbor query, while Query 2 is an example of a fixed–radius near neighbor query. Bentley [4] considers both of the queries for the case of points in Euclidean space.

It is also important to understand that while Query 3 does not make sense in general for spatial curves, it is meaningful for trajectories of differential equations. Recall that a trajectory of a differential equation is determined once the initial condition and defining equation are specified; see [3], for example. This means, in principle, that given any point on the trajectory, that point together with the defining equation is sufficient to determine the entire trajectory. Alternatively, given a sequence of points on the trajectory which is sufficient to specify the form of the differential equation defining the dynamical system, the entire trajectory can be obtained. It is for these reasons that Query 3 makes sense for trajectories of dynamical systems.

Acknowledgements

The author would like to thank Marshall Bern, Diane Greene, John Gilbert, and George Meyer for many useful discussions.

2 Representations of Paths

2.1 Trajectories

In this section, we recall some basic facts and defintions about trajectories of differential equations. Let $D_\mu = \partial/\partial x_\mu$. A *vector field*

$$E = \sum_{\mu=1}^{N} a^\mu D_\mu$$

on $\mathbf{R}^N$ is determined by specifying N functions

$$a_\mu : \mathbf{R}^N \longrightarrow \mathbf{R}.$$

We also denote the vector field by E_a. In this paper, we will consider only vector fields with polynomial coefficients; that is, the functions $a_1, \ldots, a_N$ are all polynomials in $x_1, \ldots, x_N$. A parameterized path

$$\gamma : [t^0, t^1] \subset \mathbf{R} \longrightarrow \mathbf{R}^N$$

is called a *trajectory* of the dynamical system

$$\dot{x}(t) = E_a(x(t)) \tag{1}$$

in case it is the unique solution of the intitial value problem

$$\dot{x}(t) = E_a(x(t)), \qquad x(t^0) = \gamma(t^0). \tag{2}$$

We distinguish between various representations of trajectories.

Spatial Curve Representation. In this representation, the trajectory is viewed simply as the set of points in $\mathbf{R}^N$ which comprise it.

Parameterized Path Representation. In this representation, the trajectory is viewed as a map

$$\gamma : [t^0, t^1] \subset \mathbf{R} \longrightarrow \mathbf{R}^N.$$

We may pass from the spatial curve representation to the parameterized path representation by parameterizing the curve using arclength.

Vector Field/Reference Point Representation. In this representation, we identify the trajectory with a pair (E, R), consisting of a vector field E and a reference or initial point R, where the trajectory is the solution of the initial value problem

$$\dot{x}(t) = E(x(t)), \qquad x(t^0) = R.$$

Note that the Vector Field/Reference Point representation, or VEFREP, is not unique. Indeed, several different vector fields could have a given spatial

curve as a trajectory, while any point along the spatial curve could serve as the initial value. Even so, as we will see in later sections, this representation is convenient for query processing.

2.2 Rectifying Trees

In this section, we give an algorithm whose input is a parameterized path

$$\eta : [t^0, t^1] \subset \mathbf{R} \longrightarrow \mathbf{R}^N,$$

and whose output is a labeled, rooted binary tree. We assume for conveniece that $t^0 = 0$ and $t^1 = 1$; if not, we can reparameterize. We do not assume that η is a trajectory of the dynamical system (1). This tree will be used in later sections to define polynomials $a_1, \ldots, a_N$ with the property that η is close to a trajectory of the dynamical system (1).

To define the tree, we first fix a tolerance $\epsilon > 0$. The tree we define is a subset of the complete rooted binary tree. The *height* of a node is defined to be the length of the unique path connecting the node to the root. The children of the root have height 1, their children height 2, etc. There are 2^k children of height k: number them left to right from 1 to 2^k. We assign two labels to the jth node v from the left at height k:

$$\kappa(v) = (1/2^k)\left(\eta(\frac{j}{2^k}) - \eta(\frac{j-1}{2^k})\right) \in \mathbf{R}^N$$

and

$$\theta(v) = \eta\left(\frac{j-1}{2^k}\right) \in \mathbf{R}^N.$$

We use the following stopping criterion to grow the tree. If a node has children v and v' with labels κ and κ', respectively, and if

$$||\kappa - \kappa'|| \leq \epsilon,$$

then the nodes v and v' are leaves. Here $||\cdot||$ denotes the Euclidean norm. We denote by $T(\eta)$ the tree that arises in this fashion. This tree has a simple interpretation: the θ labels represent points on the path η, while the κ labels represent approximate tangent vectors at those points. The tree is grown until the difference between two adjacent tangent vectors is uniformly small.

