Predicting Behavior in Agent-Based Models for Complex Systems

Nachi Gupta

Exeter College

University of Oxford

A thesis submitted for transfer from PRS status to DPhil status

Michaelmas 2005

Acknowledgements

I would like to thank Dr. Raphael Hauser and Dr. Neil F. Johnson for acting as my supervisors at Oxford and providing guidance and assistance, Dr. Adrian Agogino and Dr. Kagan Tumer at NASA for an exciting summer working on an interesting project in astrobiology, Dr. Nick Trefethen for academic advice and weekly lunch invitations to Balliol College, Dr. Andrew Pohorille for discussion in astrobiology, and the Clarendon Bursary for financial support.

Abstract

Many important real-world systems feature a population of interacting objects (e.g., financial markets, traffic). These systems often exhibit unexpected macroscopic behavior whose existence would have been hard to predict based solely on a knowledge of the microscopic properties of an individual (e.g., market crashes, traffic jams). Despite the lack of a universally accepted definition of complexity, such real-world systems are typically referred to as 'complex systems'. When conducting scientific research, we often times find ourselves dealing with such systems, which arise in fields ranging from physics and economics to biology and sociology. Regardless of what we are modeling, we are usually curious about the time evolution of the complex system, and thus would like to make predictions into the future. In this thesis, we focus on cases where the complex system is modeled using a multi-agent framework. We then discuss an optimization method to deduce the finite measure describing the composition of heterogeneous agents in the model. Our primary focus is on agent-based artificial market models for financial time-series, but we also discuss a different application of multi-agent modeling to astrobiology for learning the evolution of protein ecologies and which initial distributions govern the learning process. As multi-agent modeling becomes more computationally feasible due to technological advances, we see how versatile its applications are throughout science.

Contents

List of Figures

Chapter 1

Introduction

Wouldn't it be neat to predict the future?

The world around us is a a very detailed and intricate system. In particular, real-world systems tend to comprise of many interacting parts, yielding a macroscopic dynamical behavior which features a complicated mix of unpredictability and predictability at various scales, e.g., ranging from the actions of each atom to each chemical to each animal to the human mind which is in many ways still a mystery leading to politics and society. These are all very particular examples of how it is a complex system, but one can imagine many more examples such as petals on a flower or even global warming. In fact *everything* around us is a complex system! Wouldn't it be neat to predict how this complex system evolves? Well predicting this particular system is beyond the scope of this thesis and may even be philosophically or scientifically argued to be impossible. However, in order to attack such an absurd question, we might find ourselves immediately thinking along the lines of two main questions. The first question is that of modeling. How does one model a complex system? After all it is complex, right! And we notice complexity at different scales. Well, the modeling step is extremely difficult for capturing the intricacies and will always be left with something to desire. The other main question is if given such a model, purely as an observer how could we go about predicting the time evolution of the model for the complex system. This will largely depend on how predictable the model is. If we are dealing with a purely deterministic model, the evolution is trivial. However, even in this case, if we assume some error in both our model and observation of the complex system, intelligently perturbing our prediction to match the observation is no longer trivial.

In this thesis, we focus on a very specific subset of models but certainly an intuitive choice. This is the case of multi-agent based models for complex systems. While little is known about many of the complex systems around us, we often times know that

the factors affecting the system can be modeled by so called 'agents' each of which has some influence on the complex system through time. The aggregate effect of these influences cause the complex behavior within the system. In many applications, this aggregate effect is all we are able to observe despite the underlying agent framework.

In other words, we often times observe complex systems at the macroscopic level and model the system with no information about the microscopic behavior. Agent-based models are meant to capture the microscopic behavior while also exhibiting the macroscopic behavior. A key realization here is that an agent-based model may have interactions that can disrupt or be the cause of an equilibrium or chaos at any point in time, and depending on the complex system we are modeling this may happen frequently or very rarely. In light of this, it is of paramount importance that the multi-agent model resemble the microscopic behavior of the complex system we are trying to model as closely as possible, otherwise we may see chaotic model mismatching happening at random intervals especially in the case of a purely deterministic agent model.

1.1 Financial Markets

Various theories exist with regards to the possibility of predicting future movements in financial markets ranging from significant to impossible. Within this range lies the possibility that financial markets may neither be predictable nor unpredictable all the time, but may instead have periods where they are predictable and periods where they are not (i.e., segments that exhibit non-random behavior and segments that are seemingly random). Evidence for such non-random segments which we call 'pockets of predictability' have been found and shown in the past (e.g., [17, 4]). If we determine a priori when we are in such a pocket, we can trust our predictions more and allow for more effective forecasting.

Let us first reiterate the rationale behind using an agent-based complex system model to predict financial markets. Financial markets produce time-series as do many dynamical systems evolving in time, such as the ambient air-temperature, or the electrical signals generated by heart-rhythms. In principle, one could use any time-series analysis technique to build up a picture of these statistical fluctuations and variations - and then use this technique to attempt some form of prediction, either on the long or short time-scale. There are a multitude of ways for making predictions over time-series, some of which might be based on stochastic differential equations modeling the expected behavior of the given time-series, while others might perform

regressions over parameters shown to be statistically significant. For example, we might like to perform a multivariate analysis of the prices themselves in order to build up an estimate of the parameters in the multivariate expansion, and then run this model forwards. However, such a multivariate model may not bear a relation to any physical representation of the market itself. Instead, imagine that we are able to identify an artificial market model which seems to produce the aggregate statistical behavior (e.g., the stylized facts) observed in the financial market itself. It now has the additional advantage that it *also* mimics the microscopic structure of the market, e.g., it contains populations of artificial traders who use strategies in order to make decisions based on available information, and will adapt their behavior according to past successes or failures. All other things being equal, such a model may be intrinsically 'better' than a purely numerical multivariate one and may even be preferable to many other clever approaches such as certain neural network techniques, which also may not be correctly capturing a realistic representation of the microscopic details of a physical market. The question then arises as to whether such an artificial market model could be 'trained' on the real market in order to produce reliable forecasts.

Although in principle one could attempt to train *any* artificial market model on a given financial time-series, each of the model parameters will need to be estimated, and if the model has too many parameters, this will become practically impossible as the model will become over-determined. For this reason, we focus on training a minimal artificial market model. In this thesis we focus on a simple multi-agent game called the Minority Game [8], a conceptually simple model that exhibits some non-trivial behavior reminiscent of real markets. This particular choice is certainly an over-simplified model for the purposes of modeling real financial data, however, we leave the modeling step to the reader and provide a mechanism for effective forecasting based on tracking the multi-trader population generating the time-series.

Binary Agent Resource games, such as the Minority Game and its many extensions, have a number of real-world applications in addition to financial markets. For example, they are well-suited for studying traffic flow in the fairly common situation of drivers having to choose repeatedly between two particular routes from work to home. In these examples and in particular in the financial market setting, one would like to predict how the system will evolve in time given the current and past states. In the context of the artificial market corresponding to the Minority Game and its generalizations, this ends up being a parameter estimation problem [19]. In particular, it comes down to estimating the composition of the heterogeneous multi-trader

population, and specifically how this population of traders is distributed across the strategy space. We investigate the use of iterative numerical optimization schemes to estimate these quantities, in particular the population's composition across the strategy space. We then use these estimates to make forecasts on the time-series we are analyzing. Along with these forecasts, we also maintain a measure of uncertainty for our estimate. This uncertainty is an important realization for purposes of risk analysis and identification of pockets of predictability, and thus it is very important to estimate the uncertainty intelligently based on the data, without acting overly optimistic (or pessimistic) [14].

Initially we discuss a method for estimating a probability measure describing the composition of heterogeneous agents and after realizing some benefits of moving out of the probability space, we also discuss an extension from a probability measure to a finite measure which is not necessarily normalized to unit total weight. We also look into the problem of estimating the probability measure over a space of agents so large that the estimation technique becomes computationally infeasible and propose a mechanism using many runs with small subsets chosen from the full space of agents to make this a more tractable problem. The final tool we provide here is for bias removal from the estimates. As a result of choosing subsets of the full agent space, an individual run can exhibit a bias in its predictions. We would like to estimate and remove this bias, for which we propose a technique that has been widely used in other Kalman Filtering applications as well [10, 13].

1.2 Protein Ecologies

Multi-agent modeling can make an important contribution to the field of astrobiology, the study of primitive ecosystems that occurred early in Earth's history and may be present of other planets. Interestingly though, in the past, astrobiologists have rarely tried multi-agent models or even complex system models for their problems. In this thesis, we provide a specific example from the subfield of "non-genomic evolution," involving the evolution of protein ecologies that do not contain nucleic acids (DNA and RNA) which form the basis of modern evolution. Protein ecologies are important since simple proteins are easier to make through natural processes than nucleic acids and it is thought that life on Earth might have started with these ecologies. The significant feature of protein ecologies is that unlike nucleic acids, proteins cannot directly replicate themselves. As a result, much like in multi-agent systems, non-

genomic evolution is concerned with the global behavior of the entire protein ecology and its ability as a whole to form new protein ecologies.

The study of non-genomic evolution seeks to find the initial conditions and mechanisms that must be present to form self-sustaining protein ecologies [26]. Unfortunately with current knowledge it is impossible to directly simulate the evolution of a protein ecology. If we model at the scale of low-level chemical processes and look at the direct evolution of these, we find that given current technology, we are in a computationally infeasible domain to create a simulation that would encompass a full planetary ecology. We try modeling at a more feasible scale, by making simulations that model the higher level dynamics of protein interactions that map a protein's chemical structure (amino-acid sequences) to probability distributions over possible functions, which in principle would allow reasonable accuracy and still be computationally feasible. However, since few of these mappings are known at present, a meaningful simulation at this level becomes impossible as well. As a consequence non-genomic evolution is currently analyzed using greatly simplified "plausible" models for the dynamics of the protein ecology [20, 21]. These simplified dynamical models are then run and are evaluated based on a global utility function. This global utility abstractly measures the 'value' of the resulting protein ecology. Unfortunately creating a model that leads to a good final ecology is laborious and as a consequence successful models are so simple that the results of the simulation can be trivially predicted ahead of time and have little meaning.

In this thesis, we show how learning agents can be used to improve this modeling process by automatically learning parameters of protein ecology models that lead to high global fitness, allowing more complex and meaningful models than those currently used in the astrobiology community. In this approach, agents choose properties of individual proteins that affect the dynamics of the system (such as a protein's ability to split another protein) as well as protein properties that are important for producing a viable ecology, e.g., the ability of an ecology to produce lipids needed for a cell membrane. In a continuous process, agents choose protein properties while the distribution and number of proteins is being updated through the dynamics of the ecology. Using reinforcement learning, the agents attempt to make choices that lead to high global fitness, determined by a given utility function evaluating the ecology's viability. In the experiments shown, the mapping between the proteins' properties and the ecology's global utility is more complex and realistic than in the models previously used throughout the astrobiology community. As a consequence ecologies

that are able to achieve high fitness provide a better example of mechanisms that can lead to self-sustaining protein ecologies [12].

Chapter 2

Financial Markets

Throughout this chapter, we present the reader with a scheme for tracking a trader population of agents playing in a multi-agent artificial market model with the caveat that we trust the reader to choose the multi-agent model most appropriate for a specific application as we keep the tracking scheme general and adaptable to any such model. For example, if the real market in question corresponds to a mixed majority-minority game [22], then clearly it makes sense to attempt matching the real-data to such a mixed version of the game.

2.1 The Artificial Market Model

For modeling a financial market as a complex system we would like to choose an artificial market model that best exhibits the microscopic dynamics of the real market that we are aware of, while maintaining the macroscopic behavior. In terms of such models, a great amount of research has been dedicated to proposing multi-agent systems, a large class of which come from Binary Agent Resource games, which we introduce in Section 2.1.1.

2.1.1 A Binary Agent Resource Game

In a Binary Agent Resource game, a group of N agents compete for a limited resource by taking a binary action. Let's denote this action at time step k for agent i by $a_i(\Omega_{k-1})$, where the action is in response to a global information set defined by Ω_{k-1} consisting of all information up to time step $k-1$ available to all agents. For each time step, there will exist a winning decision w_k based on the action of the agents. This winning decision w_k will belong to the next global information set Ω_k and will be available to each agent in the future.

Memory	Decision
-1, -1	1
-1, 1	-1
1, -1	1
1, 1	1

Table 2.1: The left column shows the $2^2 = 4$ possible memory strings and the right column shows the strategy's response to each memory string.

2.1.2 The Minority Game

A particularly simple example of a Binary Agent Resource game is the Minority Game proposed in 1997 by Challet and Zhang [8], which highlights the competitive effects inherent in many complex adaptive systems. Over the past decade, a number of models and variants of the Minority Game have been proposed, and much research has been done in this area.

Let us first provide the intuition and motivation for the word 'minority" in modeling financial markets. Essentially, in financial markets, agents compete for a limited resource, which gives us the minority nature we would like to model. For example, if the minority group is selling, the majority group will force the price up at the following time step because of the greater demand. At the exact same time, the minority group will sell at the overvalued price and gain profit, making the minority group the winners.

We start the game with a group of N agents, each of which holds an assigned strategy. Let a given strategy have memory size m meaning it contains a binary decision for each of the 2^m possible binary strings of length m. An example of a strategy with $m = 2$ is given in Table 2.1.

