BUILDING LOCAL EQUATIONS OF MOTION TO MODEL CHAOTIC DATASETS

COOPER CROSBY

Abstract. The nature of chaotic dynamical systems makes predicting the future behavior of such systems difficult, specifically in the case of discrete datasets. This paper will present some methods to overcome this problem. By deconstructing the data into locally linear clusters and building low dimensional bases for each cluster, radial basis functions can be used to rebuild the dynamics of the system and predict future behavior.

Introduction

This paper will serve as an overview on how to build locally linear models of motion for a nonlinear dynamical system. We will discuss the notion of a dynamical system, and give an overview of the process used to produce the equations of motion from a discrete data set.

As we will see, data generated by chaotic systems is inherently difficult to predict because the behavior of future iterations of the system can vary drastically given different initial conditions. In this paper we will be considering the discrete time series generated by chaotic systems. The goal will be to outline the process for which we may discover the local equations of motion for such a system. Then, a given data point may be taken and iterated through the system according to its dynamics, such that the behavior of the system models the characteristics of the given series of data; in other words, the dynamics have been modeled.

Following a discussion on chaotic dynamical systems to better understand how and why they work, we will look at the components of the method. The data is clustered into local sets whose dynamics are linear using the Linde-Buzo-Gray algorithm, and the singular value decomposition is used to build a lower dimensional representation for each cluster. Radial basis functions are used to build the local equations of motion, so that a point may be clustered, projected to a lower dimension, and moved through the system in order to model its behavior.

1. What Is Chaos?

At its core, chaos theory is the study of nonlinear dynamics, where the future behavior of seemingly random data arises from relatively simple deterministic equations. It is important to distinguish our usage of the word 'chaos' from its generally understood meaning as a state of confusion lacking any order. The word 'chaos' refers to a system which, while apparently lacking order, actually follows a prescribed set of rules. Such a dynamical system is said to be *deterministic* and has the form

$$\mathbf{x}' = F(\mathbf{x}) \qquad \text{or} \qquad \mathbf{x}_{n+1} = F(\mathbf{x}_n)$$

for the continuous and discrete cases, respectively, where F is a function from $\mathbb{R}^n$ to $\mathbb{R}^n$. Such a system is said to be **chaotic** if it has the following properties:

Aperiodic: The same state, or configuration of information or data, is never repeated.

Bounded: On successive iterations the state remains in a finite range and does not grow without bound.

Deterministic: There is a definite rule with no stochastic terms governing the dynamics.

Sensitive Dependence on Initial Conditions: Two points that are initially close will drift apart as time proceeds.

These heuristic definitions might make the reader recall some popular examples of chaos. One such example is the weather.

Example 1. Because it is a chaotic system, meteorologists struggle to predict the weather more than a few days in advance. Since it is impossible to know the exact initial conditions of a weather system, future predictions for the behavior of the system over time deviate exponentially from the actual behavior.

There are some common misconceptions regarding chaos. For instance, simply because two points that are initially close drift apart does not guarantee that a system is chaotic.

Example 2. Consider the solutions of $x' = 3x$, given by $x(t) = ce^{(3t)}$:

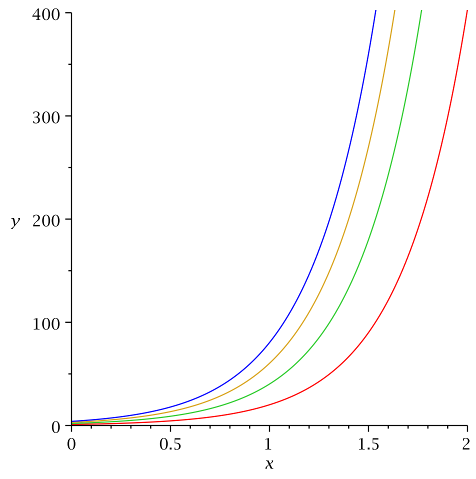

FIGURE 1. Solution curves of $x' = 3x$ for initial conditions $(0.4, 0)$, $(0.3, 0)$, $(0.2, 0)$, $(0.4, 0)$.

We see from Figure 1 that points that are initially drift apart over time; it may seem at first that the system is chaotic. But recalling our earlier definition of chaos, we see that this system is not, in fact, chaotic. With the exception of $(0, 0)$, the system grows without bound for all initial conditions.

The term 'chaos' is often confused with 'random.' And it might be easy to see why. Given the unpredictable nature of chaotic systems, their behavior can indeed appear at first to be random. But in fact there is an important difference between the two.

Example 3. Consider the plot of randomly generated data:

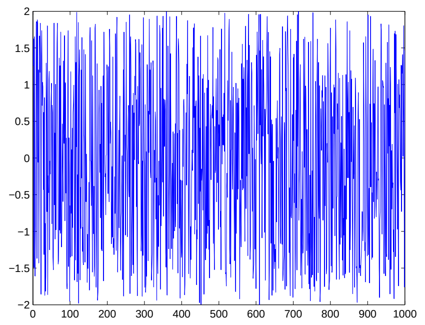

FIGURE 2. A series of randomly generated data

The data could be mistaken for being chaotic – it looks aperiodic, bounded, and future iterations certainly appear unpredictable. But given that one of the fundamental aspects of chaos is its deterministic nature, it is impossible for randomly generated data to be chaotic.

These examples serve to clarify the type of data with which this project will consider: the data arising from chaotic systems. But at this point it becomes apparent that while the definitions of chaos put forth so far make intuitive sense, they are less useful if we are handed a set of data and asked whether or not it is chaotic. For example, what if we had not known already that Figure 2 had been randomly generated? We thus present a new method to describe in greater technical detail the stochasticity of a discrete system [1].

