2.2.1 Representations

Chaotic dynamics does not occur unless f is nonlinear, so to approximate chaotic dynamics we must make nonlinear models. There are an infinite number of ways to represent nonlinear functions, and finding good nonlinear approximations is a difficult problem. As soon as we pick a representation, we make an *ad hoc* assumption, which may or may not be a good one. With prior information we may be guided to a better representation, but in the absence of any theoretical understanding we are forced to make arbitrary choices.

At this point finding a good representation is largely a matter of trial and error. We make a guess and then test to discover whether it produces a good model. Convenience plays an important role, since fitting parameters is usually time consuming unless it can be reduced to a linear problem.[6] Certain representations perform better than others across a wide class of problems, but although there are a few rigorous results [51], this is mainly a matter of numerical lore.

We now discuss a few of the representations that we are currently exploring.

Polynomials are a good representation because their parameters can be linearly fit to minimize least squares deviations, and because they arise naturally in Taylor expansions. The use of polynomials for forecasting was suggested by Wiener [76], who proposed them for moving average models, and Gabor [33,34], who proposed them for autoregressive models. Polynomials have also recently been used for fitting deterministic equations of motion by several authors [23,16,18]. The most general form for an m^{th} degree d-dimensional polynomial is

$$A_m(x_1,\ldots,\ldots x_d) = \sum_{i_1=0,\ldots,i_d=0} a_{i_1\ldots i_d} {x_1}^{i_1}\cdots {x_d}^{i_d}. \tag{9}$$

where $\sum_{j=1}^{d} i_j \leq m$. The number of parameters $a_{i_1\ldots i_d}$ is $\frac{(d+m)!}{d!m!} \approx d^m$. Fitting this many parameters rapidly becomes impractical when m and d are large. Polynomials have the disadvantage that they do not extrapolate well beyond their domain of validity since $\|A\| \to \infty$ as $\|x\| \to \infty$.

Rational approximation by the ratio of two polynomials provides an appealing alternative [62]. Rational approximations extrapolate better than polynomials, particularly when the numerator and denominator are of the same degree, since they remain bounded as $\|x\| \to \infty$. Furthermore, like polynomials, fitting parameters by least squares is linear. The work of Bayly *et al.* [3] suggests that rational approximation does a good job of fitting some chaotic attractors, and preliminary investigations of Lee and Lee [50] suggest that they may also be a good choice for forecasting problems.

[6] Even when the least squares problem is linear, the solution is not well behaved if the data are singular. This can be improved through singular value decomposition [62], which is the method we use throughout this paper.

Radial basis functions [61] have been recently suggested by Casdagli [12] for nonlinear modeling. They are a global interpolation scheme with good localization properties. In their simplest form, they only depend on the distance between points.[7]

$$R(x) = \sum_{t'=1}^{N} \lambda_{t'} \phi(\|x - x_{t'}\|) \tag{10}$$

where $\|\ \|$ is the Euclidean norm, and t' is a label attached to the points in the time series. The coefficients λ_j are chosen to satisfy the interpolation conditions $x(t'+T) = R(x(t'))$. If ϕ is taken to be $\phi(r) = (r^2 + c^2)^{-\beta}$, where $\beta > -1$ and $\beta \neq 0$, (the form we use in this paper) the linear system given by Equation (10) has a unique solution as long as the $x(t')$ are distinct. A special case of radial basis functions are thin plate splines.

Neural networks provide another alternative. Although on the surface neural networks seems quite different, on closer examination it becomes clear that this difference is superficial. Neural networks can be viewed as just another class of functional representations. This was explicitly demonstrated by Lapedes and Farber [49], who recently applied a standard feed-forward neural net with two hidden layers [65] to some of the forecasting problems that we studied in reference [23]. Their neural net can be written as

$$\hat{x}(t,T) = \sum_k w_k z_k - a_o \tag{11}$$

$$z_k = \tanh\left(\sum_j w_j y_j - a_k\right), \tag{12}$$

$$y_j = \tanh\left(\sum_i w_i x_i - a_j\right), \tag{13}$$

$$x_i = x(t - i\tau), \tag{14}$$

where y_j and z_k are the values of the neurons in the two hidden layers, and x_i are the input neurons. Thus, the neurons are simply coordinates in the state space. The label "neural network" implies a particular functional representation, involving a sum over coordinates, followed by a sigmoidal function such as tanh. While this form is motivated by biological considerations, for problems in artificial intelligence or forecasting there is no reason to adhere to it unless it is better than others.

