On-line environment anticipation using multivariate Legendre series

by

Aaron Lee, B.Sc.

A Thesis

in

Electrical Engineering

Submitted to the Graduate Faculty

of Texas Tech University in

Partial Fulfillment of

the Requirements for the Degree of

Master of Science

Approved

Richard Gale

Chair of the committee

Mohan Sridharan

Peggy Gordon Miller

Dean of the Graduate School

December, 2011

ACKNOWLEDGMENTS

I would like to thank my advisor and committee Chair Dr. Richard Gale for recognizing my abilities and subsequently making it advantageous for me to remain in graduate school at Texas Tech instead of going elsewhere. I'm grateful for the amount of flexibility that he allowed me in being able to choose my own particular path. I would also like thank Dr. Mohan Sridharan for serving on my committee, and for enriching Texas Tech's computer science department by teaching a robotics AI course that focused on creativity and results. Much thanks to my mother who told me I was smart so often that I eventually believed it, and to my father who taught me how to say "Hello World" in C.

TABLE OF CONTENTS

ABSTRACT

In this thesis we use an orthogonal series expansion to do an on-line approximation of a system function. We first discuss usefulness of this technique along with a number of its potential applications. The types of systems this method can be used with and their properties are then discussed. Then, we go through the basics of multivariate orthogonal series expansions, along with a brief explanation of Legendre polynomials. This is followed by a discussion on what the role of the mimic system is, and what types of mathematical techniques are related to this one.

We explain why the Legendre polynomials are the choice of the orthogonal basis, and the special considerations taken when calculating them. Afterwards, the issues of memory constraints are used to explain a number of design decisions related to representation of the orthogonal series. This discussion includes how the issue of there being a finite number of orthogonal functions was addressed, and how the difficulty in acquiring state relationships and information was addressed. The method used in adapting the mimic to the reference system is then explained.

The creation of randomly generated reference systems is looked into, as well as what they are used for. A series of tests were conducted on the mimic system to test its abilities, and the results of these tests are given and explained. A number of possible improvements and extensions to this method are possible, and some of these are given.

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation

We endeavor to see the future so that we so that we may change it, and vice versa. At the time of this writing, the global economy is in the midst of a major crisis. So if we are looking for an example of motivation for why we would like to predict the future, this provides a good one. Currently computer systems are already employed calculating results for macroeconomic models. People who trust these models tend to believe that the the most influential factors in the dynamical systems have been identified and included. However, types of systems such as the global economy are extremely complex, and therefore most corresponding models are extremely unreliable. This is why a typical news report will read: "Housing market performs worse than expected" instead of: "Housing market performs about as well as expected". If the most well trusted models had actually been able to predict the economic downturn of 2008, it may have been easily prevented. Like most models, macroeconomic models rely heavily on human derived assumptions. A method which is able to identify important factors humans look over would be a much needed improvement. The method presented here would be able to automatically discover trends and missed dependencies.

On a related topic, trading stocks and currencies have been analyzed by computers as well. There is a great deal of money to be made here with very little human effort, so this has garnered much attention. Multilayer neural networks are a popular tool used for this type of task. The problem with using neural networks is the difficulty in predicting what the structure of the network should be. Also, neural networks as function approximators are usually somewhat inefficient due to their inefficient use of hardware multipliers. What might work better is the method

presented here, which deals with nonlinearity easily, does not require a network structure to be developed at all, and would efficiently use computer hardware.

Control systems are another area where future prediction is helpful. Let's say instance that we have created a complex robotic arm, but we would like to avoid having to take the effort in creating a physics model that may not even be as accurate as we would like. If we could have a model that learned the cause and effect automatically by observing the arm move around its entire range of motion, it would save a great deal of human effort. This model would do just that, and would be continually improved over the lifetime of the robot.

Usually the military has its products built for ruggedness and reliability. Moving parts in tanks, jets, ships, etc are susceptible to damage due to the nature of the profession. Because of this, a good control system should be able to automatically adapt to any changes in the system. For example, a fighter jet is in the air and takes damage to wing flap causing it to behave according to a model different than what's in the jet's computer. A better, more adaptable control system for this could save the mission, and the pilot.

Artificial intelligence is another area where this sort of trend analysis is needed. There has been much work in human-robot interaction over the years, and one area which currently has gathered a lot of attention is human assisted learning. An example of this would be a human giving verbal feedback to the robot about its performance. For this application, a system could be developed to predict what sort of emotion a human is feeling when he/she speaks. The system presented here could observe a simplified frequency spectrum of the speaker over time, and predict if the person is happy with the robot, or angry.

Because the system which will be performing these sorts of tasks is attempting to reproduce the output of another system one step ahead, we will call this the mimic system. The system that the mimic system is attempting to mimic will be called the reference system. The mimic system will not necessarily accomplish anything by

itself, and often would be used in conjunction with an agent that tries to perform some task.

1.2 Discrete dynamical systems

Because we are using digital computers, we have little choice but to sample the reference system. Therefore we must treat all systems as discrete. The sorts of systems that will be dealt with are not limited to the so called discrete-event dynamic systems. The number of possible states in these systems is finite (Hrúz and Zhou 2007). For the systems we deal with here, the number of possible states is infinite. In simplest terms, a discrete dynamical system can be thought of as a set of variables describing the system, and a function that describes how these variables change with time. More precisely, it is a mapping:

$$M: \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{1.1}$$

When considering the system to have an n-dimensional state vector s, it gives the relation

$$s_{t+1} = M_t(s_t) \tag{1.2}$$

where t indicates the time step.

A system like this does not take into account any inputs or outputs to the system, so taking this into consideration the mapping would be

$$M: \mathbb{R}^{n+i} \rightarrow \mathbb{R}^{n+o} \tag{1.3}$$

where i is the number of inputs, and o the number of outputs.

The subscript on M indicates that the mapping is possibly time dependent. That is, that the way that the state vector is transformed may change at any given time step. This type of system is called *non-autonomous*, where the alternative being *autonomous* does not provide for any changes in the mapping. For the purposes of this paper we will only consider the autonomous system for the reason that for any non-autonomous system, there is an equivalent autonomous system. A non-autonomous system with an n-dimensional state vector can be thought of as a set of autonomous systems with n-dimensional state vectors, with the results of the mappings from each of the autonomous systems being multiplexed onto the state vector. However, if a non-autonomous system is linear, its autonomous equivalent is likely to be non-linear. This is not a problem, because the focus of this research is on non-linear systems. A system also may have a number of control variables that determine its behavior. However, this case will not be considered, as control variables are equivalent to additional inputs.

If the system is to be "observed" in any sort of way, there must be an output from the system which is some function of the state variables. There technically need not be any sort of output from the system for it to exist, but then again any such system does not survive Occam's razor. In practical observation of systems, there is much more information about the state variables and the mapping than is provided by whatever outputs there are. In addition, the system may also have a set of inputs, which a subset of the state variables is also a function of. Of course, when talking about inputs and outputs, we do so in a context of looking at the system from the viewpoint of another system. The choice of differentiating between two interacting systems or considering it just one system is either arbitrary, or dependant on what we wish to accomplish by looking at the system(s) in the first place. Really, one can consider the entire universe to be one system, or consider it to be a very large number of maximally simple subsystems. When we wish to consider a system to have an input vector, we can write the recurrence relation as:

$$s_{t+1} = M(x_t, s_t) \tag{1.4}$$

where x is the input vector. By using recursion, we can see that this is equivalent to

$$s_{t+1} = G(x_t, x_{t-1}, x_{t-2} \dots x_{t-n}, s_{t-n}) \tag{1.5}$$

where G is some other mapping. The discrete dynamical system is a recursive process as shown in (1.4). At each transformation of the state vector, there may be a loss of information about the original state. For example, a one dimensional state vector being any real number loses its sign information if the state transition involves the squaring of the number. New information can be added to the system if it has inputs (that aren't just functions solely of the system anyways), but original state information is still at risk of being lost over time. In general, complete information about s is not directly obtainable, so the observable output y of the system is not s, but a function of s. This is why it is unlikely that a mimic system will ever be able to predict the outputs of the system at full accuracy. For any given input to the system there will be many possible outputs. Even with more knowledge than just the inputs, such as previous inputs or a subset of set variables, there will still likely be more than one possible output. This means we are treating the outputs of the system as probability density functions, of which we are finding their expected values. So in reality the system may very well be completely deterministic, but in practice we will be thinking of it as non-deterministic. When trying to create a model for any system, one must make a large number of simplifications in order to reduce the complexity of the model. Usually, a system can never be fully understood, and even if a full understanding was possible such a model taking to account all that information would be impractically large. Therefore, scientists and engineers tend to understand a small but important subset of the state variables and their relationships, and label everything else as noise. When we say a system has noise, what we are really saying is that the system has a number of "random" inputs. Of course, most things we call random are actually accepted to be completely deterministic, and according to Everett (1973) there is really is no such

thing as random at all. However, it is still pragmatic to label many things as random regardless of whether or not they are truly random or just random*ish*.