Theorem 1. *Let $J = [a, b]$ and suppose that $f : J^n \to J^n$ and $x_{n+1} = f(x_n)$. Then f has point-wise* ***sensitive dependence on initial conditions at x*** *if there is an $\epsilon > 0$ such that for each $\delta \geq 0$, there is a y in J and a positive integer n such that, given $x \in J$,*

$$||x - y|| < \delta \quad \textit{and} \quad ||f^{[n]}(x) - f^{[n]}(y)|| > \epsilon \ .$$

The initial conditions in the definition refer to the given points x and y, and $f^{[n]}$ to the nth iteration of f. While this presents a more formal way of thinking of dependence on initial conditions, finding an ϵ to satisfy the requirements is not always a simple matter. Showing that a system does indeed have dependence on initial conditions can be difficult. We thus introduce a second method for describing the manner in which neighboring points separate iteratively, called Lyapunov exponents.

Consider the dynamical system $f : J \to J$ where $x_{n+1} = f(x_n)$. We assume that for each x in the interior of J and each small $\epsilon > 0$, there is a number $\lambda(x)$ that depends on x such that for each positive integer n,

$$|f^{[n]}(x+\epsilon) - f^{[n]}(x)| \approx (e^{\lambda(x)})^n |x + \epsilon - x|.$$

This implies that

$$\begin{aligned} e^{n\lambda(x)} &= \lim_{\epsilon \to 0} \frac{|f^{[n]}(x+\epsilon) - f^{[n]}(x)|}{\epsilon} \\ &= \frac{d}{dx}(f^{[n]}(x)). \end{aligned}$$

By taking logarithms and dividing by n we obtain

$$\lambda(x) = \frac{1}{n} \ln \left| \frac{d}{dx}(f^{[n]}(x)) \right|$$

which leads to the following definition of Lyapunov exponents [1]:

Definition 1. *Let J be a bounded interval, and $f : J \to J$ continuously differentiable on J. Fix x in J, and let $\lambda(x)$ be defined by*

$$\lambda(x) = \lim_{n \to 0} \frac{1}{n} \ln \left| \frac{d}{dx}(f^{[n]}(x)) \right|$$

*provided that the limit exists. In that case, $\lambda(x)$ is the **Lyapunov exponent of f at x**. If $\lambda(x)$ is independent of x wherever $\lambda(x)$ is defined, then the common value of $\lambda(x)$ is denoted by λ, and is the **Lyapunov exponent** of f.*

The number $\lambda(x)$ can be thought of as a measure of the average loss of information of successive iterates of points near x. Furthermore, if y is near x and the iterates of x and y remain close together, then $\lambda(x)$ is likely to be negative because of the presence of the logarithm. In the same manner, if the iterates separate from one another, $\lambda(x)$ will be positive. We arrive at the following definition of chaos [1]:

Definition 2. *A dynamic system f is **chaotic** if it satisfies at least one of the following conditions:*

i The function f has sensitive dependence on initial conditions on its domain.

ii The function f has a positive Lyapunov exponent at each point in its domain that is not eventually periodic.

The Lyapunov exponent thus gives a measure of the chaos of a system. If the system is chaotic and the long term dynamics lie in a small dimensional region, then we may try to construct local models to describe that motion. Because a chaotic system has aperiodic, bounded motion, such systems are ideal candidates for building local models of motion. To do this, though, requires a systematic way of determining the placement of the local coordinate systems from which the local models can be built. We thus introduce data clustering.

2. Data Clustering and the Linde-Buzo-Gray Algorithm

Data clustering is the assignment of a set of data points into smaller subsets, or data clusters, so that the behavior of data points within a subset is related in some way. In dealing with large data sets, data clustering can provide a useful way of breaking down the data into smaller packages so that it is more easily managed. For any algorithm that performs data clustering, we are given a data set X whose elements are vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$. To cluster the data, we want a membership function m with a domain in $\mathbb{R}^n$ that will output the cluster index. Then $m(\mathbf{x}^{(i)}) = g_i$ where g_i is the integer for the class of the data point.

For the purposes here, it will be especially useful to have a function that will identify the points in a given cluster. The characteristic function $\mathcal{C}$ is typically defined as 0 or 1 depending on whether or not a specific data point belongs to the cluster, so that

$$\mathcal{C}(\mathbf{x}) = \begin{cases} 1 & \text{if } m(\mathbf{x}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We thus have a means of describing whether a data point is contained in a given cluster, but lack a means of describing exactly what a cluster is. An easy way to do this is using Voronoi Clusters [2].

Theorem 2. *Let $\{\boldsymbol{c}^{(i)}\}_{i=1}^{k}$ be points in $\mathbb{R}^n$. These points form k Voronoi Cells, where the $\boldsymbol{j}^{th}$ cell is defined as the set of points that are closer to cell $\boldsymbol{j}$ than any other cluster:*

$$V_j = \{x \in \mathbb{R}^n: \qquad ||\boldsymbol{x} - \boldsymbol{c}^{(i)}|| < ||\boldsymbol{x} - \boldsymbol{c}^{(j)}||, \qquad i = 1, 2, \ldots, k\}$$

The points $\{\mathbf{c}^{(i)}\}_{i=1}^{k}$ are called **cluster centers**. In the rare case that a point $\mathbf{x}$ lies equally far apart from two cluster centers, it is generally included in the cluster whose index is smaller.