Lapedes and Farber determine the parameters a and w using back-propagation, which is essentially a nonlinear least squares algorithm. The tanh representation has several nice properties [49], but it has the disadvantage that parameters cannot be fit by solving a linear problem, and the the back propagation algorithm is time consuming. As a result, fitting parameters takes several orders of magnitude more computer

[7] Often a polynomial is added as well, but we do not use this.

time.

2.2.2 Local approximation

To make a good approximation for a function f, a representation must be able to conform to its variations. If f is well-behaved, any complete representation will provide a good approximation, as long as it has enough free parameters, and enough data to stably solve them. However, if f is complicated, there is no guarantee that any given representation will approximate f efficiently.

The dependence on representation can be reduced by *local approximation*. The basic idea is to break up the domain of f into local neighborhoods and fit different parameters in each neighborhood. When f is smooth the neighborhoods can be small enough so that f does not vary sharply in any of them, making the constraints of a particular representation less important.

Local approximation usually produces better fits for a given number of data points than global approximation, particularly for large data sets. It seems that most global representations reach a point of diminishing returns where adding more parameters or more data only gives a marginal improvement in accuracy. Higher order terms may oscillate, and actually cause the behavior to get worse. Past a certain point, adding more local neighborhoods is usually more efficient than adding more parameters and going to higher order. Local approximation makes it possible to use a given functional representation efficiently. The key is to choose the local neighborhood size properly, so that each neighborhood has just enough points to make the local parameter fits stable, so that adding more points would not make significant improvements.

We will define an M point *neighborhood* of x as a collection of M points $\{y_i\}$ which are in some sense close to x. Although this is perhaps a slightly unusual way to use the word "neighborhood", it is convenient in the discussion that follows. Note that although we use the language of dynamics in the following discussion, most of our remarks apply to any situation in which we wish to make a map from one set of values to another.

Although local approximation is more trouble to implement than global approximation, it is often well worth the effort, particularly for large data sets. To use local approximation in the vicinity of a point x, there are three basic steps:

1. Pick a local representation.
2. Assign neighborhoods.
3. Find a local *chart* that maps the points in each neighborhood into their future values. To make a prediction we evaluate the chart at x.

The basic idea is illustrated in Figure (2).

A simple way to assign neighborhoods is to partition the domain into disjoint sets. For example, we could use a rectangular grid. This approach is convenient, but it has

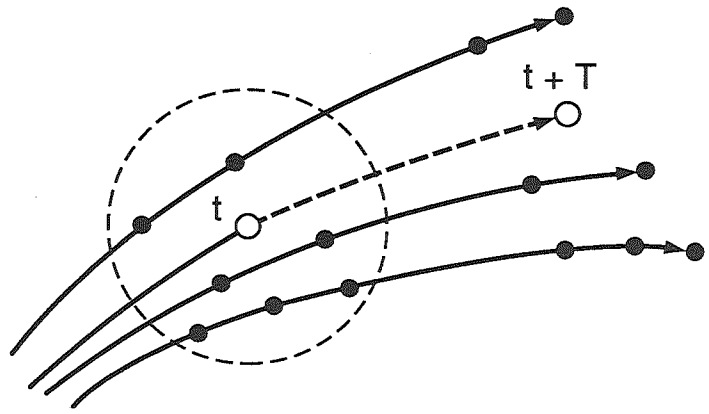

Figure 2: *Local approximation.* The current state $x(t)$ and its unknown future value $x(t+T)$ are represented by open circles. The black dots inside the dashed circle are the neighbors of $x(t)$. To make a prediction we fit a local chart with the neighbors in its domain, and the states they evolve into a time T later in its range. To make a prediction we evaluate the chart at $x(t)$.

the disadvantage that there is no overlap between the neighborhoods, and therefore no continuity between charts. A point near the boundary of its neighborhood may be poorly approximated. This is particularly true for representations such as polynomials that do a poor job of extrapolating outside their domain of validity.