2.3 Trees and Interpolating Vector Fields

In the last section, we associated a tree $T(\eta)$ to each path $\eta \subset \mathbf{R}^N$. In this section, we associate a vector field $E(\eta)$ to each path using the tree. The vector field $E(\eta)$ is simply the vector field with polynomial coefficients which interpolates the labels $(\theta(v), \eta(v))$

$$E(\eta)(\theta(v)) = \kappa(v), \tag{3}$$

for all leaves v in $T(\eta)$. Recall that $\theta(v)$ is the point on the curve η corresponding to the node v, and $\kappa(v)$ is the approximate tangent to the curve

at that point.

Suppose that the tree corresponding to the path η has K leaves. Once we fix a vector space basis of K monomials

$$x^{\beta_1}, \ldots, x^{\beta_K},$$

the equations (3) define a vector field

$$E_a = \sum_{\mu=1}^{N} a^\mu D_\mu,$$

where $a_1(x), \ldots, a_N(x)$ are polynomial functions of the form

$$a^\mu = \sum_{j=1}^{K} b_j^\mu x^{\beta_j}$$

and the b_j^μ are scalars. The equations (3) reduce to linear equations for the scalars b_j^μ. To write down these equations, let $v_1, \ldots, v_K$ denote the K leaves of the tree $T(\eta)$, and $\kappa_i(v_j)$, for $i = 1 \ldots, N$ denote the N–components of the vector $\kappa(v_j)$. Then the system (3) becomes

$$\begin{pmatrix} x^{\beta_1}\,|_{\theta(v_1)} & \cdots & x^{\beta_K}\,|_{\theta(v_1)} \\ \vdots & & \vdots \\ x^{\beta_1}\,|_{\theta(v_K)} & \cdots & x^{\beta_K}\,|_{\theta(v_K)} \end{pmatrix} \begin{pmatrix} b^\mu_{\beta_1} \\ \vdots \\ b^\mu_{\beta_K} \end{pmatrix} = \begin{pmatrix} \kappa_\mu(v_1) \\ \vdots \\ \kappa_\mu(v_K) \end{pmatrix},$$

for $\mu = 1, \ldots, N$.

The number of leaves of the tree $T(\eta)$ depends very strongly on the geometry of the path η. As the complexity of the curve grows, so does

the degree of the polynomial coefficients of the vector field $E(\eta)$. For some applications, it is better to impose an upper bound on the degree of the polynomial coefficients of the vector field $E(\eta)$. Let q denote this bound. In this case, we can define the vector field $E(\eta)$ by requiring that the coefficients b_j^μ minimize the quantity

$$\sum_{\text{leaves } v} ||E(\eta)(\theta(v)) - \kappa(v)||, \tag{4}$$

where the minimum is over vector fields with polynomial coefficients of degree less than or equal to q. See Davis [8] for explicit means of finding the coefficients b_j^μ satisfying these equations.

2.4 Reference Points

Let

$$\eta : [t^0, t^1] \subset \mathbf{R} \longrightarrow \mathbf{R}^N$$

be a parametrized path, $T(\eta)$ the corresponding tree, and $E(\eta)$ the associated vector field with polynomial coefficients. Consider the trajectory defined by the initial value problem

$$\dot{x}(t) = E_a(x(t)), \qquad x(0) = \eta(t^0).$$

Let γ denote this trajectory. In general, γ is only an approximation to the path η. Define $R(\eta) \in \mathbf{R}^N$ as follows: if the path γ and the unit sphere in

$\mathbf{R}^N$ intersect, let $R(\eta)$ denote this intersection; otherwise, let $R(\eta)$ denote the closest point between the unit sphere and the trajectory γ.