In creating an algorithm to perform data clustering, it would be useful to be able to determine how effective the algorithm is. One way to do this is to measure the **distortion error** for a given cluster i, as well as the total distortion error, given by

$$E_i = \tfrac{1}{N_i} \textstyle\sum_{j=1}^{N} ||\mathbf{x}^{(j)} - \mathbf{c}^{(i)}||^2 \chi_i(\mathbf{x}^{(j)}), \quad E_{\text{total}} = \textstyle\sum_{j=1}^{p} E_j.$$

For example, if the data is given by p points $\{x_1, x_2, \ldots, x_p\}$, we can find c that minimizes the distortion error. The error is given by

$$E = \frac{1}{p} \sum_{i=1}^{p} (x_i - c)^2.$$

Taking the derivative with respect to c, we have

$$\frac{\partial E}{\partial c} = \frac{1}{p} \sum_{i=1}^{p} 2(x_i - c).$$

The error is minimized by setting $\frac{\partial E}{\partial c} = 0$; expanding the sum gives

$$\begin{aligned} 0 &= \frac{1}{p}\left[2(x_1 + \ldots + x_p) - 2pc\right] \\ 0 &= 2(x_1 + \ldots + x_p) - 2pc \\ pc &= (x_1 + \ldots + x_p) \\ c &= \frac{(x_1 + \ldots + x_p)}{p} \end{aligned}$$

We see that the distortion error is minimized by choosing the center to be the mean of the data.

The Linde-Buzo-Gray (LBG) algorithm is a particular algorithm used in data clustering, which is generally quick and easy to visualize. The algorithm's membership function is defined so that we are in cluster i if we are in the Voronoi cell defined by the i^{th} cluster center. That is,

$$m(\mathbf{x}) = i \text{ iff } ||\mathbf{x} - \mathbf{c}^{(i)}|| < ||\mathbf{x} - \mathbf{c}^{(j)}|| \; i \neq j,\, 1 \leq j \leq p \tag{1}$$

Let X be a matrix of p data points in $\mathbb{R}^n$ and let C be a matrix of k cluster centers in $\mathbb{R}^n$. The LBG algorithm works by choosing the centers to be randomly selected data points, which it updates iteratively using the function in (1). At each pass of the algorithm, the function calculates $p \times k$ distances and performs p sorts in which each point is assigned to a cluster center. The centers are updated using this process to minimize E_{total}, so that after each iteration the center $\mathbf{c}_i$ is reset as the center of the data of the i^{th} set and the distortion error is computed. The algorithm repeats this process until the distortion error no longer continues to decrease, at which time it has calculated each of the i centers as well as the data points associated with those centers.

We consider one final criteria in clustering the data: that data points in the same cluster are all moving in the same direction. Since the dot product of two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by $\mathbf{x}_i \cdot \mathbf{x}_j = |\mathbf{x}_i||\mathbf{x}_j|\cos\theta$, it will be used to verify that a data point x_i is associated with a cluster center x_j only if the two points are moving in the same direction. The vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are formed from $x_i,\ x_{i+1}$ and $x_j,\ x_{j+1}$. Since the vectors are pointing in the same direction if the angle between them is 0, $\mathbf{x}_i$ is with cluster center $\mathbf{x}_j$ if

$$|\cos\theta| = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{|\mathbf{x}_i||\mathbf{x}_j|} = 1.$$

It is unlikely that two points are pointing in exactly the same direction, so we allow $|\cos\theta| \geq 0.5$.

3. The Singular Value Decomposition

With the data clustered, we would like to construct local low dimensional coordinate systems for the data so from which the local equations of motion can be built. To construct these local coordinate systems, we need to find bases for each cluster. A critical tool in this process will be the singular value decomposition (SVD). The SVD is an extremely useful tool in both linear algebra and data processing for many reasons, not least of which is because it can be used to find bases for the four fundamental subspaces of a matrix – the column space, the row space, the null space, and the null space of the transpose.

Recall that a symmetric matrix is one which is equal to its transpose. Before discussing the SVD, we introduce the following theorem for symmetric matrices [4]:

Spectral Theorem for Symmetric Matrices. An n x n symmetric matrix (a square matrix that is equal to its transpose) A has the following properties:

a. A has n real eigenvalues, counting multiplicities.

b. The dimension of the eigenspace for each eigenvalue λ equals the multiplicity of λ as a root of the characteristic equation.

c. The eigenspaces are mutually orthogonal, in the sense that the eigenvectors corresponding to different eigenvalues are orthogonal.

d. A is orthogonally diagonalizable.

The theorem's use will be more apparent as we continue our discussion of the singular value decomposition. We begin by defining the **singular values** of an

$m \times n$ matrix A to be the square roots of the eigenvalues of A^TA. Note that $(A^TA)^T = A^T(A^T)^T = A^TA$, so A^TA is symmetric. The singular values are denoted by $\sigma_1, \ldots, \sigma_n$ and arranged in decreasing order. Additionally, Σ is the $m \times n$ matrix

$$\Sigma = \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix}$$

where D is the $r \times r$ diagonal matrix for which the diagonal entries are the first r singular values of A, $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > 0$. We now define the singular value decomposition [4].

The Singular Value Decomposition. Let A be an m x n matrix with rank r. Then there exists an m x n matrix Σ as above, an m x m orthogonal matrix U and an n x n orthogonal matrix V such that

$$A = U\Sigma V^T.$$

This factorization is called the singular value decomposition of A and involves calculating the eigenvectors of A^TA and AA^T. The columns of V are given by the eigenvectors of A^TA; the columns of U are given by the eigenvectors of AA^T. Consider the following example:

Example. Find a singular value decomposition of $\begin{bmatrix} 2 & -1 \\ 2 & 2 \end{bmatrix}$.

First, compute A^TA=$\begin{bmatrix} 8 & 2 \\ 2 & 5 \end{bmatrix}$. The eigenvalues of A^TA are 9 and 4, with corresponding unit eigenvectors

$$\mathbf{v}_1 = \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{5} \\ -2/\sqrt{5} \end{bmatrix}$$

These unit vectors form the columns of V:

$$V = \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ 1/\sqrt{5} & -2/\sqrt{5} \end{bmatrix}$$

In a similar way, we find that

$$U = \begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \\ 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix}$$

The singular values are $\sigma_1 = 3$ and $\sigma_2 = 2$. Then the matrix Σ is

$$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

and the singular value decomposition of A is

$$A = \begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5} \\ 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2/\sqrt{5} & 1/\sqrt{5} \\ 1/\sqrt{5} & -2/\sqrt{5} \end{bmatrix}$$

In dealing with large matrices, the usefulness of the singular value decomposition can diminish because of the size of U, Σ, and V. In such instances we can find a reduced rank approximation (or reduced SVD) to the original matrix by setting all but the first k singular values equal to zero and using only the first k columns of U and V.