1.3 Fourier series of orthogonal functions

As previously stated, a system is simply a recursive function. In order to develop a mimic system, we must choose between one of the many ways of representing the recursive function. The one chosen here is an orthogonal series expansion. Each mimic output function is therefore expressed as a linear combination of multivariate orthogonal functions. The set of functions that comprise the series is called the basis. This method is similar to the familiar Fourier series of trigonometric functions, except in this case the functions in the series will be neither trigonometric nor one dimensional. However, the basic principle is the same. In this method, the multivariate orthogonal functions comprising the series are constructed by multiplying one-dimensional orthogonal functions together. Because of this we can take a well known one dimensional basis and use it to form a multivariate basis of however many dimensions desired. This is the method used by the mimic system.

Any piecewise continuous function f on the interval $(a,b)^n$ can be written as an infinite series of weighted orthonormal functions if the basis spans $L^2(a,b)^n$, that is if the orthonormal system $\{f_n\}$ is *complete* on $(a,b)^n$ (Walter, 1994). This is a pointwise convergence for piecewise continuous functions.

$$f = \sum_{n=0}^{\infty} c_n f_n \tag{1.6}$$

The weights $\{c_n\}$ are a Cauchy sequence. A Cauchy sequence is any sequence where you can pick any positive real number $\mathcal{E}$, and be guaranteed that there exists a certain point in the sequence after which any two numbers will be no more than $\mathcal{E}$ in distance from each other (Eidelman, 2004). Loosely speaking, in a Cauchy sequence the distance between numbers eventually goes to zero. Two functions f, g are defined as

orthogonal if $\langle f, g\rangle = 0$, and f is said to be orthonormal if $\langle f, f\rangle = 1$. The inner product of two functions is given by (1.7), and the more general formula is given in (1.8).

$$\langle f, g\rangle := \int_{a_0}^{b_0} \dots \int_{a_n}^{b_n} f(x_0, x_1 \dots x_n) g(x_0, x_1 \dots x_n)\, dx_0, dx_1 \dots dx_n \tag{1.7}$$

$$\langle \{a_0, a_1, a_2, \dots a_n\}, \{b_0, b_1, b_2, \dots b_n\}\rangle := \sum_{k=0}^{N} a_k b_k \tag{1.8}$$

An orthonormal system is *complete* if no non trivial function exists that is orthonormal to all members of $\{f_n\}$. That is if,

$$\nexists f \;\; \forall \{f_n\}\, \langle f, f_n\rangle = 0 \tag{1.8}$$

A test that is used to determine the completeness of an orthonormal system is to check to see if a set of functions $\{h_m\}$ that span the entire L^2 space satisfies Parseval's equality:

$$\|h_m\|^2 = \sum_{n=0}^{\infty} |c_n|^2 \tag{1.9}$$

$$c_n = \langle f, f_n\rangle \tag{1.10}$$

The reason that (1.10) determines the ideal (Fourier) coefficients is the same reason that taking the dot product between two vectors gives the scalar projection of one vector onto the other. Except in this case the vectors are now real valued functions. A particular set of functions that spans the entire $L^2(a, b)$ is the set of all characteristic functions of intervals (Walter, 1994). The reason is that any function f can be expressed as a linear combination of interval functions $\{h_n(x)\}$ (given sufficiently small intervals).

$$f(x) = a h_0(x) + b h_1(x) + c h_2(x) \dots$$

(1.11)

The idea of using interval functions can be extended to more than one dimension as well. For example, one can imagine a two-dimensional interval function that defines a square bounded region in the plane. From this, we can realize that in order to form any N-dimensional interval function we simply have to form a product between interval functions in each of the N dimensions.

$$h(x_0, x_1 \dots x_N) = \prod_{n=0}^{N} h_n(x_n)$$

(1.12)

From here, we can see that a linear combination of N-dimensional interval functions can define any function in $\mathbb{R}^N$, since they are able to span the entire space. It then follows that we can create any N-dimensional interval function by forming a product between infinite series in each of the N dimensions. Therefore, by forming a linear combination between each of these products of series, any N-dimensional function can be realized.

$$f(x_0, x_1 \dots x_N) = \sum_{j=0}^{\infty} a_j \left[\prod_{n=0}^{N} \left(\sum_{m_n=0}^{\infty} c_{m,n} f_{m,n}(x_n) \right) \right]$$

(1.13)

By gathering like terms we get:

$$f(x_0, x_1 \dots x_N) = \sum_{m=0}^{\infty} w_m f_m(x_0, x_1 \dots x_N)$$

(1.14)

which is of the same form as (1.6).

An orthogonal basis that fits the criteria of (1.8) and (1.9) are the Legendre polynomials $\{P_n\}$. The Legendre polynomials are obtained by orthogonalizing $1, x, x^2$ ... with respect to dx. A proof of orthogonality can found in (Jackson, 1941), and a proof that a Legendre expansion converges to $f \in L^2(-1,1)$ can be found in

(Walter, 1994). A recurrence formula used to find Legendre polynomials is as follows:

$$xP_n(x) = \frac{n+1}{2n+1}P_{n+1}(x) + \frac{n}{2n+1}P_{n-1}(x) \tag{1.15}$$

These polynomials are not normalized, so each must be multiplied by the factor $(n+\frac{1}{2})$.

The first six Legendre polynomials are shown in Table 1.1.

Table 1.1 First six Legendre polynomials

n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}(3x^2-1)$
3	$\frac{1}{2}(5x^3-3x)$
4	$\frac{1}{8}(35x^4-30x^2+3)$
5	$\frac{1}{8}(63x^5-70x^3+15x)$

CHAPTER II

MIMIC SYSTEM DEVELOPMENT

The mimic system developed here takes a role of an observer. Being designed to be part of a larger agent system that interacts with the environment, it observes the inputs that are given to the environment by the agent and the outputs from the environment that result.

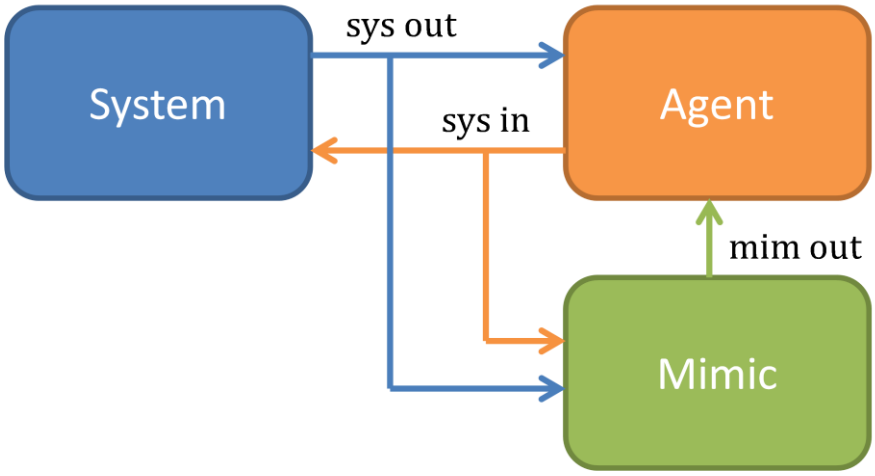

Figure 2.1 System relationship diagram

The mimic has no control of the inputs to the environment, although the agent system is certainly free to produce inputs to the environment that are conducive to mimic learning. The purpose of the mimic system is to provide the agent with information about the future. In a control system type situation, this would assist the control system in maintaining system stability, preventing failure, etc. For an artificial intelligence task it could involve motion control, or even predicting the next action of an opponent. This is well suited for systems that are discrete in nature, but continuous systems could be sampled and predicted as well.