To summarize, given a path $\eta \subset \mathbf{R}^N$, we have defined a rectification tree $T(\eta)$, a vector field $E(\eta)$, and a reference point $R(\eta) \in \mathbf{R}^N$, with the property that there is a segment of the trajectory γ which approximates the path η, where γ is the solution of the initial value problem

$$\dot{x}(t) = E(\eta)(x(t)), \qquad x(0) = R(\eta). \tag{5}$$

The pair $(E(\eta), R(\eta))$ is called the VEFREP representation of the path η.

It is now easy to define an index or invariant associated with the path η. First, fix injective functions

$$h_n : \mathbf{R}^n \longrightarrow \{1, 2, \ldots\},$$

for each $n = 1, 2, \ldots$ Next, view the coefficients of the vector field $E(\eta)$ as a $K \cdot N$ vector, so that the pair $(E(\eta), R(\eta))$ has $(K+1)N$ components. Then the *index* associated with η is defined by

$$h(\eta) = h_{(K+1)N}(E(\eta), R(\eta)).$$

This can be used to hash the trajectory; see [9].

3 Returning the Nearest Trajectory

3.1 Proximity Problems

In this section, we show that the VEFREP representation of a path reduces Queries 1 and 2 to the following queries:

Query $1'$. Given P points in $\mathbf{R}^Q$, preprocess the points in time Preprocess(P) so that given a query point we can determine in time Query(P) the point of the original set which is closest to it, using storage Storage(P).

Query $2'$. Given P points in $\mathbf{R}^Q$, preprocess the points in time Preprocess(P) so that given a query point we can determine in time Query(P) all points of the original set which are within a distance of ϵ' of it, using storage Storage(P).

Query $1'$ is called the nearest neighbor problem, and Query $2'$ is called the fixed radius near neighbor problem. For a general reference to proximity problems such as these, see [12]. This reference also describes algorithms to solve these types of problems using techniques which we do not mention here, such as Voronoi diagrams. The naive exhaustive algorithm requires

$$\begin{aligned} \text{Storage}(P) &= O(P) \\ \text{Query}(P) &= O(QP). \end{aligned}$$

In [4], Bentley shows, with the assumption of sparsity, how to use a multi-dimensional divide–and–conquer algorithm to achieve bounds

$$\begin{aligned} \text{Preprocess}(P) &= O(P \log^{Q-1} P) \\ \text{Storage}(P) &= O(P \log^{Q-1} P) \\ \text{Query}(P) &= O(\log^{Q} P). \end{aligned}$$

In [5], Bentley et. al., use a cell method and the assumption of sparsity to achieve expected bounds of

$$\begin{aligned} \text{Preprocess}(P) &= O(P) \\ \text{Query}(P) &= O(1), \end{aligned}$$

assuming that the original points are drawn independently from a uniform distribution of a bounded region.

Both of these algorithms have constants which grow exponentially with the dimension. Because of this, it is doubtful whether either one could lead to practical algorithms to answer Queries 1 and 2.

3.2 The Algorithm

In this section, we sketch an algorithm to answer Queries $1-3$. For further details, see [9].

Suppose that $\gamma_1, \ldots, \gamma_P$ are trajectories of the dynamical system

$$\dot{x}(t) = E_a(x(t)),$$

as a ranges over some parameter space. To each such trajectory γ, let

$$(E(\gamma), R(\gamma))$$

denote its VEFREP representation. We define a norm on the space of trajectories via

$$||\gamma|| = ||(R(\gamma), E(\eta))||,$$

where the norm on the right hand side is the standard Euclidean norm. Given a parameterized path η, Algorithm 1 below returns from the database the trajectory γ which contains a segment which is closer to η than any other segment from any other trajectory in the database. Notice that this provides an answer to both Query 1 and Query 3. Query 2 is handled in an analogous manner; again, see [9] for details.

Algorithm 1. The input is a parameterized path η, and the output is the trajectory γ from the database answering Query 1. Fix $\epsilon > 0$ and $q > 1$. Compute the vector field E of the VEFREP representations (E, R) in the algorithm using Equation 4.