Example Consider the 240×320 matrix A whose entries are integers between 0 and 63. Each integer denotes a specific color such that the entire matrix can be represented by a single image. By decomposing A into its singular value decomposition and producing the same image using only the first k columns of U, Σ, and V^T, we produce the images in Figure 3. Note that the leading k columns, associated with the largest singular values, first highlight the main features of the data before filling out other aspects.

Furthermore, the singular value decomposition can be used to construct the Moore-Penrose pseudoinverse of a matrix, referred to from now on as simply the **pseudoinverse**. The pseudoinverse is defined so that if the SVD is used to decompose A as $A = U\Sigma V^T$, the pseudoinverse of A is given by $A^+ = V\Sigma^{-1}U^T$ and is a

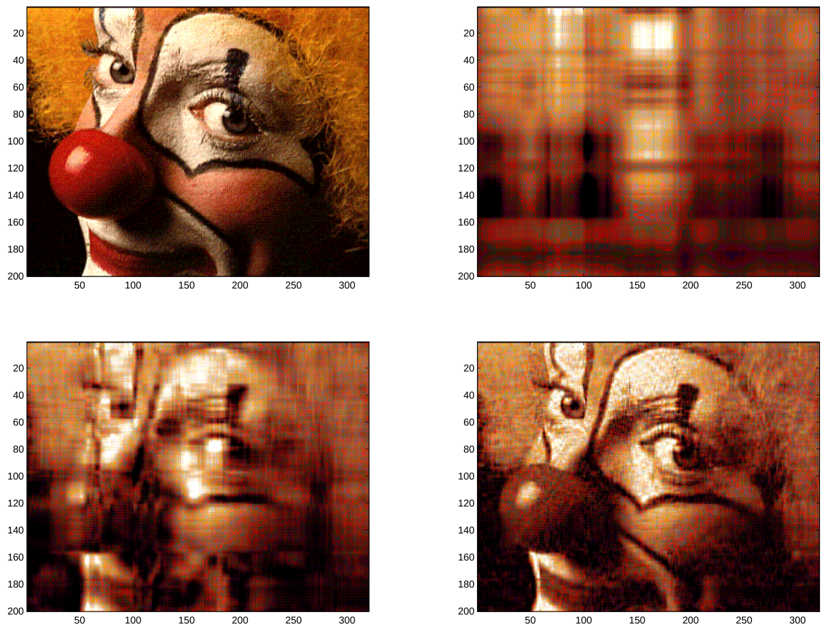

FIGURE 3. Clockwise from top-left: The image generated by A and the images generated using the SVD of A with $k = 2$, 10, and 30.

general solution to the following system of linear equations:

$$\mathbf{y} = A\mathbf{x} \qquad \mathbf{y} \in \mathbb{R}^m; \quad \mathbf{x} \in \mathbb{R}^n.$$

Although the proof will be omitted here, Moore and Penrose showed that the matrix A^+ is the unique matrix that satisfies the following properties [5]:

(1) $AA^+A = A$
(2) $A^+AA^+ = A^+$
(3) $(AA^+)^T = AA^+$
(4) $(A^+A)^T = A^+A$

Later we will see how the pseudoinverse helps build local equations of motion for the data.

4. The Karhunen-Loève Expansion

The singular value decomposition gives one way of building a basis for a set of data using the columns of U or V. We will now discuss how to build the *best* basis, including what is meant by the best basis. For a data point $\mathbf{x}$ and a basis ϕ for that data point, the notation used to denote the inner product is

$$\alpha_k^{(i)} = \langle \mathbf{x}, \boldsymbol{\phi}_k \rangle . = \mathbf{x}^T \boldsymbol{\phi}_k$$

where each α is the coordinates of $\mathbf{x}$ with respect to ϕ. Then, given a set of data

$$\left\{ \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)} \right\}$$

where each $\mathbf{x}^{(i)} \in \mathbb{R}^n$, let X be the $p \times n$ matrix formed from the data so that

$$X = \left[\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)} \right]^T .$$

Given an orthonormal basis of $\mathbb{R}^n$, for example the vectors in $\mathbf{\Phi} = [\phi_1 \ldots \phi_n]$, expand $\mathbf{x}$ in terms of the basis to obtain

$$\mathbf{x}^{(i)} = \sum_{k=1}^{n} \alpha_k \phi_k .$$

The basis can be rewritten so that

$$\mathbf{x}^{(i)} = \sum_{j=1}^{k} \alpha_j \phi_j + \sum_{j=k+1}^{n} \alpha_j \phi_j .$$

Then the error in using k basis vectors is given by the second term of the sum, so that