Each of a system's inputs, outputs, and state variables can be represented as real numbers. Sometimes systems are restricted to a finite number of states as in the case of finite state machines, but a finite number of states can be represented by single a real number. Generally speaking, if the number of possible states in a system is countably infinite the state information can be represented as a single real number. In practice however, the computer used to implement the mimic system has memory constraints. For the case of a finite number of states, the simplest method of storing the state information would be to use an unsigned integer type, although floating point types could be used as well. If the system has one or more continuous state variables (components of a state vector), floating point types provide more flexibility. The mimic system is designed to take numbers in the range of -1 to 1 as inputs. This is for a number of reasons. First of all, any computer used to run the mimic system is already limited in the range of numbers it can represent, so linearly scaling the inputs can easily change the domain to (-1,1). Also, limiting the range of the inputs and outputs of the system prevents overflow errors. Another reason is that in practice, most systems have natural boundaries for the inputs and outputs anyways.

The mimic system finds the expected value of the system output, given a number of current and past inputs and outputs. Because of this, it can be considered a kind of multiple regression. However, this method differs from normal multiple regression in a number of ways. First of all the usual method of determining coefficients in regression analysis is to use the least squares method (Ott, 2010). The method used here relies on incrementally reducing the error, which results in the coefficients "drifting" toward optimum values. Standard regression analysis is not designed to fit a function to an infinite number of samples. The mimic system is designed to continually adapt, accepting an unlimited number of samples. Also, the mimic system requires that the samples be sequential so that it can determine time dependant relationships. Also in regression analysis the function is fit to a finite sample set, usually with all samples being treated equally. For the mimic system, its function is more strongly dependant on recent samples. Usually, regression analysis

uses the least squares method to optimize the coefficients of a single polynomial. Although the mimic output functions are able to be represented by large polynomials, they are represented as linear combinations of orthogonal functions and the optimization is on the weights of those functions. Perhaps the process most similar to this is multiple time series analysis. The difference between the two is that the mimic system developed here is designed to take inputs, and thus is designed to make a hypothesis that is a function of the inputs as well.

2.1 Calculating Legendre polynomials

Legendre polynomials were chosen as the basis of choice for a number of reasons. Using trigonometric functions as the basis presented two significant issues. The first is related to the problem of calculating the result of $sin(x)$, which is an irrational number. Depending on the hardware available, trigonometric functions are calculated using power series, lookup tables, or CORDIC (Ercegovac, 2004). Using a power series to represent a trigonometric function as a polynomial is obviously less efficient then calculating orthogonal polynomials directly. Lookup tables require access to cache memory, RAM, or maybe even storage memory. Using a CORDIC algorithm on modern computers is generally slower than either of these previous two methods, and requires many shifting and adding operations.

Rather than using a brute force method of calculating the Legendre polynomials, the algorithm takes advantage of the fact that the exponents of x are either entirely even or entirely odd. By calculating x^2 and storing the result, and by changing the polynomials from the form

$$a + bu + cu^2 + du^3 + \cdots$$

to

$$a + u\left(b + u\big(c + u(d + \cdots)\big)\right)$$

the total number of number of multiplications for a polynomial of degree n is $\left\lceil \frac{n}{2} \right\rceil$. In order to expedite the calculation of Legendre polynomials during run time, all of the

coefficients for the polynomials from zero up to a certain degree are pre-calculated by using Eq. 15, and then stored in a table. The table is populated upon creation of a mimic system object. Using this table, Legendre polynomials for each of the variates are calculated. The product of these then gives the result for a normalized orthogonal function in the multivariate series expansion.

2.2 Stochastic term search

At the time of this writing, it is not possible to purchase a computer with an infinite amount of memory. Therefore, is not possible to store an infinite number of coefficients for representing an infinite series of Legendre polynomials. We address this problem by limiting the total number of orthogonal terms for each output to some number T. This can only represent the original system to a certain degree of accuracy, but if the terms are chosen carefully the accuracy of the mimic system can be maximized for a given T. Each output of the mimic system is treated as a separate entity. That is, each output (a single valued function) consists of a linear combination of orthogonal functions (called *terms*), and the outputs do not share terms. In other words, instead of a single set of terms being chosen for all outputs, and the outputs having their own set of weights for those terms, each output has its own set of terms and corresponding weights. To reduce algorithmic complexity, the number of terms for each output is *T*. There are advantages for both methods. Let's say that the total amount of memory needed to describe a term (a description of which degrees Legendre polynomials are to be used) is m_t, the amount of memory needed to describe a weight for a term is m_w, and the total number of outputs of the mimic is o. For the outputs to share terms, the total amount of memory used will be

$$Tm_t + oTm_w. \tag{2.1}$$

For outputs having their own set of terms, the memory used will be

$$To(m_t+m_w) \tag{2.2}$$

At first glance it seems it would always be advantageous for the outputs to share terms, however this doesn't take into account the accuracy lost when sharing terms because the optimal set terms for each output may vary considerably. For the worst case scenario where the outputs do not share terms, the outputs share no like terms. Therefore, in order for the sharing scheme to have least the same amount of accuracy for all outputs as the non-sharing scheme involves all outputs in the sharing scheme having all terms for each output in the non-sharing scheme. This changes the amount of memory used for the sharing scheme to

$$o(Tm_t + oTm_w) \tag{2.3}$$

for this case. The optimal solution would actually be a combination of these two methods, with some terms being shared and some non-shared, and would depend on the values of all these variables as well as the characteristics of the system. However, the method used by the mimic system is the more resource intensive and more accurate method.

Another consequence of limited memory is that it is impossible to calculate an infinite number of degrees of Legendre polynomials. There is no attempt made to optimize which degrees are used for a couple of reasons. This first is that the weights of the terms are usually inversely proportional to the degree of the polynomial, and so the lowest degree terms polynomials are far more likely to be useful that than the ones with a very high degree. After all, the weights are a Cauchy sequence. The other reason is that given this property it was felt that the additional complexity of searching for optimal degrees would put an undue calculation burden on whatever machine was to run the mimic system. It simply wouldn't be warranted to do this given the much larger amount of time it would take to attempt this.

Another way of storing all of this information would be as one large polynomial. If this was the case the information would be stored as an array of coefficients. To update the mimic output function, one could use a method of adding/subtracting from the large polynomial in units of orthogonal terms. In the

extremely likely case that Legendre polynomials of both even and odd degree were utilized for each variate, the number of coefficients that would have to be stored (and number of memory elements used) is

$$\prod_{v}^{V}(d_v + 1) \tag{2.4}$$

where d_v is the maximum degree of Legendre polynomial for a variate v. If all of the d_v were the same, this would become $(d + 1)^V$ where V is the number of variates. This non-coincidentally is the same as the number of memory elements it would take to describe the weights all of the possible terms for the same maximum Legendre polynomial degree. The problem with trying to store all possible information up to a certain polynomial degree is that the amount of memory required grows exponentially with number of variates. In the end because storing as individual terms gives the flexibility to ignore terms and thus save more memory, it was the preferred choice.