One way to cope with this problem is to enforce matching conditions between adjacent neighborhoods. For many interpolation schemes, for example, this is an essential element in achieving accuracy. Unfortunately, this becomes a difficult problem for data in more than two dimensions, which is precisely the situation that we are most interested in here.

An alternative that is more accurate than disjoint partitions and more convenient than enforcing matching conditions is to overlap the neighborhoods, so that each chart is constructed from a good set of neighbors. Let $\{y_i\}$ be the n points of the neighborhood. We want to choose $\{y_i\}$ so that the predictions are as good as possible. A simple criterion is nearness: For a given metric $\| \ \|$ and a given n we will say that $\{y_i\}$ is the *nearest neighborhood* of x if it minimizes $\sum_i \|x - y_i\|$. This criterion is not optimal, as can be seen by considering linear interpolation in two dimensions; if the triangle defined by the three nearest neighbors does not enclose x, then the interpolation may be poor. In practice we find that choosing good neighborhoods makes a big difference in the quality of our predictions.

Once we have chosen neighborhoods, the next step is to fit charts to them. To do this we must pick a representation. While we anticipate that dividing the domain

into neighborhoods reduces the dependence on representation, it is nonetheless the case that some representations are better than others. We are again forced to make an *ad hoc* decision and evaluate the results empirically based on performance.

To measure the accuracy of forecasts we use the normalized root-mean square error, or the *prediction error*

$$\bar{E}^2 = \langle E^2 \rangle = \frac{\langle (\hat{x}(t,T) - x(t+T))^2 \rangle}{\langle (x(t) - \langle x(t) \rangle)^2 \rangle} \tag{15}$$

Thus $\bar{E} = 0$ for perfect forecasts, and $\bar{E} = 1$ for forecasts made using $\hat{x}(t,T) = \langle x(t) \rangle$. Usually we compute this by averaging over time. In some of the scaling derivations it is also convenient to use the average absolute error instead of the root-mean-square error.

Local approximation schemes can be classified according to the order of the derivatives that the errors depend on. For example, suppose our charts are polynomials of degree m. Suppose we want to approximate a function f that is not itself a polynomial of degree m. In the ideal case the error depends on $f^{(m+1)}(x)$, the $m+1$ derivative of f. This implies that the errors are proportional to ϵ^{m+1}, where ϵ is the spacing between data points. The average spacing between N points uniformly distributed over a D dimensional space is $\epsilon \approx N^{-\frac{1}{D}}$. Calling q the *order of approximation*, we find[8]

$$\bar{E} \sim N^{-\frac{q}{D}}, \tag{16}$$

where in this case $q = m + 1$.

Achieving the ideal case where $q = m + 1$ is difficult for large q, since in general fitting a polynomial of degree m does not produce a fit that is accurate to order $m+1$. For example, suppose the number of data points is equal to the number of free parameters. The approximation may go through each point precisely, but in between it may oscillate wildly, producing an extremely inaccurate approximation. Even when we use more neighbors this may limit the accuracy.

To avoid this confusion we will use Equation (16) to *define* the order of local approximation, taking the limit as $N \to \infty$, and letting D be the information dimension of the underlying measure of the data points.

$$q = \lim_{N \to \infty} \frac{D |\log \bar{E}|}{\log N} \tag{17}$$

In general q may depend on D, f, the way in which we choose the neighborhoods, and other factors.[9]

Moving to representations of higher degree involves a tradeoff – higher degree representations potentially promise more accuracy, but also require larger neighborhoods. A larger neighborhood usually implies that the complexity of f increases.

[8]We will use the symbol "$\sim$" to mean "scales as", *i.e.* $z \sim y(x)$ implies $z = Cy(x)$, where C includes all dependencies on variables other than x.