$$\mathbf{x}_{\text{err}}^{(i)} = \sum_{j=k+1}^{n} \alpha_j \phi_j .$$

The error can be rewritten using a single data point $\mathbf{x}$ as

$$\|\mathbf{x}_{\text{err}}\|^2 = \sum_{j=k+1}^{n} \alpha_j^2 .$$

One term of the previous sum can be written as

$$\alpha_j^2 = \langle \phi_k, \mathbf{x}\mathbf{x}^T \phi_k \rangle,$$

And the inner product can be switched so that

$$\begin{aligned}
\sum_{i=1}^{p} \langle \phi, \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^T \phi \rangle &= \langle \phi, \mathbf{x}^{(1)} \left(\mathbf{x}^{(1)}\right)^T \phi \rangle + \langle \phi, \mathbf{x}^{(2)} \left(\mathbf{x}^{(2)}\right)^T \phi \rangle + \ldots + \langle \phi, \mathbf{x}^{(p)} \left(\mathbf{x}^{(p)}\right)^T \phi \rangle \\
&= \langle \phi^T \mathbf{x}^{(1)} \left(\mathbf{x}^{(1)}\right)^T \phi \rangle + \langle \phi^T \mathbf{x}^{(2)} \left(\mathbf{x}^{(2)}\right)^T \phi \rangle + \ldots + \langle \phi^T \mathbf{x}^{(p)} \left(\mathbf{x}^{(p)}\right)^T \phi \rangle \\
&= \phi^T \left(\mathbf{x}^{(1)} \left(\mathbf{x}^{(1)}\right)^T + \mathbf{x}^{(2)} \left(\mathbf{x}^{(2)}\right)^T + \ldots + \mathbf{x}^{(p)} \left(\mathbf{x}^{(p)}\right)^T \right) \phi \\
&= \phi^T \left[\sum_{i=1}^{p} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^T \right] \phi \\
&= \langle \phi, \left[\sum_{i=1}^{p} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^T \right] \phi \rangle.
\end{aligned}$$

The covariance matrix of X is given by

$$\begin{aligned}
C &= \frac{1}{p} X^T X \\
&= \frac{1}{p} \left[\mathbf{x}^{(1)} \left(\mathbf{x}^{(1)}\right)^T + \ldots + \mathbf{x}^{(p)} \left(\mathbf{x}^{(p)}\right)^T \right] \\
&= \frac{1}{p} \sum_{i=1}^{p} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^T .
\end{aligned}$$

The mean-square error of the data expansion is written as

$$\frac{1}{p} \sum_{i=1}^{p} \|\mathbf{x}_{err}\|^2$$

and, by making use of the above facts and some simple substitutions, it can be shown that

$$\frac{1}{p} \sum_{i=1}^{p} \|\mathbf{x}_{err}\|^2 = \sum_{j=k+1}^{n} \langle \phi_j, C\phi_j \rangle.$$

To minimize the error in building a kth dimensional basis for the data, minimize

$$\sum_{j=k+1}^{n} \langle \phi_j, C\phi_j \rangle.$$

The basis is orthonormal, so the error is minimized at

$$\max_{\phi \neq 0} \frac{\phi^T C \phi}{\phi^T \phi}.$$

The maximum of $\frac{\phi^T C\phi}{\phi^T \phi}$ occurs at the first eigenvector of C, where λ_1 is the largest eigenvalue of C. This leads directly into the Best Basis Theorem [2]:

Theorem 3 (The Best Basis Theorem)**.** *Suppose that*

- *X is a $p \times n$ matrix of p points in $\mathbb{R}^n$.*
- *X_m is the $p \times n$ matrix of mean-subtracted data.*
- *C is the covariance of X, $C = \frac{1}{p} X_m^T X_m$.*

Then the best orthonormal k-dimensional basis is given by the leading k eigenvectors of C, for any k.

One of the useful properties of the Best Basis Theorem is that given a set of data, the best basis can be calculated simply by finding the covariance matrix and taking the first k eigenvectors. The process is simplified further by noting a few nifty connections to the singular value decomposition of X, given by $X = U\Sigma V^T$ so that Σ is $k \times k$, where k is the rank of X. The covariance becomes

$$C = \frac{1}{p} X^T X = \frac{1}{p} V \Sigma U^T U \Sigma V^T = V \left(\frac{1}{p} \Sigma^2 \right) V^T$$

and, comparing this to the Best Basis Theorem, the best basis vectors for the row space of X are the right singular vectors for X. Thus, in finding the best basis, it is sufficient to decompose the matrix into its singular value decomposition and take its right singular vectors.

The value of k, which can be thought of as the dimension of the basis, still needs to be determined. Which value should be chosen? From the clown example, we know that choosing a larger value of k, gives a closer approximation to the data.

At the same time, choosing a smaller k saves time and computing power. One way to choose k is by considering the singular values of the matrix.

The singular values describe how the data is distributed. If the singular values are normalized such that the jth normalized singular value is given by

$$\hat{\sigma}_j = \frac{\sigma_j}{\sigma_1 + \sigma_2 + \ldots + \sigma_k},$$

each successive singular value can be thought of as the percentage of data associated with that singular value. Because the singular values are arranged in decreasing order, more data is associated with the first singular vector than with the second, and so on. From this logic it follows that the percentage of the data represented by the first k singular values is given by the cumulative sum of the first k normalized singular values. Then by normalizing the singular values, we find that the percent of data that is retained by restricting the model to k dimensions is given by the sum of the first k singular values.

Finally, the low dimensional basis can be used to give the low dimensional coordinates of the local cluster. If $\mathbf{x}$ is a point in $\mathbb{R}^n$ and B is the $n \times m$ basis in $\mathbb{R}^m$ for the local cluster to which $\mathbf{x}$ belongs, then $B^T\mathbf{x}$ is the m-dimensional representation of the coordinates of $\mathbf{x}$. In this way, the basis can be used to build a low dimensional representation for the coordinates of the entire cluster.

5. Radial Basis Functions

With low dimensional coordinate systems built for each local cluster, all that remains is to actually build the local equations of motion. This is accomplished using radial basis functions. A radial basis function (RBF) is a general model to build a mapping from $\mathbb{R}^m$ to $\mathbb{R}^n$. Before discussing radial basis functions, we will consider the following type of matrix [2]:

Definition 3. *Let $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P}$ be a set of points in $\mathbb{R}^n$. Let the matrix A be the $P \times P$ matrix such that the $(i,j)^{th}$ entry of A is given by*

$$A_{ij} = ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}||^2.$$

*Then A is the **Euclidean Distance Matrix** for the data set $\{\boldsymbol{x}^{(i)}\}_{i=1}^{P}$.*

Let $\phi : \mathbb{R}^+ \to \mathbb{R}$ be chosen from the following list:

$\phi(r,\sigma) = e^{(-r^2/\sigma^2)}$	Gaussian
$\phi(r) = r^3$	Cubic

Then ϕ is a **transfer function** [2]. If ϕ is applied elementwise to a matrix, then Φ is the **transfer matrix**. Transfer functions other than the ones listed here are occasionally used in modeling, but they are omitted as they are not used in this project.