An output of the mimic system will give the best representation of the corresponding output of the original system if its *T* terms are those that have the T largest magnitude of weights in the ideal infinite series. When a mimic object is first instantiated the list of terms is randomly populated. This involves choosing a random set of polynomial degrees. The weights are always initialized to zero. It is assumed that the Legendre polynomials used most are the ones with lower degrees. Therefore, each degree is assigned randomly with a ramped probability density

$$P(d) = \frac{2}{D} + \frac{2d}{D^2} \tag{2.5}$$

where D is the maximum degree. After each iteration (after every sample received), the weights of all the terms for each output are updated. Although the weights continually change, each weight tends to drift toward a specific value. Because of this the weights can be ranked according to contribution to the accuracy of the mimic system after they have been updated a certain number of times. The mimic system

begins with each output given a set of terms at random, so over time this set must change in order for the mimic system to attain highest accuracy. This is accomplished for each mimic output by periodically sorting the terms in order of the magnitudes of their weights, and then replacing the lowest ranked *k* of them with *k* new random terms. This is a stochastic search in the space of all possible terms for the best weighted *T* terms. Unlike the initial random set, choosing the degrees of the new terms is done with a flat probability distribution. When replacing the set worst terms with a set of random terms, the random terms are checked to make sure they are not duplicates of terms that are already in the term list. Duplicate terms would result in wasted space. Two questions arise when using this technique: How many terms should be replaced, and how often? The number of terms that continually get replaced affects how fast the mimic system is able to increase in accuracy, and also the final amount of accuracy it converges toward. If the mimic system only replaces one term at a time, the search process for better terms will be very slow and so its accuracy will increase more slowly. However, because only one term is being devoted to searching, it will reach a higher final accuracy. The opposite would be true if an entire half of the terms were devoted to searching. The optimal solution involves initially using a large number of terms for searching, and gradually decreasing that number to allow for a fast increase in accuracy, and high final accuracy.

A discrete dynamical system may have any number of state variables. Ideally, the mimic system would have perfect models describing these states. However, because of the chaotic nature of nonlinear systems in general, even good models for the state variables won't prevent the mimic's variables from eventually becoming completely different from the actual state variables. In addition, models for how state variables interact generally would have to be "guessed", and then tried to see if they work. Attempting to guess models for state variables is in general futile because even a guessed model that was perfect would take many system iterations before there was any indication that it worked. All other guesses would quickly succumb to chaos before they ever had a chance to prove that they were somewhat correct. Not only

would the relationships of the outputs to the state variables and the relationships of state variables to themselves have to be guessed, but the states themselves as well. This adds even more difficulty, as very small differences in initial states can produce very different results over time (Martelli, 1999). This extreme difficulty in identifying states and state relationships in a system creates a need for a different technique. By looking at (1.5), we can see that the output of a system is generally a function of past inputs to the system. By including in the *variate* vector a number of past inputs and past outputs of the system, we can take advantage of this additional information. The term *variate* will refer to any variable that is a parameter to an output function of a system. In the case of the mimic system, this refers to a member of the union of: the set of all inputs, a subset of all past inputs, the set of all reference outputs, and a subset of all past reference outputs. In the case of a reference system it is a member of the union of: the set of all inputs, the set of all reference outputs, and the set of all reference state variables.

2.3 Term weight adjustment

The usual application of a Fourier series of orthogonal functions involves finding the weights of the orthogonal functions by using (1.10). Because this involves integration with the function to be approximated, this presents a problem for a number of reasons. First of all, do we not know what the function is (and we will never precisely know). Say for example we received 100 samples that gave a perfect fit to $f = x^2$. It would only take one sample that didn't fit to invalidate that model. Because of the nature of the problem (taking samples of points) it is impossible to get a sample of every point in $\mathbb{R}^n$, which is one of the two primary reasons a fully accurate model for the original system is impossible. Generally the sample points are not equidistributed over the variate domain even as the number of samples goes to infinity, and because some regions of $\mathbb{R}^n$ may not be in the domain at all, any sort of summation approximation to integration isn't feasible. Even if methods were used to try and compensate for this by flattening the distribution and filling in the gaps, this

would require the entire variate domain be divided into regions with each region assigned a value. This would end up being equivalent to trying to approximate the function using a lookup table, and would be extremely memory intensive. Another problem is that because of unknown factors in the system, for each possible input to the system there are multiple possible outputs. This is the other of the two reasons the system can never be fully accurately modeled, because the probability distributions describing the outputs is once again a function defined at every point on the interval $(-1,1)$. Instead of integrating to find the weights, a process of error reduction is used. The orthogonal Legendre polynomials are used to approximate square integrable functions on the interval$(-1,1)$. They span $L^2(-1,1)$ so they form a complete basis. By using error reduction instead of integration, it doesn't matter that the Legendre series is not orthogonal on the range of outputs, it only matters that it is complete on that range. Because the Legendre series is complete on the interval $(-1,1)$, it is complete on any range inside that. If the range of an output is not the entire interval$(-1,1)$, how the function is defined outside of its range is irrelevant. The error minimization process allows the function to be defined outside the range as whatever results from reducing the error inside the range. In addition, because the process minimizes the error with every new sample, the outputs in the ranges with higher sample densities will end up with lower amounts of error. The weight w_t of a term is adjusted with every new sample by using the relation

$$w_{t+1} = w_t + \delta L(M_t - \widetilde{M}_t) \tag{2.6}$$

where M_t is a reference system output, $\widetilde{M}_t$ is a corresponding mimic output, L is the orthonormal function of the term, and δ is a constant. $\widetilde{M}_t$, and L are functions of the variate vector.

This intuitively makes sense, because the amount of change is proportional to the value of the orthogonal function at that variate vector. If the value happens to be zero, this indicates that the orthogonal term is not capable of contributing to the reduction of error at that point. This process adjusts the weights of each term so that

the error between the system output and the mimic output tends to decrease with each iteration. The positive constant δ determines how much adjustment to the weights occurs. If δ is very small, the mimic system will take a long time to reduce the error between it and the original system. If it's too large, the mimic system may become unstable. Instability happens when the amount of error tends to increase rather than decrease. This error increases because under certain conditions, the mimic system can overcompensate. The idea of the weight adjustment mechanism is to bring the output of the mimic slightly closer to that of the original system for a given variate vector. Because an output of a mimic system is given by

$$\widetilde{M} = \sum_{i=0}^{T} w_i L_i \tag{2.7}$$

the total change in a mimic output for a given variate vector is given by

$$\Delta\widetilde{M} = \widetilde{M}_{t+1} - \widetilde{M}_t = \delta(M_t - \widetilde{M}_t) \sum_{i=0}^{T} {L_i}^2 \tag{2.8}$$

From here, a recurrence relation of the error can be derived.

$$B = \delta \sum_{i=0}^{T} {L_i}^2$$
$$\widetilde{M}_{t+1} = \widetilde{M}_t + B(M_t - \widetilde{M}_t) \tag{2.9}$$

Let's consider M_t to be the sum of two functions, a mean value Υ (the function we wish to approximate) and a random variable X with mean 0.

$$\widetilde{M}_{t+1} = \widetilde{M}_t + B(\Upsilon + \mathrm{X} - \widetilde{M}_t)$$
$$\widetilde{M}_{t+1} - \Upsilon = \widetilde{M}_t(1 - B) - \Upsilon(1 - \mathrm{B}) + B\mathrm{X}$$
$$\widetilde{M}_{t+1} - \Upsilon = (\widetilde{M}_t - \Upsilon)(1 - \mathrm{B}) + B\mathrm{X}$$

We will now designated the error $E_t = \widetilde{M}_t - \Upsilon$.

$$E_{t+1} = E_t(1 - \mathrm{B}) + B\mathrm{X} \tag{2.10}$$

If $\Delta\widetilde{M}$ is large enough in magnitude, the error can actually increase when the mimic is given the same variate vector.

$$\left|M_{t+1} - \widetilde{M}_{t+1}\right| > \left|M_t - \widetilde{M}_t\right| \tag{2.11}$$

For any orthogonal function L formed from the products of Legendre polynomials:

$$|L(\pm 1, \pm 1, \pm 1 \ldots .)| = 1$$

For these points,

$$\Delta\widetilde{M} = \delta(M_t - \widetilde{M}_t)T \tag{2.12}$$

where T is the number of terms. If the orthogonal function $\in \mathbb{R}^2$ these points would be the corners of a square, and if it was in $\mathbb{R}^3$, it would be the corners of a cube. These are the points where the most change occurs after weight adjustments, and where the worst case scenario would happen if we were trying to prevent an average increase in error. Our ultimate goal is to find a maximum δ which under all conditions, never results in instability. As previously stated, the adjustments of weights cause a mimic output to move its result in the direction of where the system output is, and moving toward the system output may result in moving past the system output. The error tends to increase whenever

$$\left|\Delta\widetilde{M}\right| > 2\left|M_t - \widetilde{M}_t\right| \tag{2.13}$$

Considering the worst case, we can say now that there can be no case (no series of outputs or variates) in which the error will always tend to increase if

$$\left|\delta(M_t - \widetilde{M}_t)T\right| < 2\left|M_t - \widetilde{M}_t\right|$$

or

$$\delta < \frac{2}{T}$$

The mimic system by default sets $\delta = \frac{2}{T}$, however δ can be changed to a value greater than this by multiplying by an acceleration factor α. When α > 1, the error of the mimic system changes more quickly. This can result in a much faster decrease in error, instability in the mimic system, or alternating between the two. As will be shown later, sometimes the mimic can show some initial instability, but will eventually become stable after the weights adjust themselves.