[9]Note that the order of approximation as we have defined it here is one larger than the definition we gave in references [23,27]. We have changed our definition to correspond to common usage.

Finding the best compromise between these two effects is a central problem in local approximation.

A trivial example of local approximation is *first order*, or *nearest neighbor* approximation. This amounts to simply looking through the data set for the nearest neighbor, and predicting that the current state will do what the neighbor did a time T later. We approximate $x(t+T)$ by $\hat{x}(t,T) = x(t'+T)$, where $x(t')$ is the nearest neighbor of $x(t)$. For example, to predict tomorrow's weather we would search the historical record and find the weather pattern most similar to that of today, and predict that tomorrow's weather pattern will be the same as the neighboring pattern one day later.[10] First order approximation can sometimes be improved by finding more neighbors and averaging their predictions, for example, by weighting according to distance from the current state.

An approach that is usually superior is *local linear* or *second order* approximation. For the neighborhood $\{x(t')\}$ we simply fit a linear polynomial to the pairs $(x(t'), x(t'+T))$. When the number of nearest neighbors $M = d + 1$ and the simplex formed by the neighbors encloses $x(t)$ this is equivalent to linear interpolation. If the data is noisy, the chart may be more stable when the number of neighbors is greater than the minimum value. Again, this procedure can be improved somewhat by weighting the contributions of the neighboring points according to their distance from the current state. Linear approximation has the nice property that the number of free parameters and consequently the neighborhood size grows slowly with the embedding dimension.

Since the accuracy increases with the order of approximation, it is obviously desirable to make the order of approximation as large as possible. Any nonlinear representation is a candidate for higher order approximation. The criteria for a good local representation are somewhat different from those for a good global representation. On one hand, getting a good fit within a local neighborhood is easier, because the variations are less extreme. Wild variations or discontinuities can be accommodated by assigning neighborhoods properly. On the other hand, a local neighborhood necessarily has less data in it, so the representation must make efficient use of data. The order of approximation actually achieved may depend on other factors, such as the choice of neighborhoods, the dimension, or peculiarities of the data set. For reasonably low dimensional problems we have found that we usually achieve third order approximation, for example using quadratic polynomials. In low dimensions (two or less) it is sometimes possible to do better. [11] In higher dimensions we often find that we cannot do any better than second order approximation with our current techniques.

Another interesting alternative is to compute charts for each data point, and then view the parameters of these charts as new states. By fitting charts to them we can

[10]This was attempted by E.N. Lorenz, who examined roughly 4000 weather maps [52]. The results were not very successful because it was difficult to find good nearest neighbors, apparently because of the high dimensionality of weather.

[11]Casdagli has independently reached the same conclusions [12]. In two dimensions he apparently achieves *sixth* order approximation in some cases using global radial basis functions, but this does not seem to carry over in more dimensions.

make *metacharts*. For example, we could use local linear approximation both for fitting charts to past states and for interpolating between them to find a good chart for the current state. This process can obviously be continued *ad infinitum*, at least in principle. Metacharts may be smoother than charts, resulting in a more compact or more accurate description. We intend to investigate this further.[12]

Some people find local approximation objectionable because it does not result in a "closed form" model. We do not consider this a problem. The most informative diagnostic information comes from knowing statistical properties such as the Lyapunov exponents or fractal dimension. While the coefficients of a global polynomial expansion might give some extra insight, if this results in a significantly less accurate model, we may get estimates of statistical properties that are incorrect in even qualitative terms. We feel that accuracy should be the most important criterion for model selection.

2.2.3 Trajectory segmenting

For continuous time series the strategy for finding a good neighborhood is not quite the same as it is for discrete maps. In particular, the sampling time Δt becomes an important parameter, and if it is small this must be taken into account. Local neighborhoods chop a continuous trajectory into segments. We say that the sampled values of a time series lie on the same *trajectory segment* if they can be sorted so that they are contiguous in time. For example, the local neighborhood in Figure (2) contains four trajectory segments. If the sampling rate Δt is small, finding a fixed number of neighboring data points may be very different than finding a given number of trajectory segments. For example, if Δt is really small, all the neighbors of a given point will lie on the same trajectory segment. As a result they are nearly co-linear, which results in a highly singular chart. We can attempt to compensate by increasing the number of nearest neighbors, but due to nonuniformities some regions may behave differently than others. A more reliable approach is to choose neighborhoods so that they include a given number of trajectory segments, rather than a given number of points. This gives much better control over the quality of the fits.