Let $\{\mathbf{x}^{(i)}\}_{i=1}^{P} \subset \mathbb{R}^n$ and the data set (in this case, the targets to which we are mapping) $\{\mathbf{y}^{(i)}\}_{i=1}^{P} \subset \mathbb{R}^m$. The interpolation problem that will be solved using RBFs is to determine a function F such that $\mathbf{y}^{(i)} = F(\mathbf{x}^{(i)}), \quad i = 1 \ldots P$. We will assume that F is defined so that the j^{th} element of $F(\mathbf{x}^{(i)})$ is given by

$$F_j(\mathbf{x}^{(i)}) = \sum_{k=1}^{P} \omega_{kj} \phi \left(||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}|| \right) + b_j$$

where the ω_{kj} parameters are to be determined, i runs from 1 to P, and j runs from 1 to M. The problem can be rewritten as the following matrix equation, for which the constants $\omega_{kj} \in W$, a $P \times M$ matrix, need to be solved:

$$\Phi W + \mathbf{b} = Y.$$

The reader will note that this equation is an affine mapping from $\mathbb{R}^m$ to $\mathbb{R}^n$. For the mapping to be linear, let $\hat{W} = [W \ \mathbf{b}]$, so that $\hat{W}$ is the matrix W with the vector $\mathbf{b}$ appended as its last column. Let $\hat{\Phi} = [\Phi \ 1]^T$ so that $\hat{\Phi}$ is the matrix Φ

with a vector of ones appended as its last row. The matrix equation then becomes

$$\hat{W}\hat{\Phi} = Y$$

so that the vector $\mathbf{b}$ can also be solved.

If, rather than computing the full EDM, the model of the function F is replaced so that

$$F_j(\mathbf{x}^{(i)}) = \sum_{k=1}^{P} \omega_{kj}\phi\left(||\mathbf{x}^{(i)} - \mathbf{c}_{(k)}||\right) + b_j$$

with the set $\{\mathbf{c}_k\}_{k=1}^{K}$ being the set of centers, the transfer function Φ is

$$(\Phi_c)_{ij} = \phi\left(||\mathbf{x}^{(i)} - \mathbf{c}_{(j)}||\right)$$

and is a $P \times K$ matrix. Once a set of centers has been chosen, the problem is simply to find the least-squares solution to $\hat{\Phi_c}\hat{W} = Y$.The solution is $\hat{W} = \hat{\Phi}_c^+ Y$, where $\hat{\Phi}_c^+$ is the pseudoinverse of $\hat{\Phi}_c$. Since the centers have been chosen by the LBG algorithm, this setup suits our problem nicely.

Given a single data point, the equations of motion for a local cluster allow us to get the next point in the time series. By building a mapping from the low dimensional coordinate system back to the original dimension, we can iterate a point through the system according to the dynamics.

6. An Example

We will work through a broad example to see how these methods are applied to an actual set of chaotic data. Because the data set with which we are dealing is so large, the methods outlined thus far in this paper will be implemented in MATLAB.

6.1. **Introduction.** For this example we will work with the Lorenz equations. The Lorenz equations were discovered in 1963 by Edward Lorenz in his attempts to build a mathematical model of the weather. The equations themselves were derived from the modeling of Earth's atmosphere; the looping oscillators represent convection rolls between the upper and lower atmosphere.

The Lorenz equations are given by

$$x' = \sigma(y - x)$$

$$y' = x(\rho - z) - y$$

$$z' = xy - \beta z$$

where σ is called the Prandt number and ρ is called the Rayleigh number. All $\sigma, \rho, \beta > 0$, but usually $\sigma = 10$, $\beta = 8/3$ and ρ is varied, to give more interesting results.

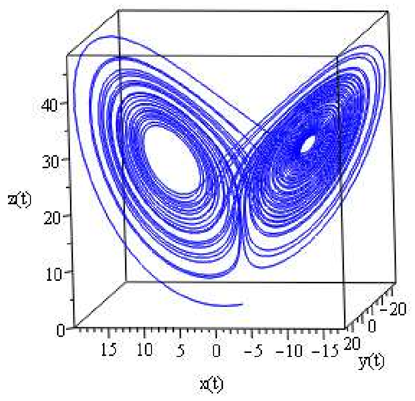

FIGURE 4. The Lorenz attractors with $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$.

Of course, as the ultimate goal is to uncover the dynamics governing the system, the preceding equations would normally remain unknown. As we are using a well-known example, however, we have the benefit of knowing the dynamics ahead of time so that we may verify the final results.

We begin with a set of data, 9000 points in 3 dimensions, generated by the Lorenz equations. Remember that for the purpose of the example, the dynamics remain unknown. Only the data is given.

6.2. **Clustering.** The first important task is the clustering itself. Recall that we will use the LBG algorithm to cluster the data. Refer to `lbgcluster.m` in the appendix. This function performs the clustering outlined by the LBG algorithm. It initializes the data centers by choosing them to be randomly selected data points. As discussed earlier, the function then updates the centers iteratively to minimize the distortion error.

When the distortion error stops decreasing, the function ends. It then builds a data structure `A` and stores each center, the data points in that center, and the distortion error for that center as elements in the data structure. The Matlab script allows us to set the desired number of clusters. The choice as to how many clusters to use is a trade off - using more clusters allows the local data sets to be more linear, but requires more computing time. In this example, we choose to initialize fifteen centers.