An issue with the error reduction method is the fact that every time the weights are adjusted the mimic outputs change suddenly, especially in the corners of the vector space where an output can change between -1 and 1 after one iteration. In the interest in presenting a more reliable and accurate output to be presented as the expected value of the reference output, the average is taken of the values of weights over time. The measured mean $\bar{X}$ is traditionally defined as an arithmetic mean.

$$\overline{X}_n = \frac{1}{n}[X_0 + X_1 + \cdots X_n]$$

However, because the mimic system is one that must continually accept new data points from an infinitely large set of data, a *modified moving average* is more appropriate.

$$\overline{X}_{t+1} = \frac{M\overline{X}_t + X_t}{M + 1} \tag{2.14}$$

Using this directly will unfortunately give for a small n an $\overline{X}$ that is on average more erroneous than that of an arithmetic mean. To fix this, an averaging technique is used that has a characteristic of changing from an arithmetic mean to a modified moving average given increasing numbers of samples.

$$\overline{X}_{t+1} = \frac{\frac{Mn}{M+n}\overline{X}_t + X_t}{\frac{Mn}{M+n}+1} = \frac{Mn\overline{X}_t + (M+n)X_t}{Mn+M+n+1} \tag{2.15}$$

In order to know these values, n must be incremented with every new sample. However this is not possible as this mimic system is designed to run ad infinitum, and there is a finite amount of memory available. Because of this, every γ_m stops incrementing its n when it reaches a very large number. This does not cause any sudden issues when this happens, as all functions that use n change very little as n becomes large. When a new term is generated as part of the mimic's stochastic term search, n is reset to zero.

2.4 Reference system generation

In order to test and characterize the functionality of a mimic system in a timely matter, a method of creating random reference systems (systems for mimicking) was developed. This was necessary not only because of the large amounts of time it would take to run experiments many different real world systems in real time, but because also of the difficulty in creating an interface, and finding enough examples of sufficient complexity. The amount of complexity in a system is proportional to the amount of information required to describe it (Zgurovsky and Pankratova, 2007). By varying the amount of information required to describe these reference systems, systems different levels of complexity can be created. The method used to generate the reference systems was designed to make them highly non-linear, and thus unlikely to be able to be mimicked with other methods such as genetic programming or neural networks. Additionally, it would have been easier to form these systems using orthogonal series, but this would bias the types of systems being created into ones that would be more easily solved by using orthogonal series. The technique that was used was to define a set of reference vectors that describe what the next outputs and states

of the system should be given the current inputs, outputs and states. Each of these reference vectors is encoded as arrays of numbers, but can be thought of as a vector existing in a vector field where the vector is $o + s$ dimensional, and the space is $i + o + s$ dimensional where i is the number of inputs, o is the number outputs, and s is the number of states. In order to be able to describe the behavior of the system at the infinite number of points in the field, an interpolation method is employed. Specifically, the vector $\hat{v}$ at a point $\hat{p}$ describing the outputs and states is determined using the following:

$$\hat{v} = \frac{\sum_{i=0}^{P} \frac{v_i}{\|\hat{p} - p_i\|^2}}{\sum_{i=0}^{P} \frac{1}{\|\hat{p} - p_i\|^2}} \tag{2.16}$$

P is the total number of points used to describe the reference system, and v_i is a reference vector at a point p_i . This is basically a weighted average of vectors with the weights inversely proportional to the square of the distance.

The reference vectors are placed randomly in the space, and each of the vector components are assigned random values in $(-1,1)$. The number of reference vectors can be changed depending on the desired complexity of the reference system. In order to maintain the same average degree of complexity for spaces of different dimensions, the number of points must increase with increasing dimension. This is if the complexity is gauged by counting the number of inflection points on plots of the outputs if they were measured by travelling on random straight lines through the space. Figure (below) shows two generated reference systems. These systems both have two inputs and one output. The xy plane is the plane of the two inputs, and three cross sections show slices through the z dimension, which is the output. The colors shown give the *next* value of the output, where green = 1, and green =-1. The first system has 16 reference points, and the second has 256.

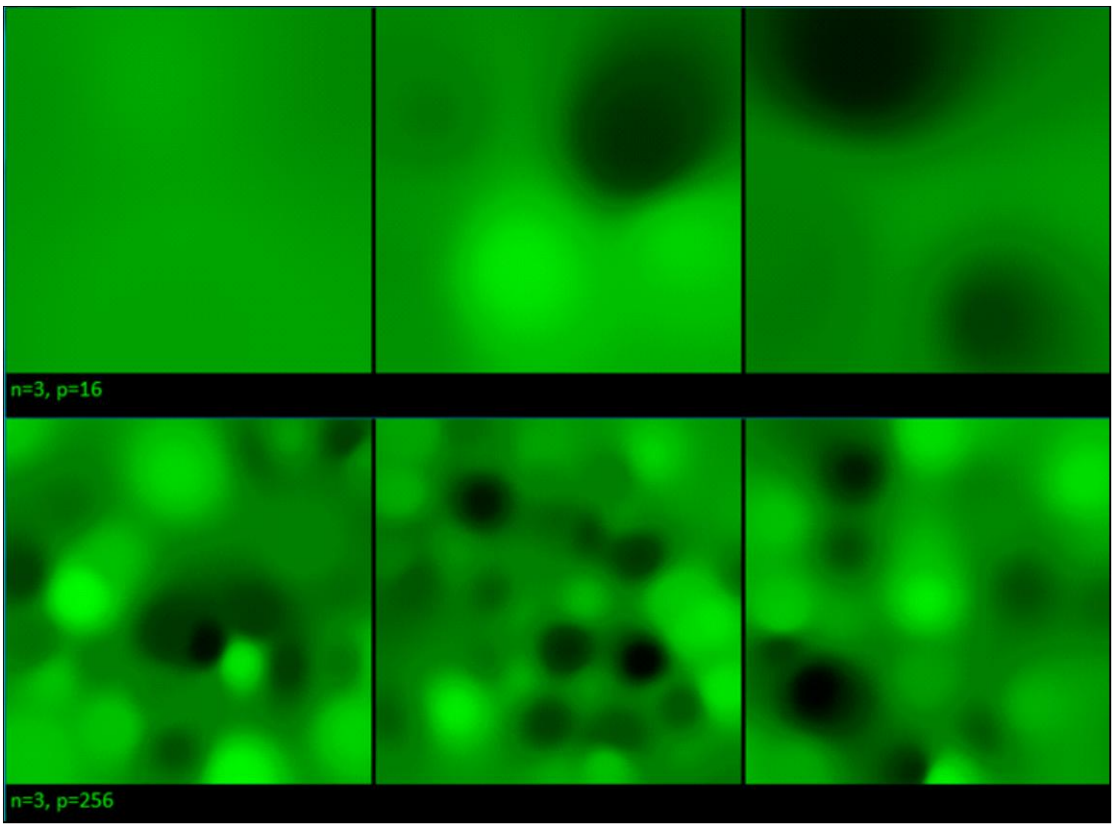

Figure 2.2 Examples of reference system generation

CHAPTER III

RESULTS

To perform tests on the mimic system, an iterative loop had to be created. Figure 3.1 shows the processes in the loop.