A given strategy will look back at the m most recent bits of the winning decisions. At time step k, this would be $(w_{k-m}, \ldots, w_{k-1})$. The strategy will then pick the corresponding binary decision. Using our example strategy in Table 2.1, if $w_{k-2} = -1$ and $w_{k-1} = 1$, the strategy would make the decision -1. Before we can explain how the strategy works, we must also define the time horizon, a bit string consisting of the T most recent bits of the winning decisions where T is significantly larger than m. For example, at the beginning of time step k, the time horizon would be $(w_{k-T}, \ldots, w_{k-1})$.

In order for the game to be interesting, we assume some agents hold more than one strategy in what we call their strategy set. We now need to define how the agents should choose amongst their strategies. Each agent scores each of their strategies over the time horizon by giving it one virtual point if it would have predicted correctly and

taking one away if it would have predicted incorrectly at each time step in the past. For example, if the time horizon was $(-1, 1, -1, -1, 1)$, we would assign $+1-1+1 = +1$ virtual points to our strategy in Table 2.1 since the first decision on $(-1, 1)$ to choose -1 would have been correct, while the second decision would have been incorrect and the third one correct again. In this way, we could score all of the agents' strategies, and the agent will simply pick the highest scoring strategy to play. The winning decision at time step k, w_k, is the minority decision made by the agents as a whole.

For ties between the scores of an agent's strategies, the agent will simply toss a fair coin to decide. Further, if there is a tie in the winning decision over all agents (i.e., an equal number of agents picked -1 and 1), we can again toss a fair coin to decide. Both of these, together, inject stochasticity into the system.

We are interested in the time-series generated by the aggregate actions of the agents. We start the series at r_0, where r stands for rates, and allow r_k to evolve by $r_k = r_{k-1} + \sum_i a_i(\Omega_{k-1})$, where $a_i(\Omega_{k-1})$ denotes the response of agent i at time k to global information set Ω_{k-1} (we assume the aggregate actions keep $r_k > 0$ if r_k represents a price or exchange rate series). For the Minority Game, this response is simply the decision made by agent i, and Ω_{k-1} consists only of the time horizon at time $k-1$. Again, agent i makes this decision by choosing the highest scoring strategy over the time horizon as explained above.

Further, let's define the difference of the rates series as $z_k = r_k - r_{k-1} = \sum_i a_i(\Omega_{k-1})$. The difference series is the series we will estimate throughout this paper. It is trivial to find the rates series given the difference series.

Figure 2.1 visually shows the details of the Minority Game. There are a number of extensions to the Minority Game (e.g. [22, 3, 7, 16]). A wealth of resources can also be found at [1].

2.1.3 Extension to an Artificial Market Model

If we choose a real rate series, and find the difference series as $z_k = r_k - r_{k-1}$, then we can define the winning decisions and the time horizon in terms of the real series. The winning decision at time step k, w_k can be defined by w_k = -sgn(z_k), where sgn(k) represents the sign function, which is 1 for positive k, -1 for negative k, and 0 otherwise. This simply states that when z_k is positive, the minority chose -1 and vice versa. Note that z_k will never be 0 since we toss a fair coin for ties. As before, the time horizon is defined by the winning decisions. At time step k, this would simply be $(w_{k-T}, \ldots, w_{k-1})$. Figure 2.2 shows the extension to an artificial market model.

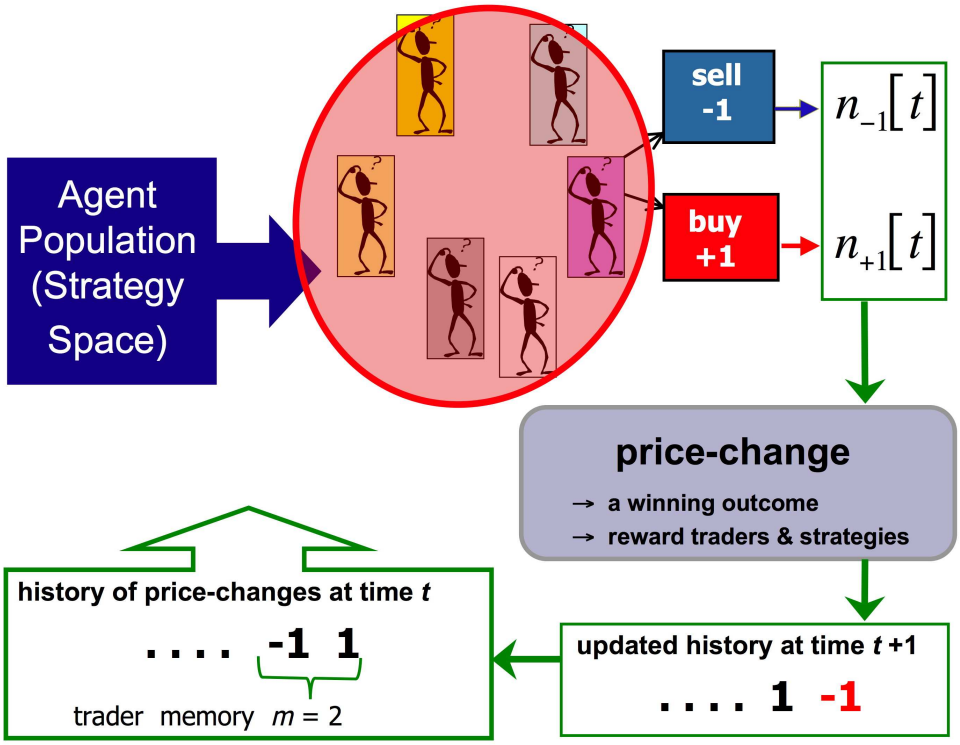

Figure 2.1: In the bottom left corner, we have a time history which is fed to the agents, i.e., $(w_{k-T}, \ldots, w_{k-1})$. The agents then choose their winning strategy from which they make a decision (in this picture they have a memory size of 2). In the top right corner, we count the number of buy and sell decisions and choose the minority as the winning decision which is then updated into the time history. We can also re-score the agent's strategies based on the winning decision.

2.2 Choosing Parameters

Now that we have shown how a multi-agent artificial market model might make predictions, we would like to estimate the heterogeneity of agents playing the game. There are many parameters in the Minority Game, so we have to choose a way to parameterize the game. We provide here one possible way of parameterizing the game to fit the methodology proposed in this paper. Note that the game need not be a Binary Agent Resource game, but for the purposes of demonstration, we assume it is and we fix m and T for all agents. We also say each agent possesses exactly two strategies. Next, we remove the parameter N specifying the number of agents, by

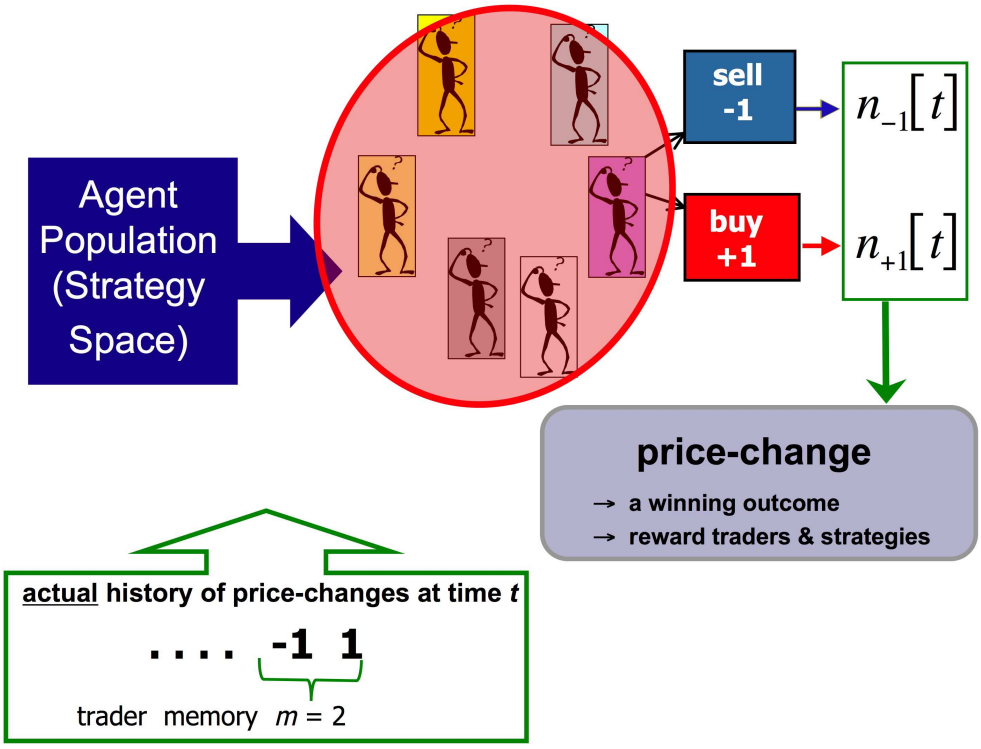

Figure 2.2: This is nearly the same as Figure 2.1 except that rather than calculating the winning decision at each time-step, we simply look at the real time-history. Note that the calculated winning decision need not match the real history in such an artificial market model.

allowing this to tend towards infinity and instead looking at the probability measure over all possible strategy sets with two strategies of memory size m. For example, if $m = 1$, there are $2^{2^1} = 4$ strategies with a memory size of one (there are 2^m possible bit strings of length m and 2 responses to each bit string). So there are $\binom{4}{2} = 6$ distinct pairs of strategies with memory size $m = 1$. The parameter space we would like to estimate is the probability measure over these six possible strategy sets, which describes the composition of heterogeneity. Note that this is a very specific case with six possible strategy set and the method we will describe can be adapted to any number of strategies sets each of which can contain more than two strategies all of various memory sizes. Further, the multi-agent model need not resemble the Minority Game.

This estimation problem is then an inequality constrained problem with the con-

straints that each probability of playing a given strategy set must be greater than or equal to 0 and the probabilities must sum up to 1. For example, if x_k is the vector at time step k representing the heterogeneity over N types of agents, we could write this as

$$x_k = \begin{bmatrix} x_{1,k} \\ \vdots \\ x_{N,k} \end{bmatrix} \tag{2.1}$$

with the following two constraints

$$\sum_i x_{i,k} = 1 \text{ and } x_{i,k} \geq 0, \forall i \tag{2.2}$$

For this problem, we choose to use a recursive optimization scheme similar to a Kalman Filter, but where we also force the equality and inequality constraints on the estimates.

2.3 Recursive Optimization Scheme

In the following subsections, we will introduce the Kalman Filter after which we will be able to discuss some desirable extensions for our applications.

2.3.1 Kalman Filter

A Kalman Filter is a recursive least squares implementation, which makes only one pass through the data such that it can wait for each measurement to come in real time and make an estimate at that time given all the information from the past. In addition, the Kalman Filter holds a minimal amount of information in memory at each time for a cheap computational cost in solving the optimization problem. Using this filter, we can make a forecast n steps ahead with a covariance about this forecast. As it is a predictor-corrector system, it can make predictions and upon observation of real data perturb the predictions slightly as a correction.

The Kalman Filter attempts to find the best estimate at every iteration for a system governed by the following model:

$$x_k = F_{k,k-1}x_{k-1} + u_{k,k-1}, \qquad u_{k,k-1} \sim N(0, Q_{k,k-1}) \tag{2.3}$$

$$z_k = H_k x_k + v_k, \qquad v_k \sim N(0, R_k) \tag{2.4}$$

Here x_k represents the true state of the underlying system, which in our case is the probability measure over the agents. $F_{k,k-1}$ is the matrix used to make the transition from state x_{k-1} to x_k. In our applications, we choose $F_{k,k-1}$ to be the identity matrix for all k since we assume the probability measure over the strategies over small regions doesn't change very drastically. Modeling a way to perturb the measure at each time step would be quite a difficult task, however, if desired we could incorporate any other choice of transition matrix into this model, even one that is dependent on previous outcomes. The variable z_k represents the measurement (or observation). H_k is the matrix that takes the state into measurement space. For our artificial market model, H_k will be a row vector containing the decisions based on each of the agent's winning strategies. So x_k, the measure over the agents, acts as a weighting on the decisions for each agent, and the inner product $H_k x_k$ can be thought of as a weighted average of the agents' decisions. The variables $u_{k,k-1}$ and v_k are both noise terms which are normally distributed with mean 0 and variances $Q_{k,k-1}$ and R_k, respectively.