Step 1. This step is a precomputation. For each trajectory γ_i, $i = 1, \ldots, P$, compute its VEFREP representation $(E(\gamma_i), R(\gamma_i))$.

Step 2. Given a query path η, compute its VEFREP representation $(E(\eta), R(\eta))$.

Step 3. Solve the nearest neighbor problem for query point $(E(\eta), R(\eta))$, given the set of points

$$\{(E(\gamma_i), R(\gamma_i)) : i = 1, \ldots, P\}.$$

Let (E, R) denote the resulting nearest neighbor.

Step 4. Compute the trajectory γ which is the solution to the initial value problem

$$\dot{x}(t) = E(x(t)), \qquad x(0) = R.$$

4 Conclusion

In this paper, we have described preliminary work concerned with queries of databases containing trajectories of differential equations. Our eventual goal is to build an extensible, relational database supporting these types of queries.

Trajectories of differential equations have many different representations. For the types of queries considered here, we have chosen to represent parameterized trajectories

$$\gamma : [t^0, t^1] \subset \mathbf{R} \longrightarrow \mathbf{R}^N$$

by a pair, consisting of a vector field E on $\mathbf{R}^N$ with polynomial coefficients and a point $R \in \mathbf{R}^N$ such that the trajectory is the solution of the initial value problem:

$$\dot{x}(t) = E(x(t)), \qquad x(t^0) = R.$$

We call this a VEFREP representation.

Using the VEFREP representation, we have reduced Query 1 and Query 2 to standard problems in the computational geometry of points in Euclidean space, the nearest neighbor problem, and the near neighbor problem, respectively.

Much work remains to be done: the efficiency and implementation of the algorithm needs to be studied. We do not claim that the algorithm described here is necessarily the most practical for querying large databases of trajectories. On the other hand, it is simple enough that its properties can easily be understood and investigated. It is our hope that this will lead to more practical algorithms.

References

[1] H. Abelson and G. J. Sussman, Dynamicists' Workbench I: "Automatic Preparation of Numerical Experiments," R. Grossman (editor), *Sym-*

bolic Computation: Applications to Scientific Computing, SIAM, 1989.

[2] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G. Sussman, J. Wisdom, and K. Yip, "Intelligence in Scientific Computing," Comm. ACM, **32** (1989), 546–562.

[3] V. I. Arnold, *Ordinary Differential Equations*, MIT, Boston, 1973.

[4] J. L. Bentley, "Multidimensional Divide–and–Conquer," Comm. ACM, **23** (1980), 214–229.

[5] J. L. Bentley, B. W. Weide, and A. C. Yao, "Optimal Expected–Time Algorithms for Closest Point Problems," ACM Transactions on Math. Software, **6** (1980), 563–580.

[6] M. J. Carey (editor), "Special Issue on Extensible Database Systems," *Database Engineering*, June, 1987.

[7] M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson, E. J. Shekita, "The Architecture of the EXODUS Extensible DBMS," *Proceedings of the Object–Oriented Database Workshop*, ACM, 1986, 52–65.

[8] P. Davis, *Interpolation and Approximation*, Dover, New York, 1975.

[9] R. Grossman, "Querying Databases of Trajectories of Differential Equations: Searching," in preparation.

[10] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer, 1984.

[11] E. Kishon and H. Wolfson, "3–D Curve Matching," *Courant Institute of Mathematical Sciences Technical Report* 283 (1987).

[12] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer, New York, 1985.

[13] J. T. Schwartz and M. Sharir, "Identification of Partially Obscured Objects in Two Dimensions by Matching of Noisy *Characteristic Curves*," International J. Robotics Research, **6** (1987), 29–44.

[14] M. Stonebreaker, "Inclusion of New Types in Relational Data Base Systems," *Proceedings of IEEE/Data Engineering*, IEEE, 1986, 262–269.

[15] J. D. Ullman, *Principles of Database and Knowledge–Base Systems, Volume 1*, Computer Science Press, Rockville, MD, 1988.

[16] H. Wolfson, "On Curve Matching," *Proceedings of Workshop on Computer Vision, Miami Beach*, IEEE, 1987, 307–310.