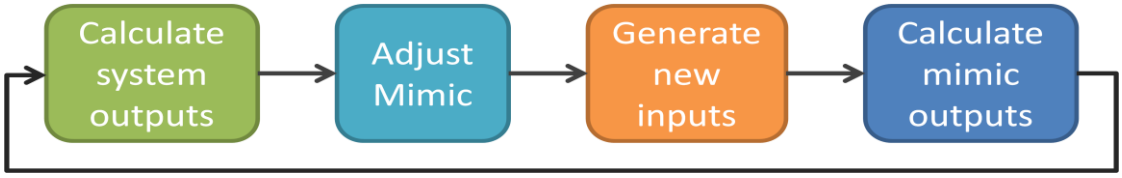

Figure 3.1 Test iteration loop

The reference system begins by calculating the output for the input information stored. The mimic system, which had already calculated its expected value for the output, then adjusts itself to better approximate the system function. After a new set of inputs are created and stored, the mimic system then calculates what the next expected output is. A number of variables were manipulated to observe the effect of the mimic's behavior. The naming of the variables is a follows:

n_i = Number of inputs

n_o = Number of outputs

P_i = Number of past inputs

P_o = Number of past outputs

T = Number of terms

R_m = Proportion of terms always used for searching

R_e = Proportion of terms not always used for searching, initially used for searching

R_{ed} = Decay factor of extra searching terms

α = Acceleration factor

In order for the reader to get a visual example of how a mimic system evolves over time, a simple example of a well defined function was hard coded, and then mimicked. Figure below demonstrates the approximation of the two-parameter, single valued function *J*

$$J_{t+1} = cos(4x + 2J_t)\, cos(4x + 9J_t)$$

after many iterations. Black indicates -1, and green indicates +1.

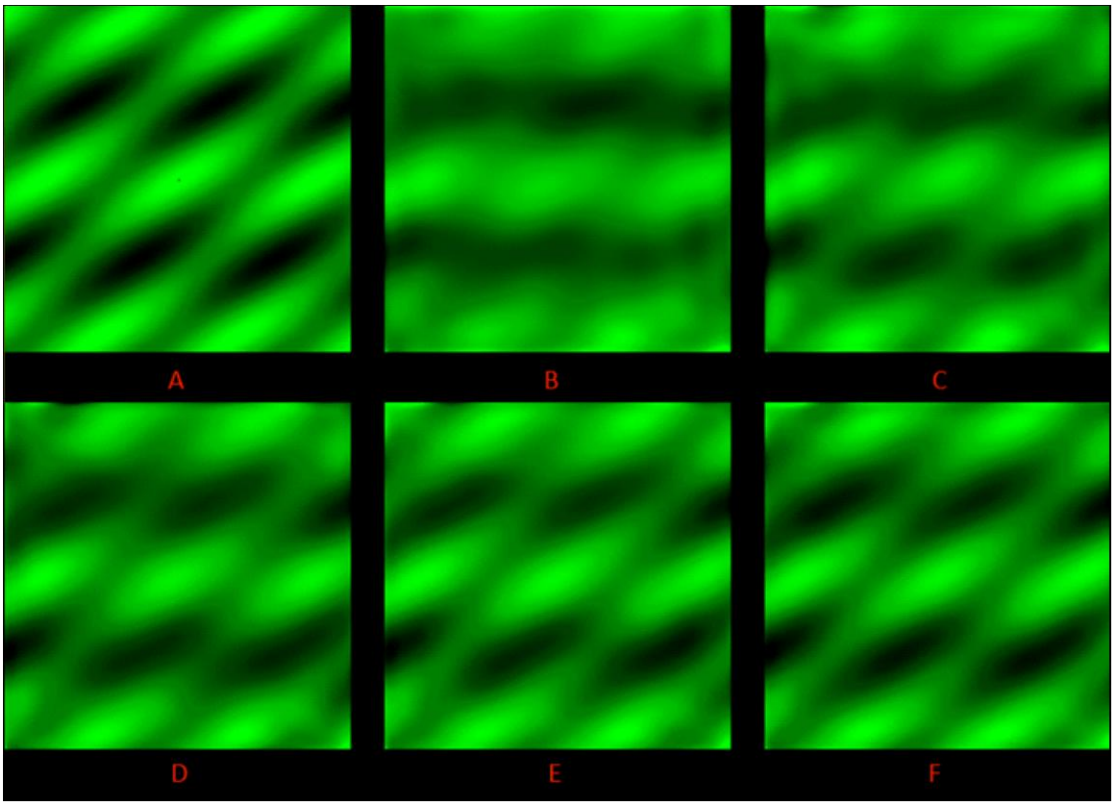

Figure 3.2 Example of 2D function approximation

'A' shows the reference system function, while 'B' through 'F' show the mimic function starting at 1000 iterations going up to 5000 iterations in increments of 1000. It is shown that the mimic function begins to approximate the system fairly well after 5000 iterations. Even at a relatively low number of iterations, it quickly begins to look

vaguely like the reference system. For this function, the domain of the output was the entire interval (-1,1).

The purpose in varying the number of points used to describe the generated reference systems is to vary the amount of complexity. In order to demonstrate how the characteristics of the mimic system change with the complexity of the reference system, a test was run on 90 randomly generated systems to see which degrees of Legendre polynomials were used the most. Reference systems were generated with nine different numbers of points, $2^1 \rightarrow 2^9$. Ten systems of each were generated. A number U for every combination of degree d and number of points was calculated to give a rough measure of how much each degree was used. Each term in the series expansion consists of a weight, and a list of polynomial degrees. The amount (we'll call k_t) a degree is used in a term equals the number of times it appears in the degree list. If the weight of a term is w_t and there are T terms,

$$U = \sum_{t}^{T} k_t w_t$$

A plot of U summed over the 10 systems as a function of degree and number of points is shown in Figure 3.3, where black=0, and white=40.

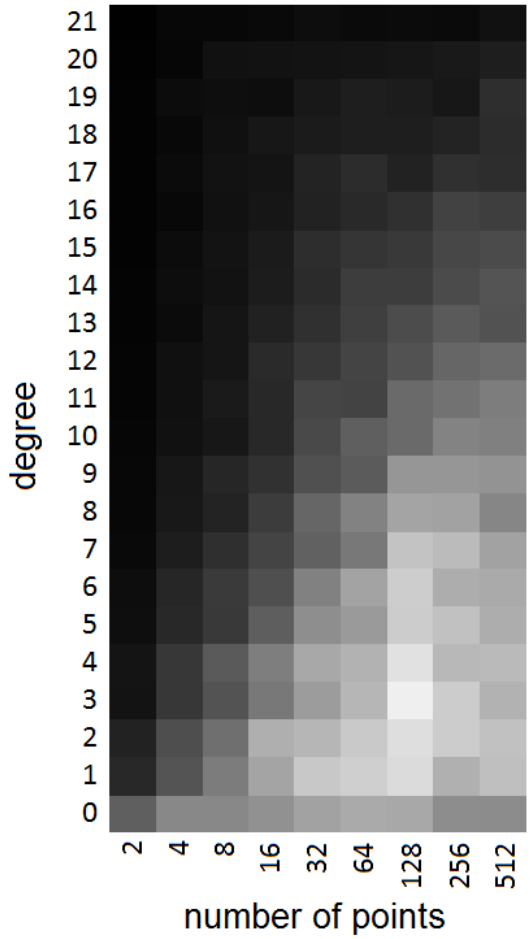

Figure 3.3 Polynomial degree use vs. number of reference points

As shown, the more complex the system the more the higher degrees of Legendre polynomials are used. A similar plot was created to show the usage of the different polynomial degrees of an individual reference system over time. The reference system generated had 20 reference points, one input, one output (known state), and one unknown state. Figure 3.4 shows how the degree usage of mimic system changed over time for this system.