2.2.4 Nonstationarity

Most of the results in this paper assume that we are dealing with stationary data. The assumption that the trajectory is on an attractor guarantees this, as long as the parameters are held constant. Variations of parameters, however, can result in nonstationary behavior. The most straightforward way to deal with this is to extend the state space to include time. It can be included in the metric, so that the search for nearest neighbors favors recent data. With time as a coordinate, charts can be constructed that take into account trends and other time dependent effects. Similarly,

[12] Y.C. Lee, who has independently suggested this approach, makes the intriguing observation that the local approximants can be used to "re-embed" the data. If f has low complexity, this might reduce the size of the data base.

a periodic function of time can be included to cope with seasonality or other periodic behavior.

2.2.5 Discontinuities

The problems caused by discontinuities can be minimized by choosing neighborhoods properly, so that their boundaries follow and do not cross discontinuities. The worst situation occurs when the points in a neighborhood are on different sides of a discontinuity – a smooth chart will inevitably produce a poor approximation.

In order to detect the presence of a discontinuity it is necessary to examine the range as well as the domain of the transformation. Two points on opposite sides of a discontinuity may be nearby in the domain. However, by extending the metric to include the range as well as the domain, singularities become evident, since points on opposite sides of a discontinuity are far apart. By definition the future value of the current state is unknown, so we cannot use it for neighborhood relations that involve both past and future values, but this is not true for the rest of the database. By extending the metric to include the range values, we can make sure that neighborhoods are chosen so that they do not cross the discontinuity. This guarantees that each chart is fit with a consistent set of points. As long as we place the current state in the correct neighborhood, on the proper side of any discontinuities, the predictions will be good. At least we win some of the time, rather than losing all the time. This procedure also makes it possible to deal with multiple valued behavior, since points on the same branch can be distinguished from those on different branches.

2.2.6 Implementing local approximation on computers

Finding neighbors in a multidimensional data set is time consuming when done by brute force. The most straightforward approach is to compute the distance to each point, which takes roughly N steps for N points. This can be reduced to roughly $\log N$ steps by organizing the data with a decision tree, such as the *k-d tree* [6,5]. The basic principle is illustrated in Figure (3).

The data set is partitioned one coordinate at a time. Any criterion can be used for the partitions; for example, for the results reported here we find the coordinate with the largest range, and partition it at its median value. The values corresponding to each partition are stored in the tree as *keys*. A given query point can be located quickly by comparing its coordinates to the keys. If we want to find the nearest neighbors this makes it possible to eliminate many points from consideration without actually computing their distance. The k-d tree has the nice property that it flexibly partitions only the parts of the space that actually contain data, adding partitions only where they are needed. Providing the k-d tree is used to find nearest neighbors, the principal speed limitation for local approximation comes from fitting the parameters of the charts.

The best approach for a given application depends on whether it is constrained by speed, computer memory, or the availability of data. At one extreme, the most

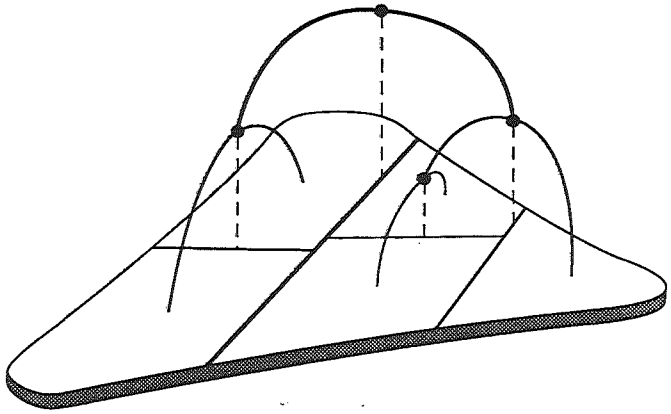

Figure 3: *The k-d tree.* The data is partitioned one coordinate at a time. Each partition is assigned a node in the tree, as indicated by the dashed lines. The partitions can be made on any coordinate, and the tree can have different depth in different regions. The result is a flexible partition of the data space, with finer partitions where there is more data. Use of the k-d tree speeds up the search for nearest neighbors, decreasing the number of steps from N to $\log N$.