The Kalman Filter will at every iteration make a prediction for x_k which we denote by $\hat{x}_{k|k-1}$. We use the notation $k|k-1$ since we will only use measurements provided until time step $k-1$ to make the prediction at time k. We can define the state prediction error $\tilde{x}_{k|k-1}$ as the difference between the true state and the state prediction.

$$\tilde{x}_{k|k-1} = x_k - \hat{x}_{k|k-1} \tag{2.5}$$

In addition, the Kalman Filter will provide a state estimate for x_k given all the measurements provided up to and including time step k. We denote these estimates by $\hat{x}_{k|k}$. We can similarly define the state estimate error by

$$\tilde{x}_{k|k} = x_k - \hat{x}_{k|k} \tag{2.6}$$

Since we assume $u_{k,k-1}$ is normally distributed with mean 0, we make the state prediction simply by using $F_{k,k-1}$ to make the transition. This is given by

$$\hat{x}_{k|k-1} = F_{k,k-1}\hat{x}_{k-1|k-1} \tag{2.7}$$

We can also calculate the associated covariance for the state prediction, which we call the covariance prediction. This is actually just the expectation of the outer product of the state prediction error with itself. This is given by

$$P_{k|k-1} = F_{k,k-1}P_{k-1|k-1}F'_{k,k-1} + Q_{k,k-1} \tag{2.8}$$

Notice that we use the prime notation on a matrix throughout this paper to denote the transpose. Now we can make a prediction on what we expect to see for our measurement, which we call the measurement prediction by

$$\hat{z}_{k|k-1} = H_k \hat{x}_{k|k-1} \tag{2.9}$$

The difference between our true measurement and our measurement prediction is called the measurement residual, which we calculate by

$$\nu_k = z_k - \hat{z}_{k|k-1} \tag{2.10}$$

We can also calculate the associated covariance for the measurement residual, which we call the measurement residual covariance. Since we don't actually need the measurement z_k, and only need its associated covariance R_k, we also call this the predicted measurement residual covariance when we have R_k, but not yet z_k. This is calculated by

$$S_k = H_k P_{k|k-1} H_k' + R_k \tag{2.11}$$

Next, we will calculate the Kalman Gain, which lies at the heart of the Kalman Filter. This essentially tells us how much we prefer our new observed measurement over our state prediction. We calculate this by

$$K_k = P_{k|k-1} H_k' S_k^{-1} \tag{2.12}$$

Using the Kalman Gain and measurement residual, we update the state estimate. If we look carefully at the following equation, we are essentially taking a weighted sum of our state prediction with the Kalman Gain multiplied by the measurement residual. So the Kalman Gain is telling us how much to "weight in" information contained in the new measurement. We calculate the updated state estimate by

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \nu_k \tag{2.13}$$

Finally, we calculate the updated covariance estimate. This is actually just the expectation of the outer product of the state error estimate with itself. Here we will give the most numerically stable form of this equation, as this form prevents loss of symmetry and best preserves positive definiteness

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)' + K_k R_k K_k^T \tag{2.14}$$

The covariance matrices throughout the Kalman Filter give us a way to measure the uncertainty of our state prediction, state estimate, and the measurement residual. Also, notice that the Kalman Filter is recursive, and we require an initial estimate $\hat{x}_{0|0}$ and associated covariance matrix $P_{0|0}$. Here we simply provided the equations of the Kalman Filter without derivation. For a more thorough understanding of the Kalman Filter, there are a number of available resources [5].

As we are estimating a vector in which each element has a non-negative value, we would like to force the Kalman Filter to have some inequality constraints. In the next section, we will introduce a generalization for nonlinear equality constraints followed by an extension to inequality constraints.

2.3.2 Nonlinear Equality Constraints

Let's add to our model given by equations (2.3) and (2.4) the following smooth nonlinear equality constraints

$$e_k(x_k) = 0 \tag{2.15}$$

Notice that our constraints provided in equation (2.2) are actually linear. We present the nonlinear case for further completeness here. We rewrite the problem we would like to solve where we use the superscript c to denote constrained. We should also rephrase the problem we would like to solve now: we are given the last prediction and its covariance, the current measurement and its covariance, and a set of equality constraints and would like to make the current prediction and find its covariance matrix in the equality constrained space.

Let's write the problem we are solving as

$$z_k^c = h_k^c(x_k) + v_k^c, \qquad v_k^c \sim N(0, R_k^c) \tag{2.16}$$

Here z_k^c, h_k^c, and v_k^c are all vectors, each having three distinct parts. The first part represents the prediction for the current time step, the second part is the measurement, and the third part is the equality constraint. z_k^c effectively still represents the measurement, with the prediction treated as a "pseudo-measurement" with its associated covariance.

$$z_k^c = \begin{bmatrix} F_{k,k-1}\hat{x}_{k-1|k-1} \\ z_k \\ 0 \end{bmatrix} \tag{2.17}$$

The matrix h_k^c takes our state into the measurement space as before

$$h_k^c(x_k) = \begin{bmatrix} x_k \\ H_k x_k \\ e_k(x_k) \end{bmatrix} \tag{2.18}$$

Notice that by combining equations (2.5) and (2.6), we can rewrite the state prediction error as

$$\tilde{x}_{k|k-1} = F_{k,k-1}\tilde{x}_{k-1|k-1} + u_{k,k-1} \tag{2.19}$$

Now we define v_k^c as the noise term using equation (2.19), where v_k^c is normally distributed with mean 0 and covariance given by matrix R_k^c.

$$v_k^c = \begin{bmatrix} -F_{k,k-1}\tilde{x}_{k-1|k-1} - u_{k,k-1} \\ v_k \\ 0 \end{bmatrix} \tag{2.20}$$

The block diagonal elements of covariance matrix R_k^c represent the covariance of each element of v_k^c. We define the covariance of the state estimate error at time step k as $P_{k|k}$. Notice also that R_k^c contains no block off-diagonal elements implying no cross-correlations.

$$R_k^c = \begin{bmatrix} F_{k,k-1}P_{k-1|k-1}{F_{k,k-1}}' + Q_{k,k-1} & 0 & 0 \\ 0 & R_k & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.21}$$

This method of expressing our problem can be thought of as a fusion of the state prediction and the new measurement at each iteration under the given equality constraints. Much like when we showed the Kalman Filter, we will simply write the solution here [9, 34].

$$\hat{x}_{k|k,j} = \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} R_k^c & H_{k,j}^c \\ {H_{k,j}^c}' & 0 \end{bmatrix}^+ \begin{bmatrix} z_k^c - h_k^c(\hat{x}_{k|k,j-1}^c) + H_{k,j}^c \hat{x}_{k|k,j-1}^c \\ 0 \end{bmatrix} \tag{2.22}$$

Notice the we use the $^+$ notation on a matrix throughout this paper to denote the pseudo-inverse. In this method we are iterating over a dummy variable j within each time step until we fall within a predetermined convergence bound $\left|\hat{x}_{k|k,j} - \hat{x}_{k|k,j-1}\right| \leq c_k$ or hit a chosen number of maximum iterations. We initialize our first iteration as $\hat{x}_{k|k,0} = \hat{x}_{k-1|k-1}$ and use the final iteration as $\hat{x}_{k|k} = \hat{x}_{k|k,J}$ where J represents the final iteration.

Also, notice that we allowed the equality constraints to be nonlinear. As a result, we define $H_{k,j}^c = \frac{\partial h_k^c}{\partial x_k}(\hat{x}_{k|k,j-1})$ which gives us a local approximation to the direction of h_k^c.

A stronger form for this solution can be found in past work [9, 34], where R_k^c reflects the tightening of the covariance for the state prediction based on the new estimate at each iteration of j. We do not use this form and tighten the covariance matrix within these iterations, since in our form, we can change the number of equality constraints between iterations of j. We will find this useful in the next section. Not tightening the covariance matrix in this way is reflected in a larger covariance matrix for the estimate as well. This covariance matrix is calculated as

$$P_{k|k,j} = -\begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} R_k^c & H_{k,j}^c \\ H_{k,j}^{c\prime} & 0 \end{bmatrix}^+ \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{2.23}$$

Notice that for faster computation times, we need only calculate $P_{k|k,j}$ for the final iteration of j. Further, if our equality constraints are in fact independent of j, we can calculate $H_{k,j}^c$ only once for each k. This would also imply the pseudo-inverse in equations (2.22) and (2.23) can be calculated only once for each k.

This method, while very different from the Kalman Filter presented earlier, provides us with an estimate $\hat{x}_{k|k}$ and a covariance matrix for the estimate $P_{k|k}$ at each time step similar to the Kalman Filter. However, this method allowed us to incorporate equality constraints.

2.3.3 Nonlinear Inequality Constraints

We will now extend the equality constrained problem to an inequality constrained problem. To our system given by equations (2.3), (2.4), and (2.15), we will also add the smooth inequality constraints given by

$$l_k(x_k) \geq 0. \tag{2.24}$$

Our method will be to keep a subset of the inequality constraints active at any time. An active constraint is simply a constraint that we treat as an equality constraint. An inactive constraint we will relax (ignore) when solving our optimization problem. After, solving the problem, we then check if our solution lies in the space given by the inequality constraints. If it doesn't we start from the solution in our previous iteration and move in the direction of the new solution until we hit a set of constraints. For the next iteration, this set of constraints will be the new active constraints.

We formulate the problem in the same way as before keeping equations (2.16), (2.17), (2.20), and (2.21) the same to set up the problem. However, we replace equation (2.18) by

$$h_k^c(x_k) = \begin{bmatrix} x_k \\ H_k x_k \\ e_k(x_k) \\ l_{k,j}^a(x_k) \end{bmatrix} \tag{2.25}$$

$l_{k,j}^a$ represents the set of active inequality constraints. Notice that while we keep equations (2.17), (2.20), and (2.21) the same, these will need to be padded by additional zeros appropriately to match the size of $l_{k,j}^a$. Now we solve the equality constrained problem consisting of the equality constraints and the active inequality constraints (which we treat as equality constraints) using equations (2.22) and (2.23). Let's call the solution from equation (2.22) $\hat{x}_{k|k,j}^*$ since we have not checked if this solution lies in the inequality constrained space yet. In order to check this, we find the vector that we moved along to reach $\hat{x}_{k|k,j}^*$. This is simply

$$d = \hat{x}_{k|k,j}^* - \hat{x}_{k|k,j-1} \tag{2.26}$$

We now iterate through each of our inequality constraints to check if they are satisfied. If they are all satisfied, we choose $t_{\max} = 1$, and if they are not, we choose the largest value of $t_{\max}$ such that $\hat{x}_{k|k,j-1} + t_{\max} d$ lies in the inequality constrained space. We choose our estimate to be

$$\hat{x}_{k|k,j} = \hat{x}_{k|k,j-1} + t_{\max} d \tag{2.27}$$

We also would like to remember the inequality constraints which are being touched in this new solution. These constraints will now become active for the next iteration and lie in $l_{k,j+1}^a$. Note that $l_{k,0}^a = l_{k-1,J}^a$, where J represents the final iteration of a given time step.

Note also that we do not perturb the error covariance matrix from equation (2.23) in any way. Under the assumption that our model is a well-matched model for the data, enforcing inequality constraints (as dictated by the model) should only make our estimate better. Having a slightly larger covariance matrix is better than having an overly optimistic one based on a bad choice for the perturbation [30].

In Figure 2.3, we provide a schematic diagram showing how this optimization scheme fits into the multi-agent game for making predictions. In most tracking applications, the transition matrix $F_{k,k-1}$ drives the system evolution through time and the measurement matrix H_k describes a fixed coordinate transform. Interestingly, here H_k changes significantly. In fact, the process noise $Q_{k,k-1}$ and measurement noise R_k combined with H_k are really what drive the system evolution through time.