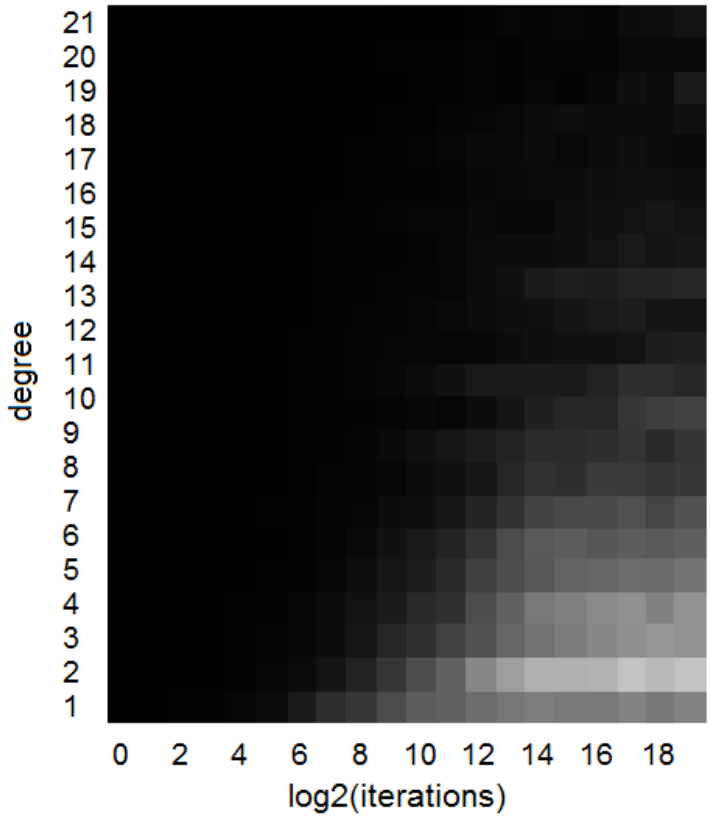

Figure 3.4 Example of polynomial degree use over time

For this particular trial, the average error was plotted as a function of time as well. This is shown in Figure 3.5.

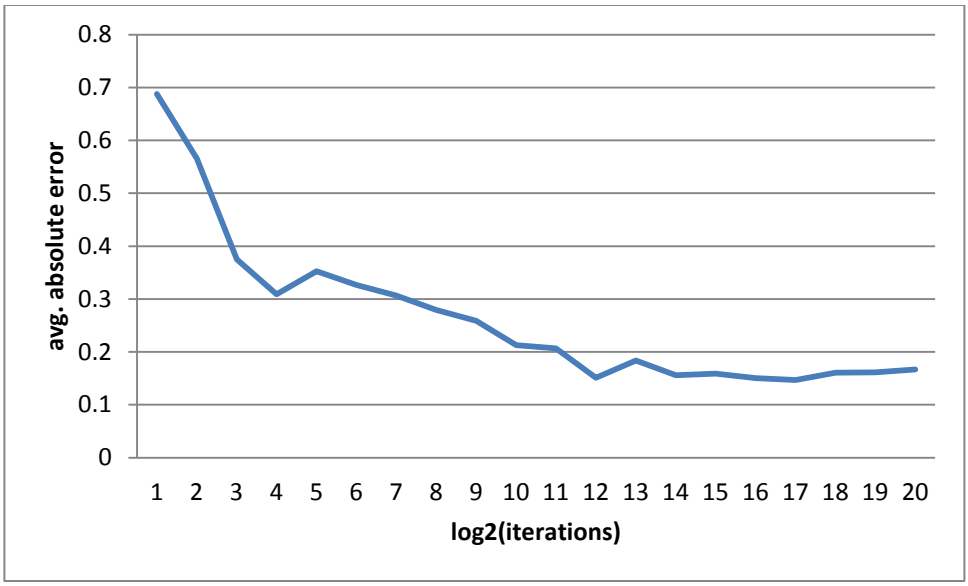

Figure 3.5 Example of average error over time

In order to show how the mimic system varies its behavior with the changing of its control variables, another series of tests were conducted. For all of these tests, the same set of 50 randomly generated systems was used. These systems varied in their complexity by varying the number of reference points used to define them. The number of reference points was a random number between 1 and 256. All of the 50 reference systems had one input, one output (known state), and one unknown state. All of the results shown for average error are the averages of the errors of all 50 systems. The systems were not saved in memory. Each system was represented by a single number which was used to seed the random number generator. Because the pseudo-random number generator is actually deterministic, each system was able to be recreated by simply seeding the random number generator with its seed number before the creation process. The inputs given to the system were also pseudo-randomly generated. The accuracy of the mimic system is ultimately limited by the number of terms. As the number of terms in the series expansion increases, the additional benefit of adding more terms decreases. Figure 3.6 shows the graph of an experiment where each system ran for 10000 iterations.

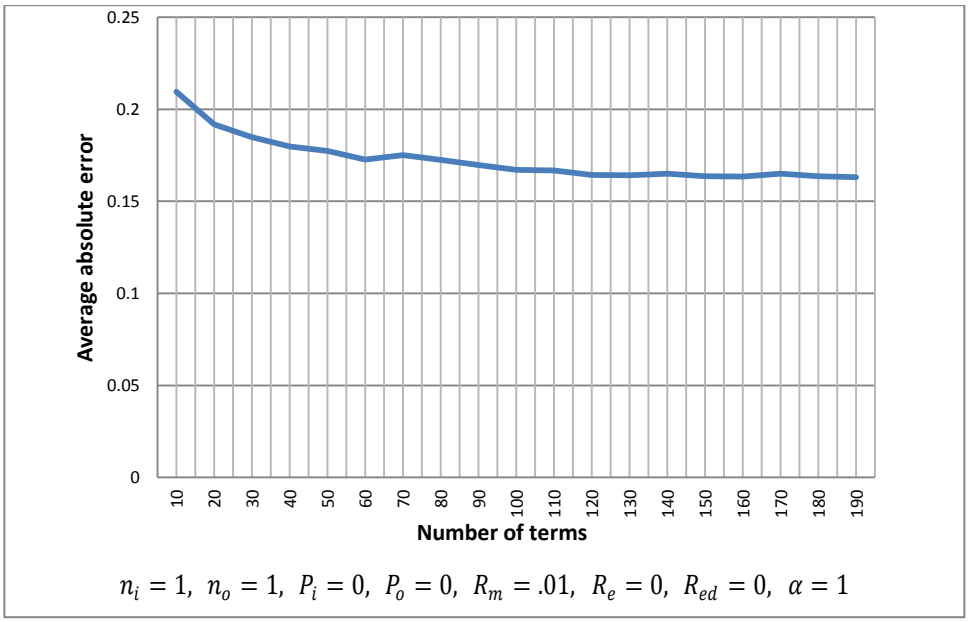

Figure 3.6 Number of terms vs. mimic error

The acceleration factor α when chosen carefully can also improve the speed at which the mimic system reduces its error. Shown in Figure 3.7 are the results from using acceleration factors from 1 to 20 on trials of 100 iterations. As shown, there is a point at about $\alpha = 12$ where the error reaches an average minimum. The error to the left of the point increases because the term weight adjustment is simply less, and so the mimic does not adjust as quickly. The error increasing to the right is explained by the weights being on average adjusted too much.