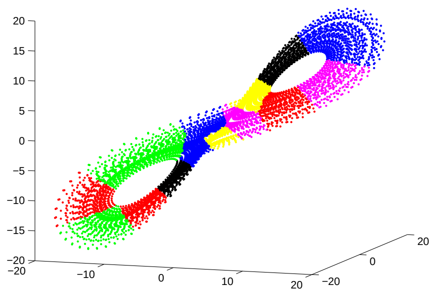

FIGURE 5. The results of the clustering algorithm on the Lorenz data, with each color representing a different cluster center

6.3. **Local SVD and Best Basis.** Refer to `localsvd.m` in the appendix. Recall the earlier discussion of the singular value decomposition and the Karhunen-Loève expansion. This function implements those two pieces of work to build the best basis for each of the local data clusters. For each cluster, `localsvd.m` generates the best basis by decomposing the cluster into its singular value decomposition and taking the right singular vectors. It then determines a cutoff for the embedding dimension. The embedding dimension is determined using the method of normalizing the singular values described earlier. In this example the cutoff is set to 0.95. The function adds each local basis and the embedding for each local basis as elements in the data structure `A`.

6.4. **Projecting To a Lower Dimensional Subspace.** By projecting each local data cluster to its lower dimensional basis, we find the lower dimensional coordinates of each local cluster. For the projection, refer to `projection.m` in the appendix.

6.5. **Iterating a Point Through the System Using RBFs.** Refer to `localmotion.m` in the appendix. Radial basis functions are used to build the local equations of motion. The Gaussian $\phi(r) = e^{-r^2/-\sigma^2}$ is used for the transfer function, and we let MATLAB approximate the best width, σ. After the algorithm has found the equations of motion, it uses the script in `sim.m` to project the point back to the original dimension and advance it one iteration. In this way, given an initial point we may move it through the system for any number of iterations so that we can predict its position at some point in the future.

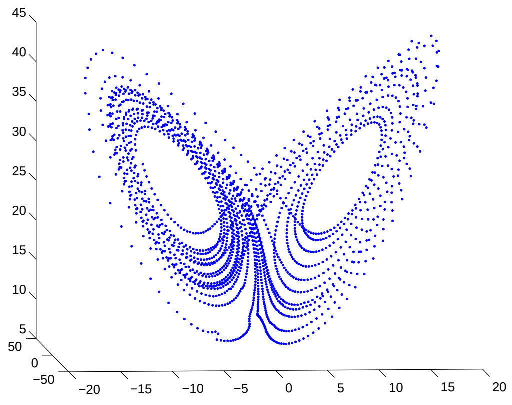

FIGURE 6. The plot of each predicted iteration. Comparing to the original plot of the Lorenz attractors, we see that we have indeed constructed local models of motion for the system.

In the case of the Lorenz data, we consider the point `[-10.5419 -4.9817 35.1046]`. After 2000 time steps, its position is `[12.0000 4.8074 38.0597]`. Because we have constructed the local equations of motion and the algorithms to iterate the point through the system, we can predict the position of the point at any future time step.

Conclusion

We have thus shown that for a discrete, chaotic system, local equations of motion may be built such that a model of the system can be constructed, and point iterated through it. The Linde-Buzo-Gray algorithm clusters the data into locally linear groups so that we can rebuild the dynamics of each. The singular value decomposition is an important tool in this process because it can be used to build the best lower dimensional basis for each cluster. By projecting the local cluster onto the basis, we obtain the lower dimensional coordinate systems for each cluster. Finally, radial basis functions may be used to rebuild the dynamics of each linear cluster and to iterate a given point through the system. The code and algorithms included here will hopefully provide incentive for further investigation of this process, as they may be used with any appropriate dataset.

Acknowledgments

I am very grateful to Professor Dough Hundley for his efforts in guiding me through this project which, without his knowledge and insight, would not have been possible. I would also like to thank Professor Pat Keef, Ian Coleman, and Kailey Bolles for their help in the editing process.

Appendix

1. lbgcluster.m.

```
function A=lbgcluster(X,C,I)


%FUNCTION A=LBGCLUSTER(C,I)
%  Take in a data set gX (GLOBAL) and centers in C, outputs the clusters in A(i).ldat, A(i).Center
%  Also computes the distortion errors in A(i).distort.  NOTE:  A(i).ldat is a vector of
%  indices pulled from the index set I.
%Stopping criterion:  The distortion errors stop decreasing.
```

```
[numcenters,cdim]=size(C);
[numpts,ndim]=size(X);
lengthI=length(I);

%% Error checks:
if cdim~=ndim
   error('Dimensions do not check in LBGCLUSTER, %d, %d\n',cdim,ndim);
end
if lengthI~=numpts
   error('Number of points do not match length of index:  %d %d\n',numpts,lengthI);
end

lbgflag=0;

%% Initialize the structure A with the current centers and data
for i=1:numcenters
   A(i).center=C(i,:);
end



%Initial Sort:
P=dist(X,C');        %Returns P as numpts x numcenters
[H,idx]=min(P,[],2); %Now H holds the distortion measures, and
                     % idx holds the cluster indices
sumdistort=0;

for i=1:numcenters
   idx1=find(idx==i);
   if isempty(idx1)
      fprintf('No data in cluster %d\n',i);
      fprintf('To go back to the loop, type >> return\n')
      keyboard     %The keyboard command is useful for debugging.
      error('No data in one cluster');
   end

   A(i).ldat=I(idx1);
   A(i).distort=mean(H(idx1));
   sumdistort=sumdistort+A(i).distort
   A(i).center=mean(X(idx1,:));
   C(i,:)=A(i).center;
end

old_distort=sumdistort;
%% Main loop here

while lbgflag==0
  %Sort the distances
   P=dist(X,C');  %Returns P as numpts x numcenters
   [H,idx]=min(P,[],2);
   %Now H holds the distortion measures, and idx holds the cluster indices
   sumdistort=0;
   for i=1:numcenters
      idx1=find(idx==i);
      if length(idx1)>0
        A(i).ldat=I(idx1);
        A(i).distort=mean(H(idx1));
        sumdistort=sumdistort+(1/length(idx1))*A(i).distort;
        A(i).center=mean(X(idx1,:));
        C(i,:)=A(i).center;
      else
         error('No data in cluster %d\n',i);
      end

   end
   % Stopping criteria:
   sumdistort
   if sumdistort>old_distort
```

```
      lbgflag=1;
   else
      old_distort=sumdistort;
   end


end
```