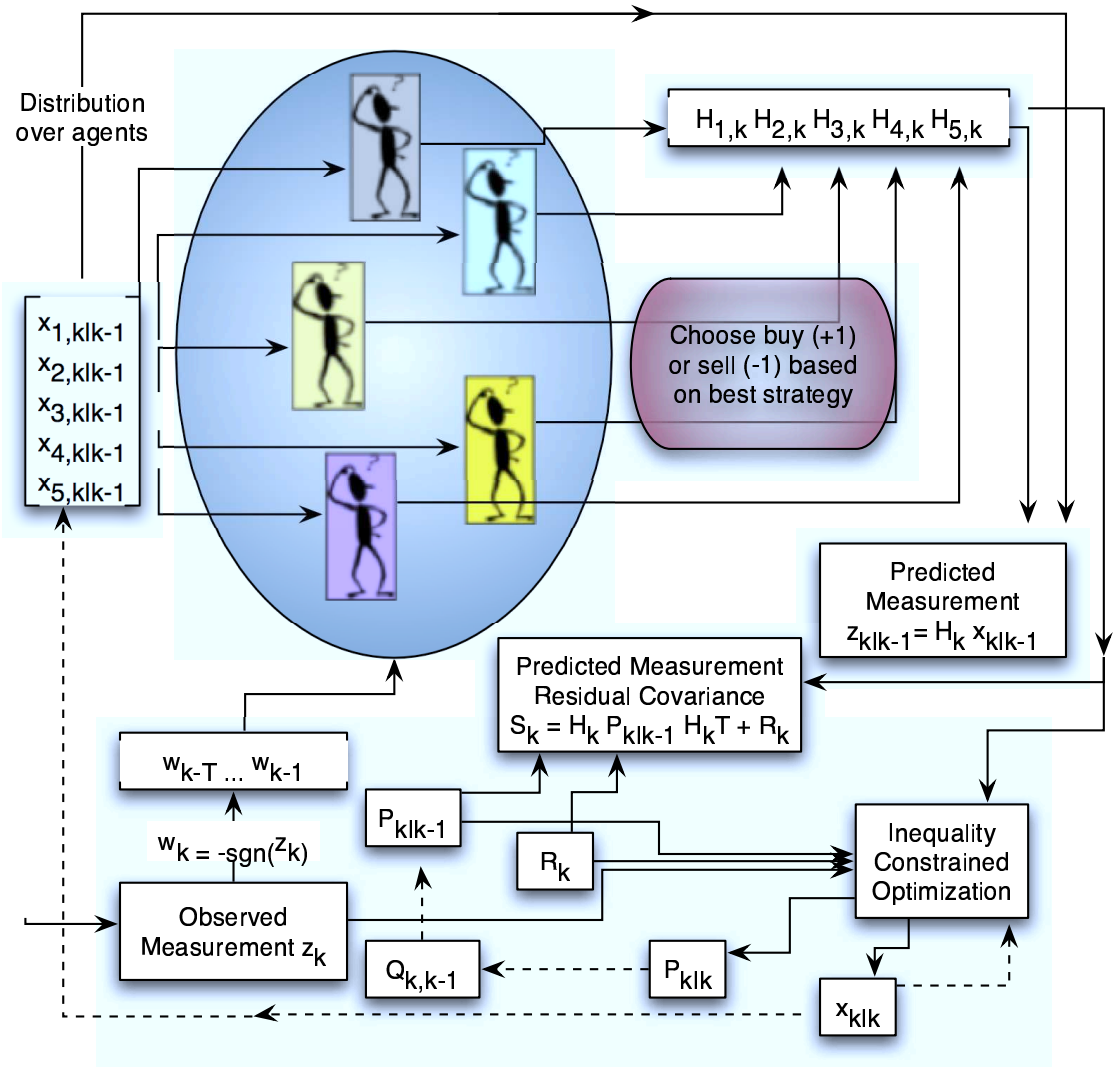

Figure 2.3: We summarize the main idea of the recursive method we describe for predicting the heterogeneity over agents playing (where we drop the ˆ notation). We describe a situation with 5 types of agents, where each type has more than one strategy. They each score their strategies over the sliding time horizon window $(w_{k-T} \cdots w_{k-i})$ and choose the best one. H_k represents the decisions they each then make in this time step, which in the case of a Binary Agent Resource Game is +1 or -1. Taking the dot product of the estimate for the heterogeneity over the agents and their decisions, we arrive at our prediction for the measurement. We then allow the recursion into the optimization technique. Notice that since we choose $F_{k,k-1}$ as the identity matrix for all k, we have simply omitted it in this diagram, and we also assume initial conditions are provided. In Section 2.4, we describe how we arrive at the noise parameters $Q_{k,k-1}$ and R_k, which appear in the diagram.

2.4 Covariance Matching Techniques

In many applications of Kalman Filtering, the process noise $Q_{k,k-1}$ and measurement noise R_k are known. However, in our application we are not provided with this information a priori so we would like to estimate it. These can often times be difficult to approximate especially when there is a known model mismatch. We will present one possible method to approximate these [23]. We choose to match the process noise and measurement noise to the past measurement residual process.

2.4.1 Determining the Process Noise and Measurement Noise

In addition to estimating $Q_{k,k-1}$ and R_k, we estimate the measurement residual covariance S_k. We can actually determine the measurement residual covariance from equation (2.11), but estimating it using covariance matching to the past measurement residual process could be considered more accurate.

We estimate S_k by taking a window of size W_k (which is picked in advance for statistical smoothing) and time-averaging the measurement residual covariance based on the measurement residual process. This is simply the average of all the outer products of the measurement residuals over this window.

$$\hat{S}_k^* = \frac{1}{W_k - 1} \sum_{j=k-W_k}^{k-1} \nu_j {\nu_j}' \tag{2.28}$$

Next, let's estimate R_k. This is done similarly. If we refer back to equation (2.11), we can simply calculate this by

$$\hat{R}_k^* = \frac{1}{W_k - 1} \sum_{j=k-W_k}^{k-1} \nu_j {\nu_j}' - H_j P_{j|j-1} H_j' \tag{2.29}$$

We use our choice of R_k along with our measurement residual covariance S_k to estimate $Q_{k,k-1}$. Combining equations (2.8) and (2.11) we have

$$S_k = H_k(F_{k,k-1} P_{k-1|k-1} {F_{k,k-1}}' + Q_{k,k-1}) H_k + R_k \tag{2.30}$$

Bringing all $Q_{k,k-1}$ terms to one side leaves us with

$$H_k Q_{k,k-1} {H_k}' = S_k - H_k F_k P_{k-1|k-1} {F_k}' {H_k}' - R_k \tag{2.31}$$

And solving for $Q_{k,k-1}$ gives us

$$\hat{Q}^*_{k,k-1} = (H_k{}'H_k)^+ H_k{}' \left(S_k - H_k F_k P_{k-1|k-1} F_k{}' H_k{}' - R_k\right) H_k (H_k{}'H_k)^+ \tag{2.32}$$

It may be desirable to keep $\hat{Q}^*_{k,k-1}$ diagonal if we do not believe the process noise has any cross-correlation (having cross-correlation is generally quite rare). In addition, keeping the process noise diagonal has the effect of making our covariance matrix "more positive definite." This can be done simply by setting the off diagonal terms of $\hat{Q}^*_{k,k-1}$ equal to 0.

It is also important to keep in mind that we are estimating covariance matrices here which must be symmetric and positive semidefinite (note that the diagonal elements should always be greater than or equal to zero as these are variances).

2.4.2 Upper and Lower Bounds for Covariance Matrices

We might also like to denote a minimum and maximum number we are willing to accept for each element of our covariance matrices. The motivation for maintaining a minimum is that we may not want to become overly optimistic. If the covariances drop to zero, we will assume the random variable is perfectly known. The reason for maintaining a maximum is in case we believe the covariance actually is upper bounded. Let us denote these matrices by $S_k^{\min}$, $R_k^{\min}$, $Q_k^{\min}$, $S_k^{\max}$, $R_k^{\max}$, and $Q_k^{\max}$. We apply these by

$$\hat{S}_k = \frac{1}{W_k - 1} \sum_{j=k-W_k}^{k-1} \min\left(S_j^{\max}, \max\left(S_j^{\min}, \nu_j \nu_j{}'\right)\right), \tag{2.33}$$

$$\hat{R}_k = \frac{1}{W_k - 1} \sum_{j=k-W_k}^{k-1} \min\left(R_j^{\max}, \max\left(R_j^{\min}, \nu_j \nu_j{}' - H_j P_{j|j-1} H_j'\right)\right), \tag{2.34}$$

and, using equation (2.32),

$$\hat{Q}_k = \min\left(Q_k^{\max}, \max\left(Q_k^{\min}, \hat{Q}_k^*\right)\right) \tag{2.35}$$

Keep in mind that the diagonal elements of $S_k^{\min}$, $R_k^{\min}$, and $Q_k^{\min}$ must all be greater than or equal to zero. This is a very simple way of lower and upper bounding these matrices. We could imagine a number of ways to approach this problem some of which might be much better at preserving the original information. Our hope for the application mentioned in this paper is that the bounds are rarely touched if ever.

2.4.3 Choosing Bounds for a Probability Measure

For the minimum and maximum acceptable covariances used in the covariance matching scheme for a probability measure, we choose

$$S_k^{\min} = R_k^{\min} = 0 \text{ and } S_k^{\max} = R_k^{\max} = 1 \tag{2.36}$$

Note that both the measurements and the measurement residuals must lie in $[-1, 1]$. This is true because in the extreme situations, all the strategies can choose either -1 or 1. It is also known that the variance of the distribution with half of its weight at a and the other half at b is given by $\frac{(b-a)^2}{4}$. Applying this formula gives us the maximum acceptable variances of 1 in equation (2.36). We use a similar idea to choose the minimum and maximum acceptable process noise.

$$Q_k^{\min} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{2.37}$$

and

$$Q_k^{\max} = \begin{bmatrix} .25 & 0 & \cdots & 0 \\ 0 & .25 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & .25 \end{bmatrix} \tag{2.38}$$

We choose .25 for the diagonal terms of Q_k^{max} by our previous logic since each element of the probability measure must lie in $[0, 1]$. We force the diagonal elements to be 0 in order to keep our covariances more positive definite.

In addition, we state the following definition

$$\text{cov}(x, y) = \text{cor}(x, y)\sigma_x\sigma_y \tag{2.39}$$

Noticing that the $\text{cor}(x, y)$ takes its most extreme values at ± 1 and σ_x and σ_y both take their largest values at $\frac{b-a}{2}$, we can state

$$|\text{cov}(x, y)| \leq \left(\frac{b-a}{2}\right)^2 \tag{2.40}$$

We may also be interested in bounding our state prediction error and state estimate error covariances since we know we are estimating a probability measure. Using equation (2.40) for the off diagonal terms, we can bound these by

$$P_{k|k-1}^{\min} = P_{k|k}^{\min} = \begin{bmatrix} 0 & -.25 & \cdots & -.25 \\ -.25 & 0 & \cdots & -.25 \\ \vdots & \vdots & \ddots & \vdots \\ -.25 & -.25 & \cdots & 0 \end{bmatrix} \tag{2.41}$$

and

$$P_{k|k-1}^{\max} = P_{k|k}^{\max} = \begin{bmatrix} .25 & .25 & \cdots & .25 \\ .25 & .25 & \cdots & .25 \\ \vdots & \vdots & \ddots & \vdots \\ .25 & .25 & \cdots & .25 \end{bmatrix} \tag{2.42}$$

2.5 Application to a Simulated Minority Game

In this section, we will apply the discussed methods to a simulation of the Minority Game. In the next section, we will apply these methods to real financial data.

2.5.1 Generating Simulation Data

We choose parameters $m = 1$ for the memory size and allow two strategies per agent resulting in six overall combinations as described in Section 2.2. We also choose the time horizon window length to be $T = 50$. We randomly choose a bit string of length 50 to serve as the initial time horizon, and we also randomly choose a probability measure over the six possible strategy sets. We run the simulation over 150 time steps. This results in a rate series r_k from which we can extract the difference series z_k.

2.5.2 Forming the Estimation Problem

To track the difference series z_k, we set the problem up similar to how it was generated with $m = 1$ giving six possible strategy sets to estimate a probability measure over, and we choose the time horizon window length to be $T = 50$. For the covariance matching, we choose W_k in equations (2.33) and (2.34) to be equal to $T = 50$. Notice that while we have less than 50 measurement residuals, we choose W_k to be the number of measurement residuals we have. When we have 0 measurement residuals (at the initial point), we choose $R_k = 0$ so we can heavily trust the first measurement to strengthen our initialization.

We choose our initial state $\hat{x}_{0|0} = \frac{1}{s}1_s$, where s is the number of strategy sets (in our case 6) and 1_s is a column vector of size s full of 1's, i.e., a uniform distribution. We

also choose our initial covariance $P_{0|0} = .25I_{s\times s}$, where $I_{s\times s}$ represents the $s\times s$ identity matrix (.25 for the same reason as in Equation (2.38)). We start the optimization problem at time step $T+1$ since we use the first T data points to initialize the time horizon. We also assign no process noise initially and zero measurement noise on the first measurement.

Using the methods described thus far, we can make predictions on this system. After making predictions, we need to decide which predictions to accept with greater certainty, which we discuss next.

2.5.3 Effective Forecasting

When is the forecast produced by this method good and how good? We could base this on the predicted measurement residual covariance S_k which is a prediction of the covariance of the measurement residual process. Note that we can either use equation (2.11) along with equation (2.8) or we can use equation (2.28) to calculate S_k. The residual-based estimate given by equation (2.28) will generally provide a smoother function through k which might be desirable to find pockets of predictability.

There are various ways of using S_k to decide when we would like to make a prediction. We choose to implement a very simple method where we take a threshold value t_k such that we place k in our set of prediction time-steps if $S_k \leq t_k$.

2.5.4 Results of simulation

We show the results from the simulated data in Figure 2.4. We choose the threshold value t_k in this plot to be 10^{-3} for all k. Notice that in our case, t_k is a scalar since the measurements are scalars. As we can see in the plot, we are able to make good forecasts at over 30 points (where our measurement residuals lie within the measurement residual standard deviation). We attempt to make forecasts at the last 100 points of the 150 simulated data points (we use the first 50 points to generate the initial time horizon), and we choose to make a predictions at only 34 of these points. It happens to be that these 34 points are the first 34 that we attempt to forecast. The reason for this is that after we have 2 bad predictions (at the end of the plot), we never recover to a good prediction since the covariance matching scheme drives the estimated process noise $\hat{Q}_{k,k-1}$, estimated measurement noise $\hat{R}_k$, and estimated measurement residual covariance $\hat{S}_k$ up due to the large spike in the single residual which affects the statistical smoothing for 50 time steps. Since the covariance of the state remains tight and the covariance of the measurements is relatively large, new

measurements aren't trusted and given much weight for creating forecasts. In Section 2.8, we provide a method to make the covariance matrix less prone to error.

At the same time, because of the transient by the statistical smoothing, we continue to make forecasts immediately after the first false prediction. We might choose to incorporate a scheme to not make predictions for some length of time immediately after a false prediction to allow $\hat{Q}_{k,k-1}$, $\hat{R}_k$, and $\hat{S}_k$ to respond to the shock caused by the false prediction. Further, we might like to significantly increase the process noise after a false prediction to effectively cause new measurements to have a stronger weight in forming estimates.