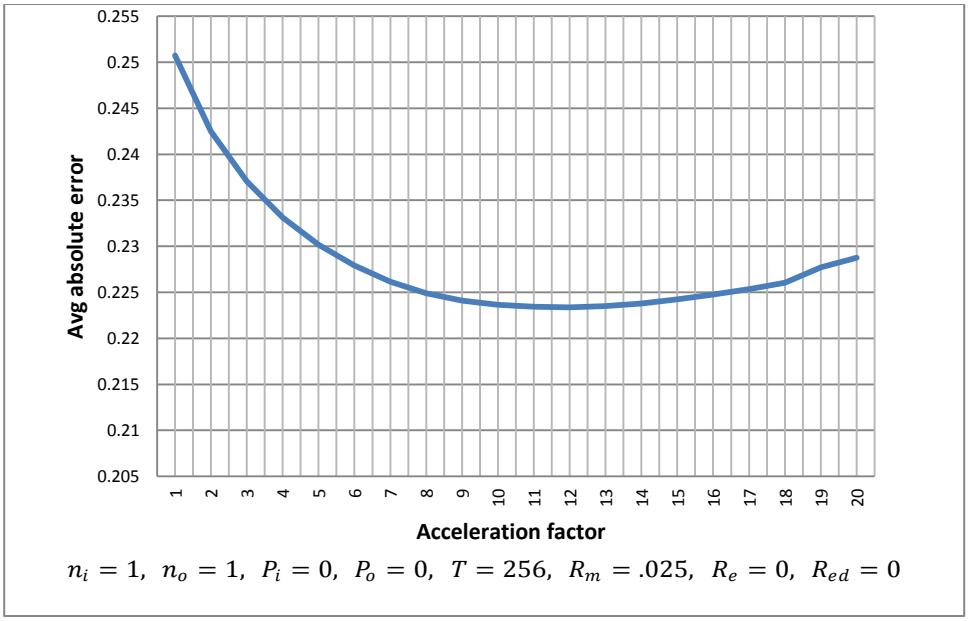

Figure 3.7 Acceleration factor vs. mimic error at 100 iterations

A similar experiment was conducted with changing the acceleration factor, but this time with the number of iterations being 1000. The global minimum is now at a smaller α because the term weights are now closer to being optimal. If the term weights were optimal, this graph would be strictly increasing because any adjustments in weight should only increase the average error. These results are shown in Figure 3.8.

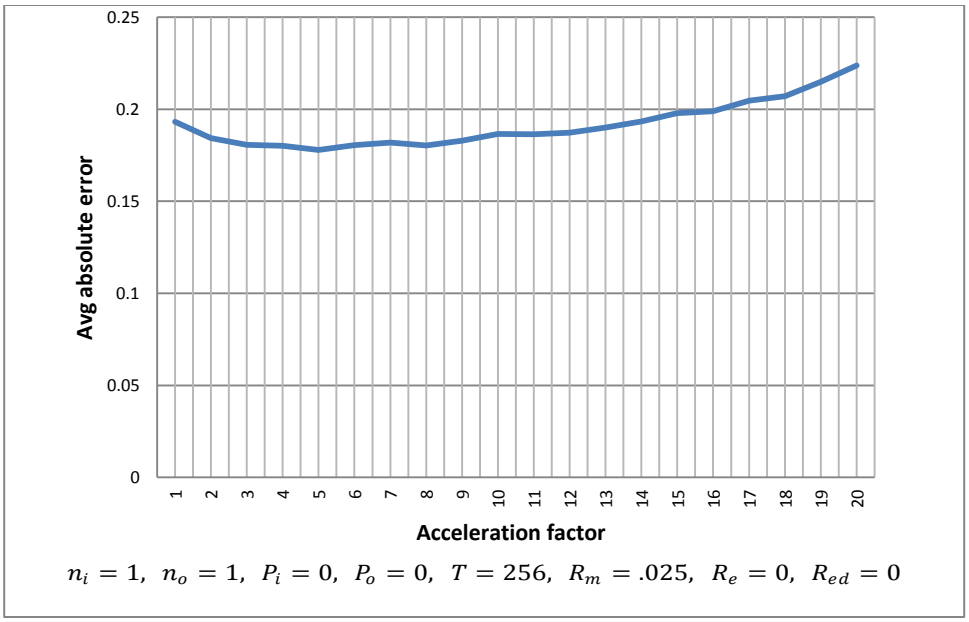

Figure 3.8 Acceleration factor vs. mimic error at 1000 iterations

CHAPTER IV

CONCLUSIONS AND FUTURE RESEARCH

In the pursuit of writing more efficient code, a significant improvement could be made in reducing the amount of memory used for storing term information. Usually a data type in modern computers is either 32 or 64 bits long. This number of bits is significantly more than is necessary for storing a positive integer representing a Legendre polynomial's degree. If information regarding multiple polynomials' degrees terms and be represented in a single memory slot, a significant memory savings can be achieved. If the amount of memory used to represent a polynomial degree is reduced by a factor of r, then the amount of memory needed to describe a normalized orthogonal function is also reduced by a factor of r. This paves the way for more terms, and more accuracy. There would however be a small speed penalty because of additional processing.

Sometimes systems are subject to sudden, long term changes in behavior. These occurrences would cause the average error between it and the mimic to suddenly jump before the mimic gradually adapts. It may be possible in the future to include a mechanism that detects these sudden jumps, and responds by making the mimic more aggressive in trying to minimize the error. The assumption would be that the system has undergone a major change, and significant changes need to be made to the mimic. The response may include briefly increasing the acceleration factor and the number of terms devoted to searching for better terms.

As previously stated, the mimic system gives an expected value of the next output of the reference system or environment. If we consider the output of the environment to be a random variable, then we expect it to have a probability distribution. For many distributions, especially ones that are bimodal, giving an expected value is a very poor estimator of what the next value will be. Figure 4.1

shows two examples of one-dimensional probability distributions, one where the expected value is a good estimator, and one where it is a poor estimator.

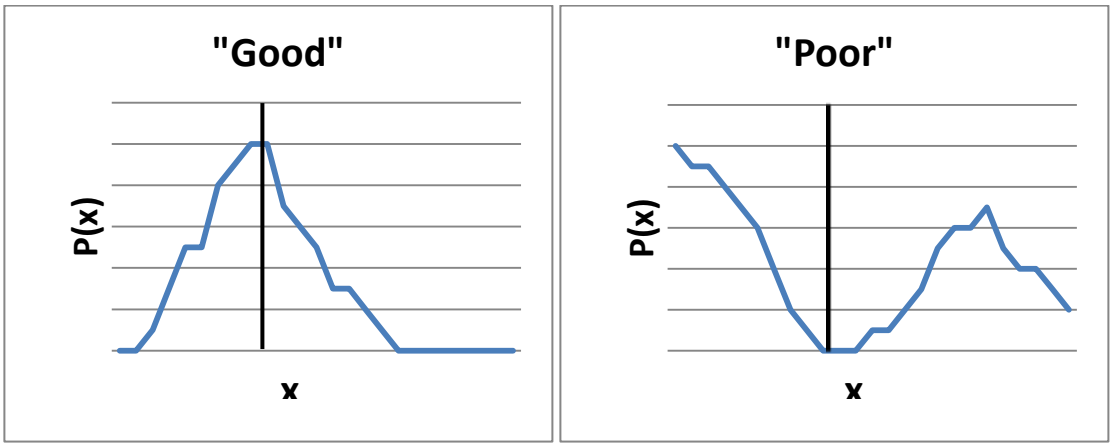

Figure 4.1 Examples of good and poor estimation

For some applications the expected value is always sufficient, and for all applications the expected value is always better than nothing. But it is the case where the expected value is a poor estimator that provides motivation for attempting to approximate the entire probability distribution. Being able to do this is particularly helpful when trying to predict the future multiple steps ahead. This type of situation can be considered to be a Partially Observable Markov Decision Process. Taking this next step would go a long way toward making the mimic system more capable in advanced artificial intelligence problems.

BIBLIOGRAPHY

Eidelman, Y., Milman, and Tsolomitis, *Functional analysis. An introduction*. Graduate Studies in Mathematics, 66. American Mathematical Society, Providence, RI, 2004.

Ercegovac, M., Lang, T., *Digital Arithmetic.* Elsevier, San Francisco, CA, 2004.

Everett, H., III, 1973, "The Theory of The Universal Wavefunction ", in "The Many–Worlds Interpretation of Quantum Mechanics" edited by B. DeWitt and N. Graham (Princeton University Press).

Hruz, B., and Zhou, M.C., *Modeling and control of discrete event dynamic systems*, Springer, London, UK, 2007.

Jackson, D., *Fourier series and orthogonal polynomials*, The Mathematical Association of America, 1941.

Martelli, M., *Introduction to discrete dynamical systems and chaos*, John Wiley & Sons, Inc., New York, 1999.

Ott, L., *An introduction to statistical methods and data analysis*. PWS-Kent Publishing Company, Boston, MA, third edition, 1988.

Walter, G. G., *Wavelets and other orthogonal systems with applications*, CRC, New York, 1994.

Zgurovsky, M. Z., Pankratova, N. D., *System analysis: Theory and applications*. Springer-Verlag. Berlin-Heidelberg, 2007