2. localsvd.m.

```
function A=localsvd(A,X,cutoff)
%Function A=localsvd(A)
%  Constructs the local axes for each of the data sets contained in the
%  structure A (that was produced by lbgcluster).
%  X is a tall matrix- number pts x dim

m=length(A);
for j=1:m
    % A(j) contains the structure
    %  A(j).center is 1 x dimension
    %  A(j).ldat is
    mm=length(A(j).ldat);
    B=X( A(j).ldat, :);  %Matrix B is mm x dim
    B2=B-repmat(A(j).center,mm,1);  %B2 is center subtracted

    [U,S,V]=svd(B2,'econ');

    % Determine a cutoff for the dimension of the local data using S, store
    % the result as A(j).dim

    hh=diag(S)./sum(diag(S));
    hh=cumsum(hh);

    i=1;
    while (hh(i)<cutoff)
        i=i+1;
    end
       A(j).dim=i;

    %Find the best basis for B2, store the result as A(j).basis (like 2 x 3
    %or 3 x 2) using the columns of V

    % The best basis vectors for the rowspace of B2 are the right singular
    % vectors for B2.

    % Thus the best basis is formed by the column vectors of V.
    A(j).basis=V(:,1:i);
end
```

3. projection.m.

```
function [A] = projection(A,X)
%Projects data onto basis, back onto data
%Inputs data structure A, data X
%Outputs new elements of A

hh={'r.','b.','g.','k.','m.','y.','r.','b.','g.','k.','m.','y.','r.','b.','g.'};
        %allows us to plot each projection as a different, assigned color

for j=1:length(A)
    mm=length(A(j).ldat);
    XX=X(A(j).ldat,:)-repmat(A(j).center,mm,1); %matrix of mean-subtracted data (jth cluster)
    dim=A(j).dim;

    A(j).projection=XX*A(j).basis(:,1:dim);
    A(j).projection2=A(j).projection*A(j).basis(:,1:dim)';

    if j==1
        hold on
    end
    XX=A(j).projection2+repmat(A(j).center,mm,1);
    plot3(XX(:,1),XX(:,2),XX(:,3),hh{j});
```

```
end
hold off
```

4. localmotion.m.

```
function A=local_motion(A,X)
%FUNCTION [C,b]=local_motion(A,X)
%This function uses batch learning to uncover local equations of motion of
%the system x'=Cx+b, where x' is approximated as x_(n+1)-x_(n)

%(using C to differentiate from the data structure A which we are also
%using)



for j=1:length(A)


    [m,n]=size(A(j).basis);
    mm=length(A(j).ldat);

    % Let a neural network advance you
    % from the projected point v not just to x(t) but to x(t+delta t).
    P=(X(A(j).ldat,:)-repmat(A(j).center,mm,1))*A(j).basis;
    T=X(A(j).ldat+1,:)-repmat(A(j).center,mm,1);

    %For the RBFs, P and T should be dim x numpts
    [net,tr]=newrb(P',T',0.0001);

    %******************************************************************
    %  If training was bad, increase the dimension by 1 and train again.
    %
    if tr.perf(end)>0.0001
        fprintf('Retraining this cluster\n')
      B=X( A(j).ldat, :);  %Matrix B is mm x dim
      [m2,dim2]=size(B);
      B=B-repmat(A(j).center,m2,1);  %B is center subtracted
      [U,S,V]=svd(B,'econ');
      A(j).dim=A(j).dim+1;
      A(j).basis=V(:,1:A(j).dim);
      P=(X(A(j).ldat,:)-repmat(A(j).center,mm,1))*A(j).basis;
      T=X(A(j).ldat+1,:)-repmat(A(j).center,mm,1);
      net=newrb(P',T',0.0001);
    end
    %
    %***********End of retraining**************************************

    A(j).rbf=net;

end
```

5. sim.m.

```
%% Simulator for the data

load Model;  % contains the model A and starting points in x (2 rows)



numpts=2000;  %Number of points to generate in the model
Data=zeros(numpts,3);  %Store the trajectory here...




for j=1:numpts

    %  Find the cluster
    idx=FindCluster(A,x);
    % lagdim=length(A(idx).center); (not going to use - lagdim is always 3)
    lagdim=A(idx).dim;

    xn=x(2,:);
    dxn=diff(x);
    % Project to local coordinates
    Px=(xn-A(idx).center)*A(idx).basis;


    % Advance to the next point using the RBF
    xnew=sim(A(idx).rbf,Px');
    xnew=xnew'+A(idx).center;
    % Re-set for next iteration
    x(1,:)=xn;
    x(2,:)=xnew;


    Data(j,:)=xnew;
end


plot3(Data(:,1),Data(:,2),Data(:,3),'.')
```

References

[1] Alligood, Kathleen T., Tim Sauer, and James A. Yorke. *Chaos: An Introduction to Dynamical Systems*. New York: Springer, 2000. Print.

[2] Hundley, Doug R. *An Introduction to Empierical Modeling*. 2002

[3] Kaplan, Daniel, and Leon Glass. *Understanding Nonlinear Dynamics*. New York: Springer, 2003. Print.

[4] Lay, David C. *Linear Algebra and Its Applications*. Reading, MA: Addison-Wesley, 2000. Print.

[5] *The Moore-Penrose Pseudoinverse*. Caltech Robotics. Web. http://robotics.caltech.edu/jwb/courses/ME115/handouts/pseudo.pdf.