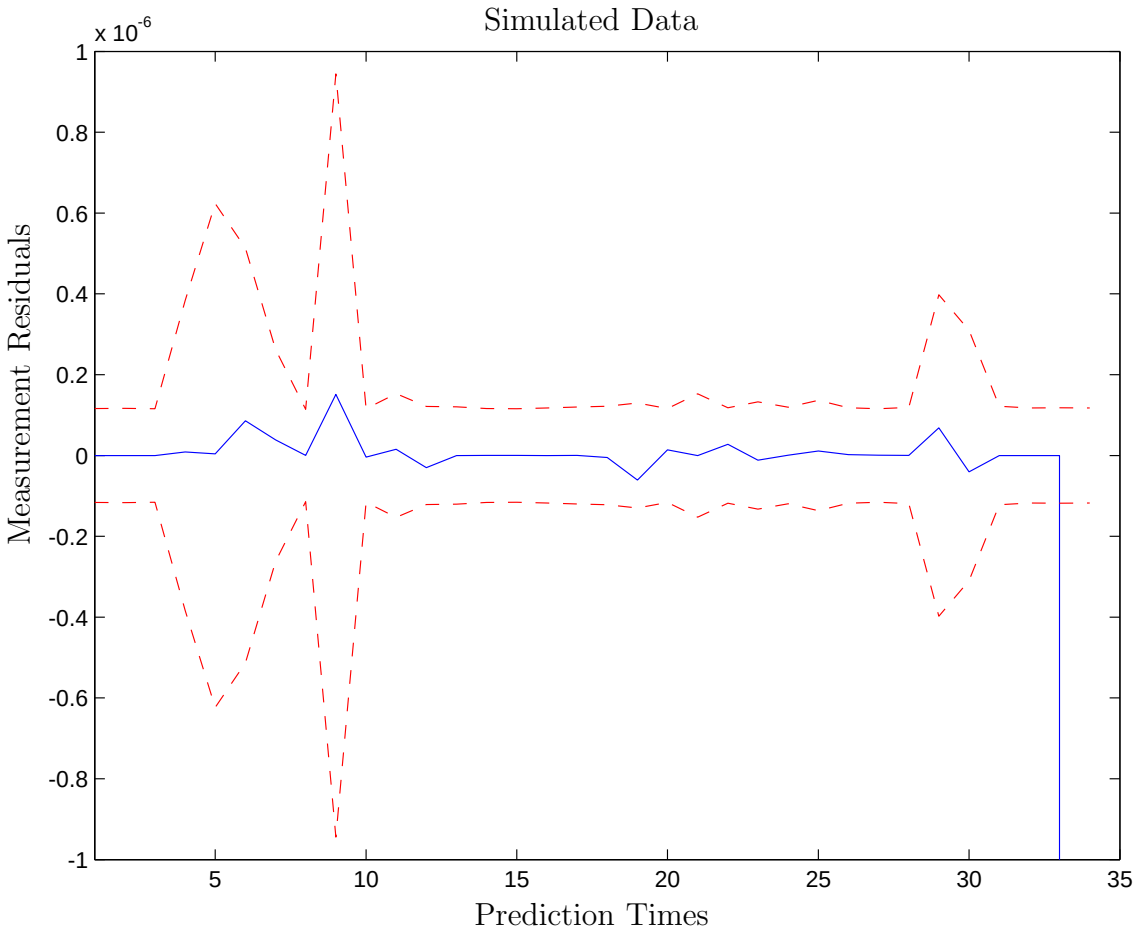

Figure 2.4: The solid line represents the measurement residual process and the dashed line represents one standard deviation about zero based on the predicted measurement residual variance. We select "Prediction Times" as times when the data is in a predictable state based on the measurement residual covariance. These times need not be consecutive in the original data although often times are.

2.6 Application to a Real Foreign Exchange Series

We now apply the ideas in this paper to real financial data. We choose hourly USD/YEN foreign exchange rate data from 1993 to 1994 provided by Dr. J. James of Bank One in London.

2.6.1 Scaling the difference series

First we find the difference series from the rate series as before. For the time being, let's call this $z_k^* = r_k - r_{k-1}$. Since our algorithm only makes forecasts in $[-1, 1]$, we might like to scale our inputs to this domain as best as possible. Assuming that we have measurements a priori up to time step K, we estimate the scaling based on these measurements. Let's denote the set of all measurements up to time K by z_K^* where z_K^* in our case is a vector of scalar measurements. We choose the following method:

Let's denote the minimum and maximum elements of vector V by the functions $\min(V)$ and $\max(V)$, respectively. And let's denote the minimum and maximum elements of z_K^* by $z_K^{\min}$ and $z_K^{\max}$, respectively. We first proportionally scale the spacing between elements of z_k^* so the difference between the minimum element and the maximum element is 2 (the size of $[-1, 1]$). We denote this by z_k^{**}

$$z_k^{**} = \frac{2z_k^*}{z_K^{\max} - z_K^{\min}} \tag{2.43}$$

Next, we scale the elements so the minimum element is at -1. This will automatically place the maximum element at $+1$.

$$z_k = z_k^{**} - (\min(z_k^{**}) + 1) \tag{2.44}$$

We do the calculation for $z_K^{\min}$ and $z_K^{\max}$ once with all of the a priori information we have. Then we can use equations (2.43) and (2.44) to scale for any time step k. Our hope is that based on the a priori information, our choice of $z_K^{\min}$ and $z_K^{\max}$ will reflect the true spacing. If we know true values for these, we can use them instead. Notice also that we can still find the rate series r_k from this definition for the measurements simply by inverting the process. There will certainly be a number of different ways to do this scaling as well.

2.6.2 Forming the Estimation Problem

For the rest of this estimation problem, we actually do the setup exactly the same as in Section 2.5.2 and we follow the effective forecasting scheme exactly as in Section 2.5.3 choosing threshold value t_k to be 10^{-3} again for all k.

2.6.3 Results on the Real Data

We show the results from the real data in Figure 2.5. Here we had over 4000 data points. We chose to make predictions at about 100 data points of which over 90 we accept. These are again the first points we attempt to make a forecast on. And again we see some false predictions near the end. Incorporating a scheme to not make predictions immediately after a false prediction as mentioned in Section 2.5.4 would leave us with only 1 false prediction and over 90 good predictions.

2.7 Additional Algorithm Tests

We do some Monte Carlo runs and discuss two additional tests to check the validity of our algorithm. The first is a test to check the inner workings of the algorithm, while the second checks the functional results that we are interested in.

2.7.1 State Estimate Errors

As a further validation to the method described in this paper, we would like to check that the state estimate errors (2.6) are tending towards a zero mean process.

We set up the problem exactly as in Section 2.5 and do 400 Monte Carlo runs where our Monte Carlo space is the initial time horizon (2^{50} possibilities) and our initial estimate used for generating the simulation (infinite possibilities) as described in Section 2.5.1. For each run, we pick the time horizon uniformly at random and we pick the initial distribution by taking a random vector with each element chosen uniformly at random from [0,1] and then normalizing the vector.

In order to determine the state estimate errors, we take as the true state the expectation of our true state $\frac{1}{6}1_6$ which is also our educated guess for our initial state. Notice that the red line in Figure 2.6 starts out initially with a very tight covariance, which is a result of choosing our initial state as the expectation of the true state (but not the true state).

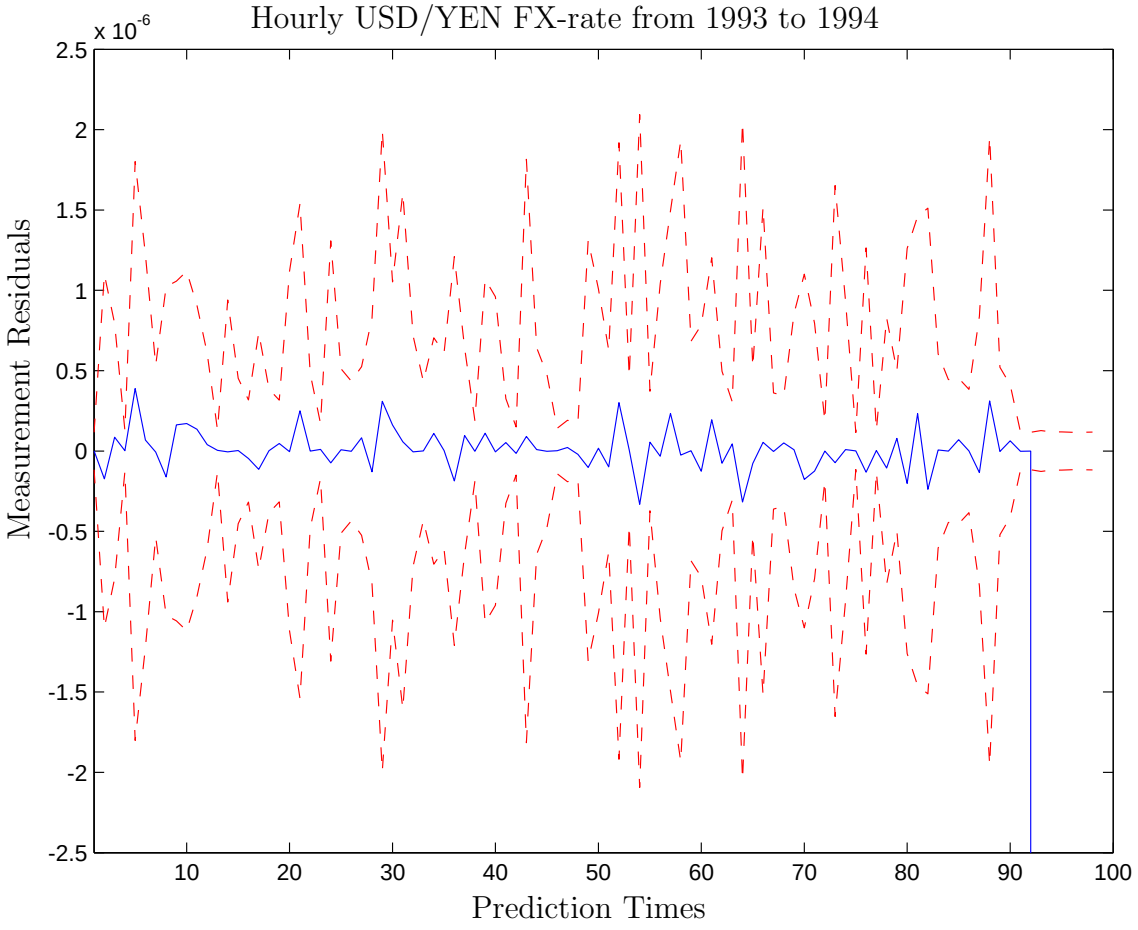

Figure 2.5: The solid line represents the measurement residual process and the dashed line represents one standard deviation about zero based on the predicted measurement residual variance. We select "Prediction Times" as times when the data is in a predictable state based on the measurement residual covariance. These times need not be consecutive in the original data although often times are.

2.7.2 Measurement Residuals Process

The other important test would be to ensure that our forecast method would give measurement residuals that have mean zero also. This is a given based on Figure 2.6. We show this result in Figure 2.7. Notice that the parabolic nature of the green and black lines here is due to the fact that we take our first measurement to have no noise.