accurate predictions come from finding optimal neighborhoods. This is much easier to do once we know the state, which means that we must find neighbors and fit charts as we go. At the other extreme, the fastest approach is to use fixed disjoint neighborhoods, compute charts in advance, and store them in the tree. This approach is extremely fast, requiring only the order of $\log N$ steps to locate the proper chart, plus the time needed to evaluate the chart. Even though it requires more computer memory, for real-time applications this may be the method of choice. Our numerical experiments indicate, however, that the sacrifice in accuracy can be considerable, especially in higher dimensions. Even in one dimension the difference between these two approaches for quadratic polynomial approximation can be almost an order of magnitude, as shown in Figure (4). A compromise between these two extremes is to compute charts for all the points in the data base, find the nearest neighbor of the current state, and borrow its chart to make a forecast. This procedure is fast, but consumes even more computer memory than fixed disjoint partitions.

If an application is limited by data, it may be important to update the tree after each prediction, to make use of each new data point in the next prediction. At the opposite extreme, if an unlimited supply of data is available and computer memory is at a premium, it may be desirable to find an optimal data set by selectively inserting new points where the approximation is bad and removing old points where it is already good. Regions where f changes rapidly should be covered densely, while regions where

f is relatively smooth only need to be covered sparsely. A simple criterion is to insert states that result in bad predictions, and compensate by removing states that result in good predictions. Aspects of this are discussed by Omohundro [58].

In our numerical work here, for convenience we simply use the first part of the data set to build a data base, and use the second part to make predictions. Of course, for a nonstationary process, it would be important to continually update the data base instead.

Although local approximation is certainly more trouble than global approximation, with the k-d tree it is quite fast. The k-d tree is easy to implement, as long as one avoids caveman computer languages such as FORTRAN. Local approximation does not require massive amounts of computer power. The computations for this paper were made with a SUN-3 microcomputer.

2.2.7 An historical note

At the time we wrote the paper of reference [23] we were unaware of other work using local approximation for forecasting. Now we are more informed, and a few historical comments are in order. The use of nearest neighbor (first order) approximation goes at least as far back as the work of Lorenz [52] in 1969. Linear approximation with fixed disjoint partitions was introduced by Tong and Lim [75] in 1980. Priestley gave a more general approach in the same year, and also suggested the possibility that higher order approximation might be useful. In the dynamical systems literature in 1981 local linear approximation was independently proposed as a means of computing dimension by Froehling et al. [31]. It was also proposed by Eckmann and Ruelle [21,20] and Sano and Sawada [66] to compute Lyapunov exponents, except that they omit the constant term.

More recently, several authors have independently suggested various forms of local approximation [35,18,23,12]. Linear interpolation using nearest neighbors was suggested by Peter Grassberger [35], who implemented it for the Henón map. Crutchfield and MacNamara [18] also suggested local linear approximation with disjoint neighborhoods, which they refer to as an "atlas dynamical system". In reference [23] we suggested local approximation using nearest neighbors, demonstrated its effectiveness on several experimental and numerical time series, and proposed the scaling laws that make the advantages of higher order approximation clear. Stimulated by our work, Lapedes and Farber [49] showed that neural nets could also be used for forecasting. Casdagli suggested the use of radial basis functions [12].

The use of global polynomials goes back to the work of Gabor [33] and Wiener [76]. It was also recently suggested independently by Cremers and Hübler [16], Crutchfield and McNamara [18], us [23], and Lapedes and Farber [49].

2.3 Comparison to statistically motivated methods

In this section we compare the methods discussed here, which are motivated by deterministic dynamics, to previous nonlinear forecasting methods that are based on