Finally, in Figure 2.8, we show one more plot in which at each time step, we keep only those points which have $S_k \leq t_k = 10^{-3}$. In any given time step, at most 3 points of the 400 were removed. These would correspond to difficult to estimate problems

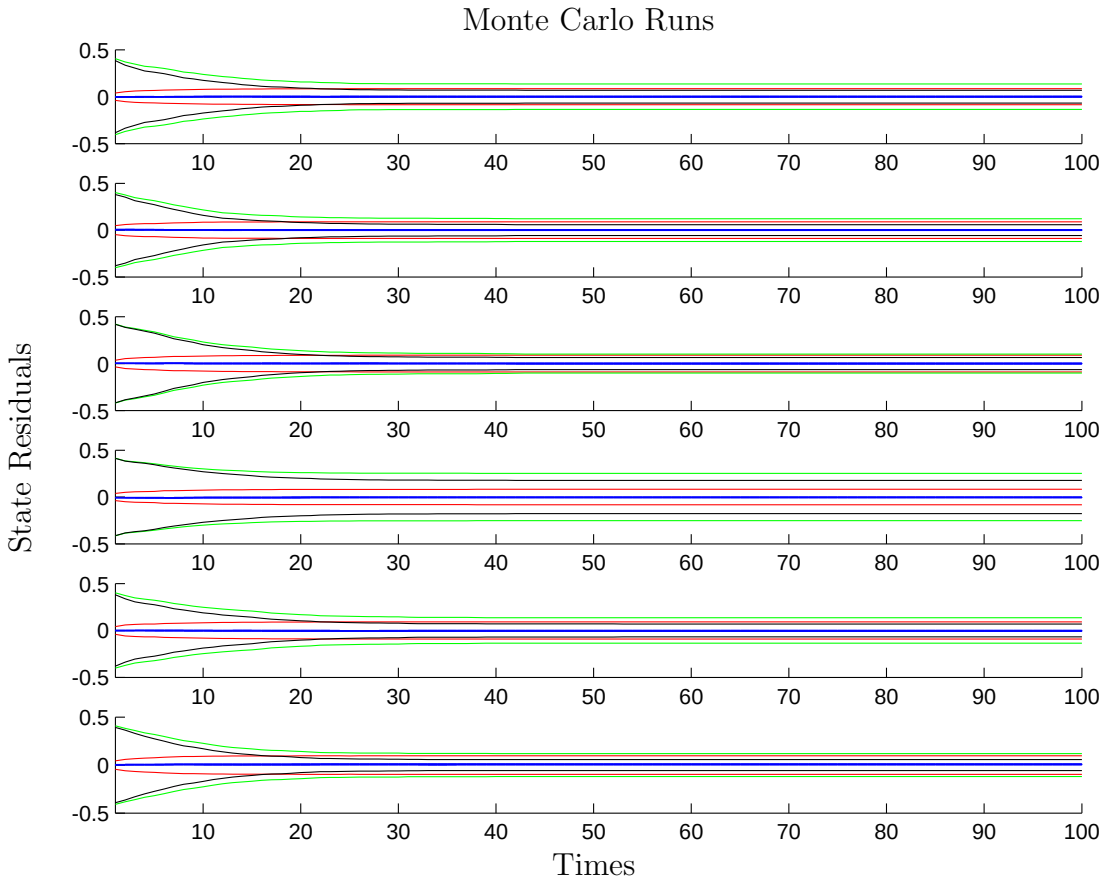

Figure 2.6: In each of the above plots, the blue line indicates the mean measurement residual process with the error bars indicating the standard error of the mean as given over 400 runs. The red line indicates the calculated standard deviation based on the sample statistics centered around zero. The green line indicates the mean standard deviation calculated as the square root of the mean variance over the 400 runs. The black line indicated the mean standard deviation calculated as the mean of the standard deviations over the 400 runs.

(such as the example we chose for Figure 2.4). We notice that removing these points tightens our covariances drastically as hoped.

2.8 Removing the Probability Sum Constraint

We can imagine also allowing the estimate to move out of the probability space but still allowing all elements of our vector x_k to remain non-negative. If we relax this further condition, we find the benefits of lying in the probability space are outweighed by those of allowing the estimate to move out of this space. A large benefit of staying

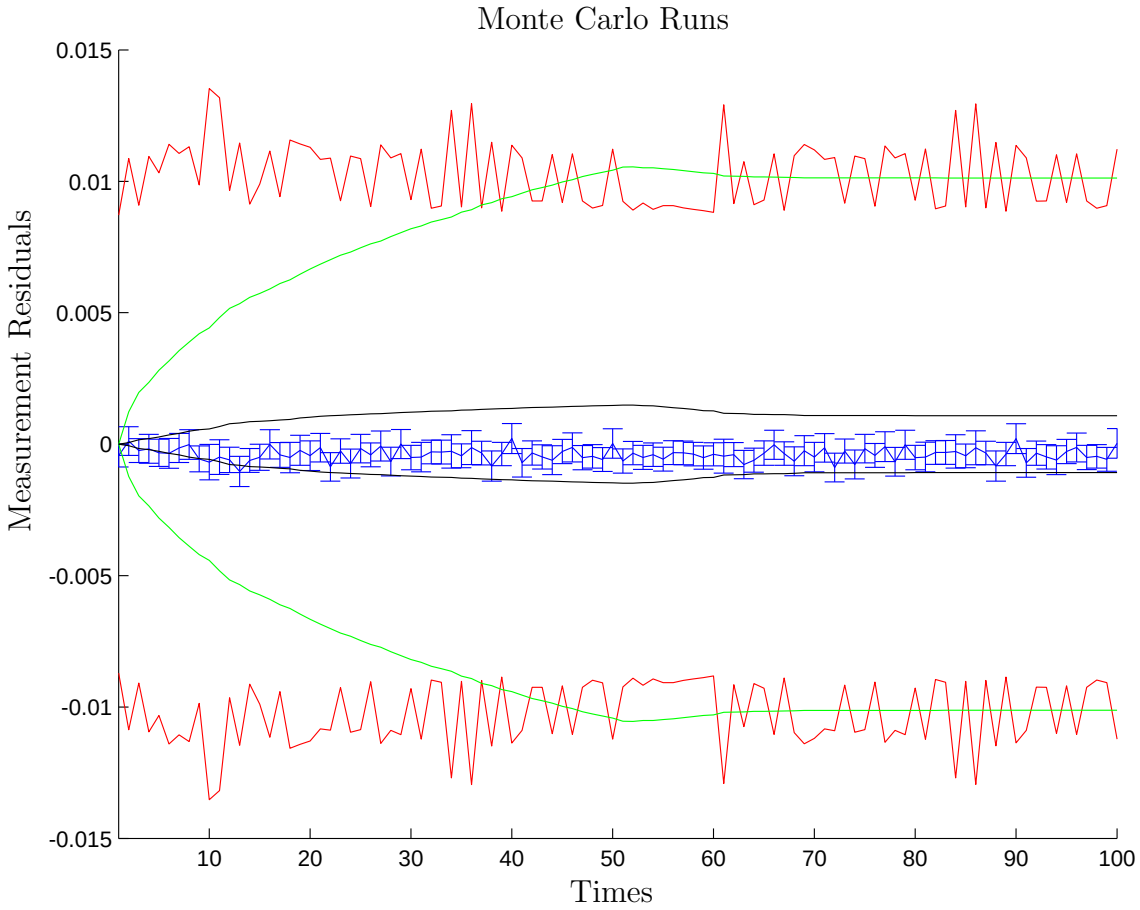

Figure 2.7: The blue line indicates the mean measurement residual process with the error bars indicating the standard error of the mean as given over 400 runs. The red line indicates the calculated standard deviation based on the sample statistics centered around zero. The green line indicates the mean standard deviation calculated as the square root of the mean variance over the 400 runs. The black line indicated the mean standard deviation calculated as the mean of the standard deviations over the 400 runs.

in the probability space is the ability to bound covariance matrices (as upper bounds on the probability of certain events are known). However, this benefit is mainly interesting during the start-up phase of the prediction algorithm. Over time the covariance usually tightens around the estimates where we would expect to make a prediction.

On the other hand, staying constrained to a probability space removes one degree of freedom from our system (i.e., $y_n = 1 - y_1 - \cdots - y_{n-1}$). This can cause the covariance of our estimate to also become ill-formed at times and cause numerical

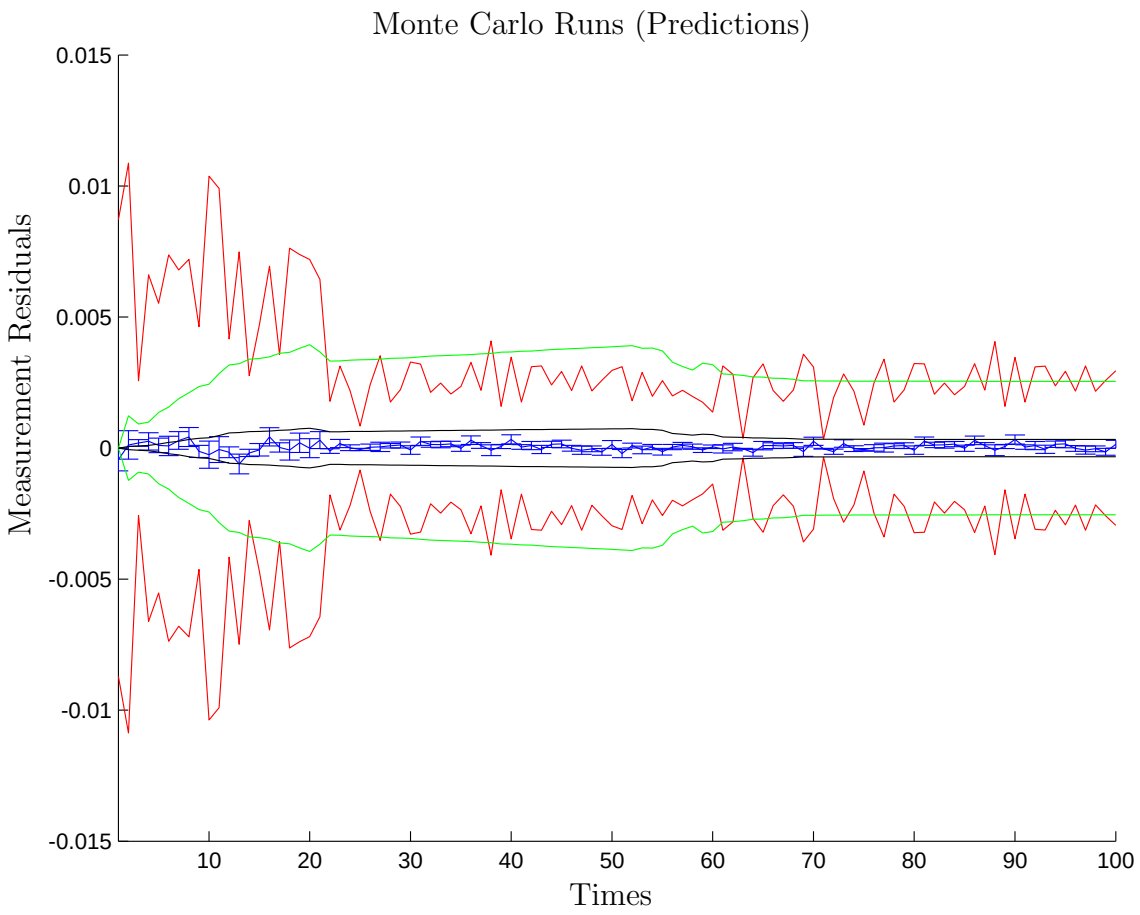

Figure 2.8: The blue line indicates the mean measurement residual process with the error bars indicating the standard error of the mean. The red line indicates the calculated standard deviation based on the sample statistics centered around zero. The green line indicates the mean standard deviation calculated as the square root of the mean variance. The black line indicated the mean standard deviation calculated as the mean of the standard deviations.

loss of positive definiteness (Figure 2.9). In general, we artificially disallow this to happen by perturbing our matrix by a fixed amount on the diagonal elements if we see this phenomenon. However, this is a lesser problem if we choose to estimate a finite measure that is not constrained to unit total weight (Figure 2.10). In addition, we no longer worry about scaling the time-series so the possible predictions lie in a range similar to the financial movements we encounter. With a probability measure, the predicted financial movements can lie in the range $[-1, 1]$, where every agent makes the same decision. If we move away from the probability space, we no longer scale the time-series initially and the financial movements can range from $(-\infty, \infty)$, which

is certainly *more* realistic (Section 2.6.1).

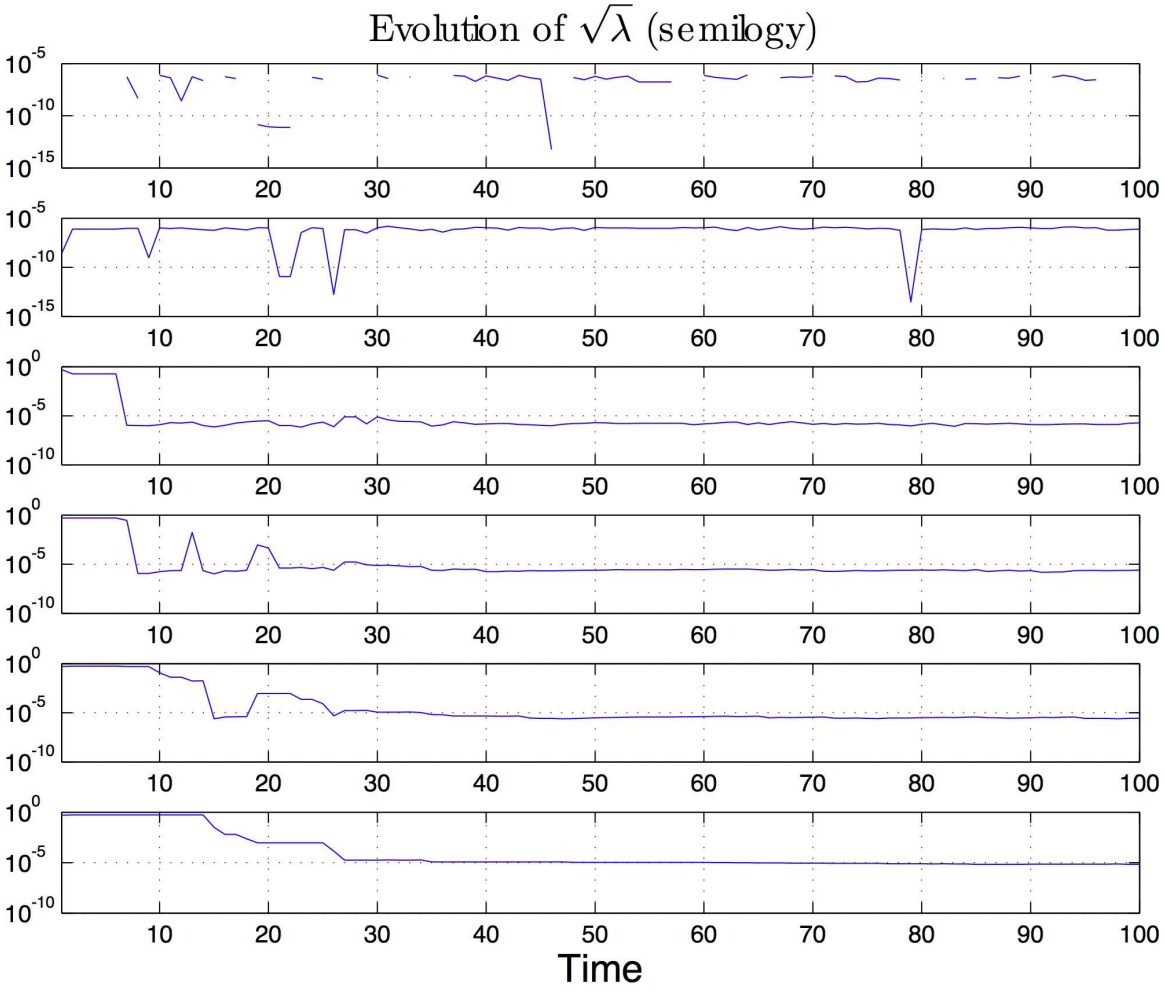

Figure 2.9: The above plots show the square roots of the eigenvalues of the state estimate's covariance matrix for a case where we are estimating a probability measure. These are plotted in ascending order on a semilogy scale and were taken over a simulation of 100 points. We see that due to the probability constraint, we lose a degree of freedom, which is reflected in one of the eigenvalues often times going to zero, but only one of the eigenvalues exhibits this behavior.

2.9 Estimation with Many Possible Types of Agents

It is very likely that we will come across artificial market models we would like to use that allow many different Agent Types, e.g., $N > 100$. As N grows, not only does our state space grow linearly, but our covariance space will grow quadratically. We quickly reach areas where we may no longer be in a computationally feasible region.

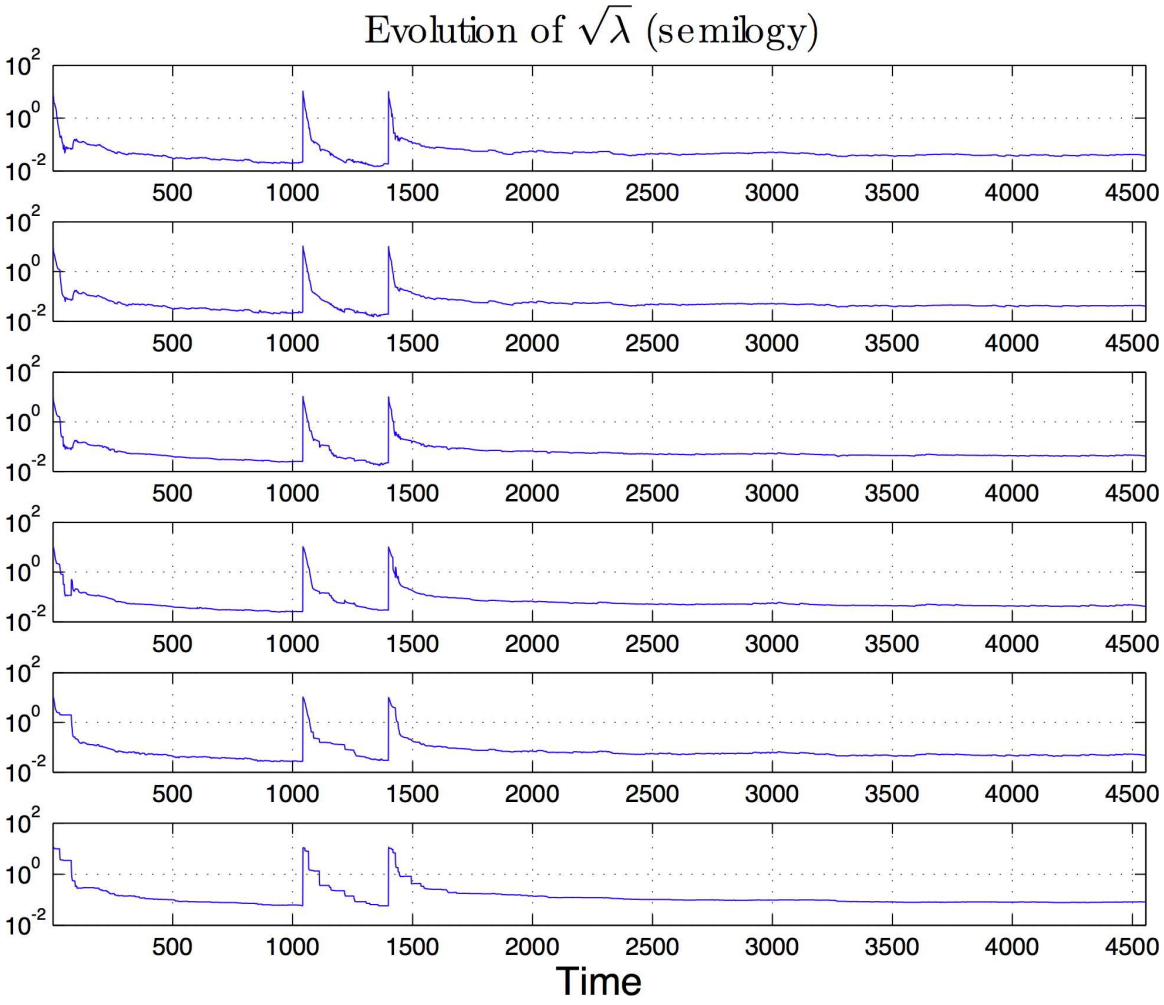

Figure 2.10: The above plots show the square roots of the eigenvalues of the state estimate's covariance matrix where the estimate is not confined to the probability space. These are plotted in ascending order on a semilogy scale and were taken from the real data in Section 2.6. In this case, we see that there is no longer a fear of losing nonsingularity in the matrix.

For example, if we look at strategies for agents playing the Minority Game and define a type of agent to have exactly 2 strategies, as we increase the memory sizes of our agents, our full set of pairs of strategies grows very quickly in relation to the memory size (e.g., $m = 1$ yields $2^{2^1} = 4$ strategies and $\binom{4}{2} = 6$ pairs of strategies, $m = 2$ yields $2^{2^2} = 16$ strategies and $\binom{16}{2} = 120$ pairs of strategies, $m = 3$ yields $2^{2^3} = 256$ strategies and $\binom{256}{2} = 32640$ pairs of strategies, $m = 4$ yields $2^{2^4} = 65536$ strategies and $\binom{65536}{2} = 2147450880$ pairs of strategies, ...). If we were interested in simultaneously allowing all possible pairs of strategies, our vectors and matrices for these computations would have dimension that would not be of reasonable complexity,

especially in situations where real-time computations are needed.

In such situations, we propose selecting a subset of the full set of strategies uniformly at random and choosing these as the only set that could be in play for the time series. We can then do this a number of times and average over the predictions and their covariances. We would hope that this would cause a smoothing of the predictions and remove outlier points. In addition we might notice certain periods that are generally more predictable by doing this, which we call pockets of predictability.

2.9.1 Averaging over Multiple Runs

For each run j of our M runs, we have our predicted measurement at time k given by $\hat{z}_{k,j}$ and our predicted covariance for the measurement residual as $S_{k,j}$. Using the predicted measurements, we can simply average to find our best estimate of the prediction.

$$\hat{z}_k^* = \sum_{j=1}^{M} \frac{\hat{z}_{k,j}}{M} \tag{2.45}$$

And similarly, we can calculate our best estimate of the predicted covariance for the measurement residual.

$$S_k^* = \sum_{j=1}^{M} \frac{S_{k,j}}{M} \tag{2.46}$$

2.9.2 Bias Estimation

Since we are choosing subsets of the full strategy space, we could certainly expect that in some runs, a number of the strategies might tend to behave the same. This doesn't mean that the run is useless and provides no information. In fact, it could be the case that the run provides much information, except that the predictions always tend to be biased in one direction or the other. So what we might like to do is remove bias from the system.

The simplest way to do this is to augment the Kalman Filter's state space with a vector of elements representing possible bias [10]. We can model this bias as lying in the state space, the measurement space, or some combination of elements of either or both. This is all done very simply. We can redefine the model for our problem as

$$x_k^b = F_{k,k-1}^b x_{k-1}^b + u_{k,k-1}, \qquad u_{k,k-1} \sim N(0, Q_{k,k-1}^b) \tag{2.47}$$

$$z_k = H_k^b x_k^b + v_k, \qquad v_k \sim N(0, R_k) \tag{2.48}$$

where x_k^b represents the augmented state and b_k is the bias vector at time step k

$$x_k^b = \begin{bmatrix} x_k \\ b_k \end{bmatrix} \tag{2.49}$$

The transition matrix must also be augmented to match the augmented state. In the top left corner, we place our original transition matrix, and in the top right corner we place $B_{k,k-1}$ representing how the estimated bias term should be added into the dynamics. In the bottom left we have the zero matrix so the bias term is not dependent on the state x_k, and in the bottom right, we have the identity matrix indicating that the bias is updated as itself exactly at each time.

$$F_{k,k-1}^b = \begin{bmatrix} F_{k,k-1} & B_{k,k-1} \\ 0 & I \end{bmatrix} \tag{2.50}$$

Similarly, we horizontally augment our measurement matrix, where C_k represents how the bias terms should be added into the measurement space.

$$H_k^b = \begin{bmatrix} H_k & C_k \end{bmatrix} \tag{2.51}$$

For the process noise, we keep the off diagonal elements as 0, assuming no cross-correlations between the state and the bias. We also generally assume no noise in the bias term and keep its noise covariance as 0, but of course that can be changed easily enough if the reader would like to model the bias with some noise.

$$Q_{k,k-1}^b = \begin{bmatrix} Q_{k,k-1} & 0 \\ 0 & 0 \end{bmatrix} \tag{2.52}$$

We can take this bias model framework and place it into the inequality constrained filtering scheme provided earlier with model given by Equation (2.16), where we simply use the augmented states, when necessary, rather than the regular state space (e.g., let $x_k = x_k^b$).

2.10 Another Look at the Same Foreign Exchange Series

We combine all of these ideas to take a different approach on the same data set used in Section 2.6 for the purposes of demonstration.

For our choice of artificial market model, we continue using the Minority Game, although we still do not claim that this is necessarily a good model for the time-series.

However, we believe it does share some of the characteristics we might expect to see in a real financial time-series. We look at all pairs of strategies with memory size $m = 4$ of which there are 2,147,450,880 as our set of possible types for agents. Since the size of this computation would not be tractable, we take a random subset of 5 of these types and use these 5 as the only possible types of agents to play the game, but we do 100 such runs, each time choosing 5 types at random.

In addition, we allow for a single bias removal term. We could have many more terms for the bias, but we only use one in order to limit the growth of the state space. We assume this entire bias lies in a shifting of the measurements, so we choose $B_{k,k-1} = 0$ in equation (2.50) and C_k to be the identity matrix in equation (2.51) (or in our case simply 1 since C_k is 1x1).

For the analysis of how well our forecasts perform, we calculate the residual log returns and plot these. Given our time-series, we can calculate the log return of the exchange rate as $l_k = \log(r_k) - \log(r_{k-1})$. Note that based on our definition for z_k from Section 2.1.3, we can write the log return also as $l_k = \log(r_{k-1} + z_k) - \log(r_{k-1})$. Similarly, we can define our predicted forecast for the log return as $\hat{l}_k = \log(r_{k-1} + \hat{z}_k) - \log(r_{k-1})$. Given these two quantities, we can calculate the residual of the predicted log return and the observed log return as $\tilde{l}_k = l_k - \hat{l}_k$. Using the delta method [6], we can also calculate the variance of this residual to be $\frac{S_k}{(r_{k-1})^2}$.

We decide to do 100 such runs, which we average over using the method described in Section 2.9.1. A good test for whether our variances are overly optimistic is to check if the measure satisfies the Chebyshev Inequality. For example, we can check visually that no more than about $\frac{1}{9}$ of the residuals lie within 3σ's. This must be true of any probability measure. We plot the residual log return along with 3σ bounds and the standard error of the mean in Figure 2.11. Visually, we would say the residuals in Figure 2.11 would certainly satisfy this condition if they were centered about 0 (using further bias removal should do this).

2.11 Extension to Other Games

Note that we chose the Minority Game throughout this thesis as the game we think best exhibits the dynamics of the financial time-series we analyze. The method for forecasting we describe in this paper can be used with a number of different models. We can easily extend this technique to any multi-agent model where we are interested in estimating the composition of heterogeneous agents.

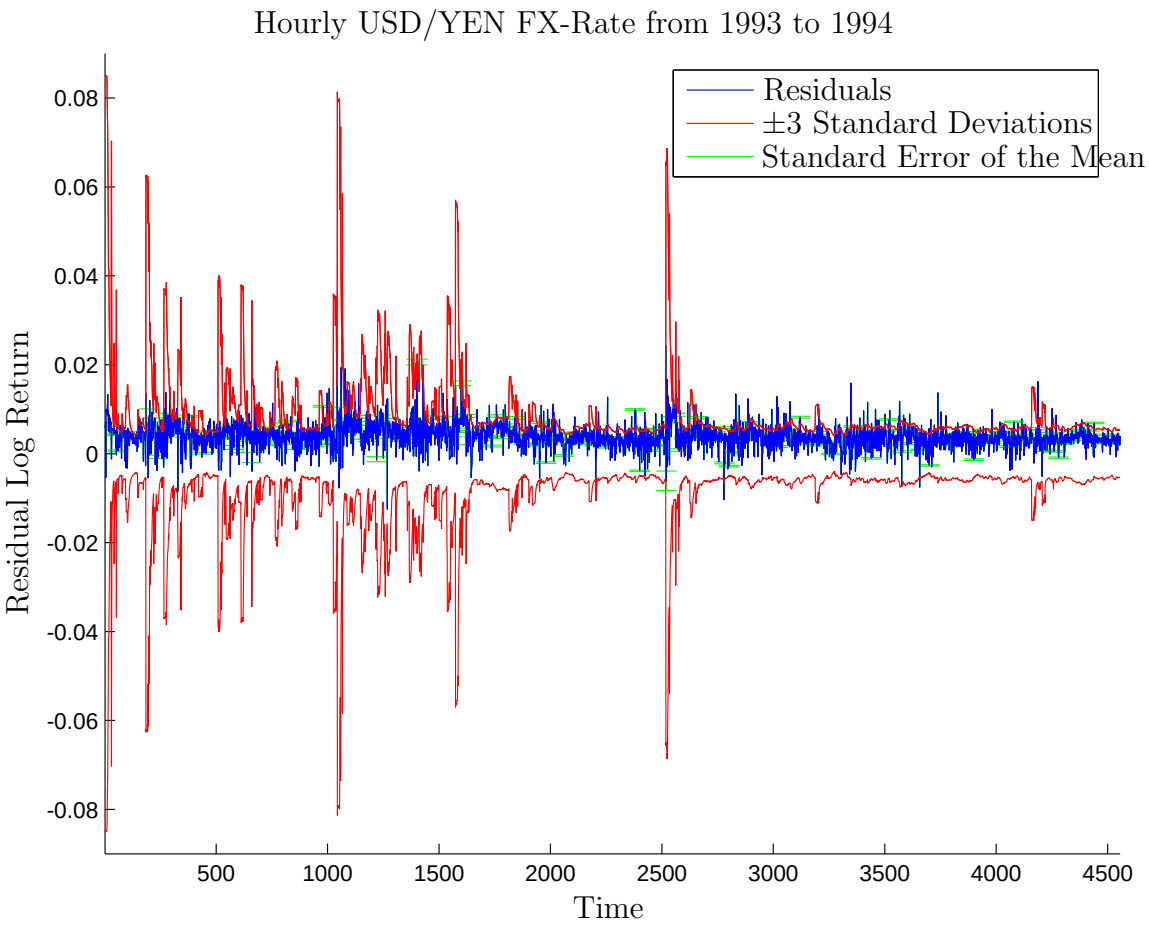

Figure 2.11: We show the residuals of the log returns plotted with 3σ's centered about 0 and the standard error of the mean over the 100 runs. Despite, the one parameter bias removal, we still see a general bias in the data, without which the residuals look much cleaner. Perhaps modeling the bias in higher dimension would remove this.

Chapter 3

Non-Genomic Evolution

3.1 Background and Previous Work

All current life is based on organisms replicating through information coded in their DNA (Deoxyribonucleic acid) or RNA (Ribonucleic acid). DNA and RNA are composed of sequences of nucleic acids, which can form base pairs allowing replication of the sequence. Due to their ability to replicate, models of the origins of life based on DNA or RNA are the most popular models [18]. However, DNA and/or RNA based models have many difficulties [26]. Current organisms require a complex interaction between DNA and RNA to survive. Even hypothetical organisms based only on RNA ("RNA worlds") require complicated interactions between different types of RNA [15, 18]. In addition, the nucleic acid sequences which form DNA and RNA are difficult to make under the pre-biotic conditions that existed early in Earth's history. Therefore, there is a growing interest in non-genomic evolution - the study of organisms that do not use nucleic acids, and instead use only proteins. The non-genomic evolutionary model of primitive organisms has the advantage that the proteins are made of sequences of amino acids, which have been shown to be made under conditions similar to those that existed in Earth's early history. However, since amino acids cannot form base pairs, unlike nucleic acids, proteins cannot be directly replicated (Figure 3.1). Therefore the field of non-genomic evolution focuses on global properties of protein ecologies to see if protocells (precursors to cells which have some properties of cells such as having a membrane, but do not have all cell functions or cell components) formed from these ecologies can produce new protocells [15, 26]. Understanding the combined behavior of the ecology is paramount in order to ascertain the potential place of nongenetic evolution in the history of life on Earth.

While recent laboratory and computation progress has been made in understanding self-replicating protein ecologies, we are still far from being able to precisely model

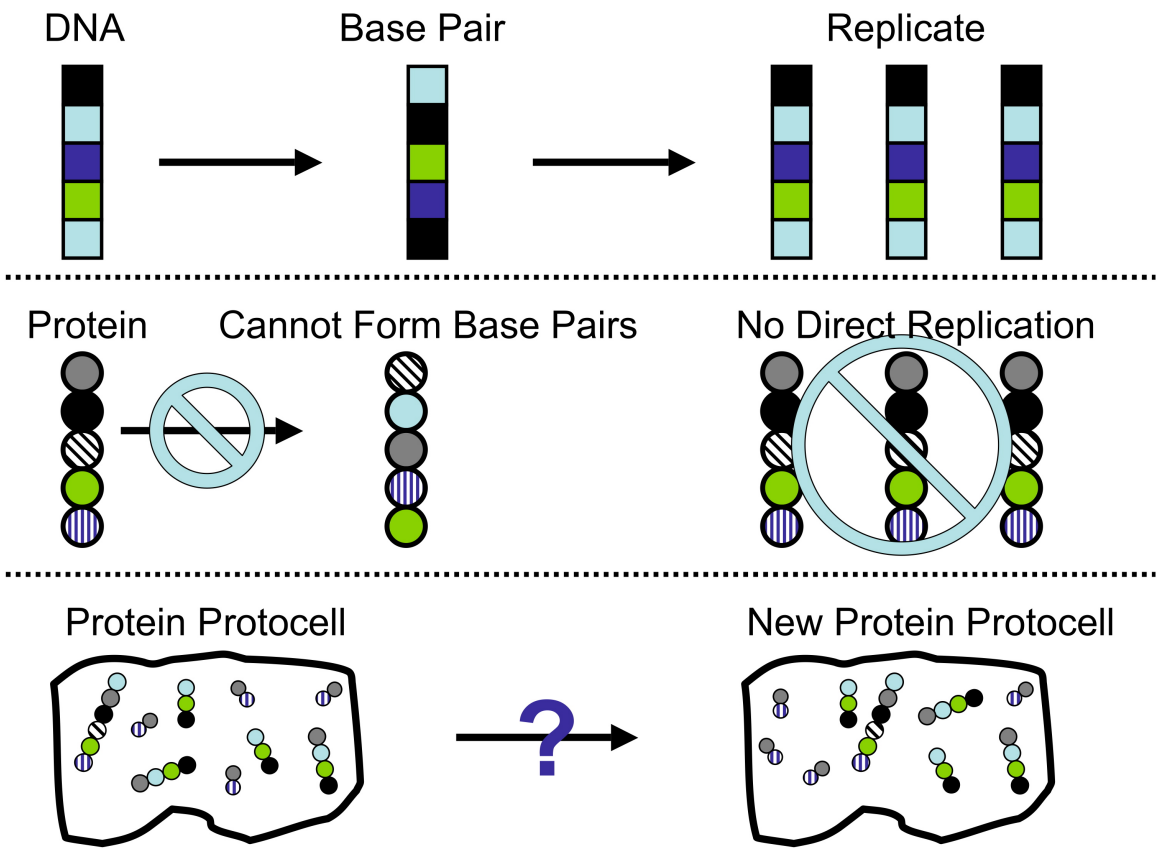

Figure 3.1: Replication of Proteins. DNA and RNA can form base pairs and directly replicate their sequences. Proteins cannot form base pairs and therefore cannot directly replicate themselves. However as a system, a group of proteins forming a protocell may be able to replicate by creating other protocells with approximately the same characteristics.

the evolution of a protocell based on its constituent proteins [20, 21, 28, 29, 36]. As a result current models are greatly simplified, using high-level assumption about the dynamics of the system [26]. One of the few models for non-genomic evolution is described in New and Pohorille [26] and is based on the relationship between the length of the protein and its efficiency - the percentage of time it succeeds in performing operations on other proteins. In this model, proteins perform two operations: hydrolysis and ligation. Hydrolysis is the process of decomposing a protein, whereas ligation is the process of building a new protein by joining two proteins. In this model the efficiency of ligation and hydrolysis are based on Gaussians, with longer proteins generally being more efficient. The fitness evaluation of the protocell in this model is based on protein length with longer proteins being evaluated higher than proteins with shorter length. The computational experiments performed with this model show that protocells become more fit with time. However, in this model the fitness eval-

uation is only concerned with the efficiency of two functions while numerous other properties are required for a viable protocell, such as a protein's ability to produce cell-membrane building lipids.

In this chapter, we will expand on these results and develop a more complex model of non-genomic evolution with the help of a multi-agent system. This setup differs from previous ones in that efficiencies that the proteins attain at certain tasks is determined by the actions of learning agents, allowing many more degrees of freedom in the dynamics of the evolution. In addition the proteins have more functions besides their ability to break and build other proteins. Finally the global utility used in evaluating the fitness of the protocell takes into account the distribution of all the protein functions in the protocell.

3.2 The Protocell Model

The model of the primitive protocell used in this paper is based on different protein types performing different tasks within the protocell and contributing differently to the fitness of the protocell. In this model a protocell has a set of proteins of various types. The two most important types of proteins are combiner and breaker proteins. Combiner proteins take two other proteins and combine them into a single protein (ligation). Breaker proteins take a single other protein and break them into two separate proteins (hydrolysis). Other protein types have important functions such as lipid production for cell walls, energy production and energy transport. However in our model, these other proteins do not actively contribute to the dynamics of the system and are only important in evaluating the final viability of the protocell as defined by the system's global utility.

3.2.1 Global Utility

For a protocell to survive and thrive, it needs different types of proteins performing different tasks. Certain types of proteins are needed for building cell membranes, while others are needed for tasks such as energy production, energy transportation and chemical exchange. If a cell has a poor distribution of proteins, such as having no proteins for building cell membranes, it will be unlikely to survive and should receive a poor utility. Our model makes an abstraction over the different types of possible proteins, ignoring their low-level functions. We only assume that there are optimal numbers of each type of protein and we use an exponential decay to assign utility to off-optimal distributions.

To formally define the global utility of a protocell, let there be k possible types of proteins and let y_i be the desired amount of proteins of type i in the system, while x_i represents the current amount of proteins of type i present in the system. We choose the functional form $x_i e^{-\frac{x_i}{y_i}}$, which takes its optimal value of 1 when $x_i = y_i$, to describe how close we are to the desired number of proteins performing function k. The intuition behind this choice is that there is an optimal amount of a particular protein needed in the protocell. If there is less than that amount, the protocell would not function properly, and if there is more than that amount, the protein uses more resources than needed to perform its task.

The global utility for the protocell is simply the sum of these functions:

$$G(x, y) = \sum_{i=1}^{k} x_i e^{-\frac{x_i}{y_i}} , \tag{3.1}$$

where x is the vector of the current amounts of each protein type and y is the vector of desired amounts of each protein type.

3.2.2 Protocell Dynamics

The dynamics of the protocell are determined by hydrolysis and ligation. During ligation, a "combiner protein" joins two proteins into a single protein. During hydrolysis, on the other hand, a protein is broken into two proteins. There are two crucial issues in both operations. First, the combiner or breaker protein has to choose the proteins on which it will act. Second the protein(s) resulting from the operation need(s) to inherit certain properties of their predecessors. These issues will be modeled through concepts of *specialization* and *clustering* respectively.

Based on current theory of non-genomic evolution, it is unlikely that combiner proteins in early protocells were specialized, i.e., they will combine two other proteins together regardless of what type they are [26]. However, it is likely that breaker proteins were somewhat specialized, i.e., there would be some types of proteins which a particular breaker protein would not break. While the actual specialization dynamics are somewhat complex, we model this specialization by assigning each breaker protein a single type of protein which it will never break. However, the breaker protein would break all other proteins.

When new proteins are created as a result of combining or breaking operations, they are assigned types based on the proteins they came from. This assignment will be modeled through the principle of "protein type clusters." In this model we assume that proteins of the same type tend to congregate near each other, forming clusters.

In our model each protein belongs to exactly one cluster, though multiple clusters may have proteins of similar types (Figure 3.2). When a new protein is created as a result of a breaking operation it will tend to go to a cluster of the same type as its parent (though not necessarily the same cluster). When a new protein is created based on combing two proteins, it will go to a cluster chosen based on the properties of the parent clusters. Details of these assignments are given in Section 3.3.

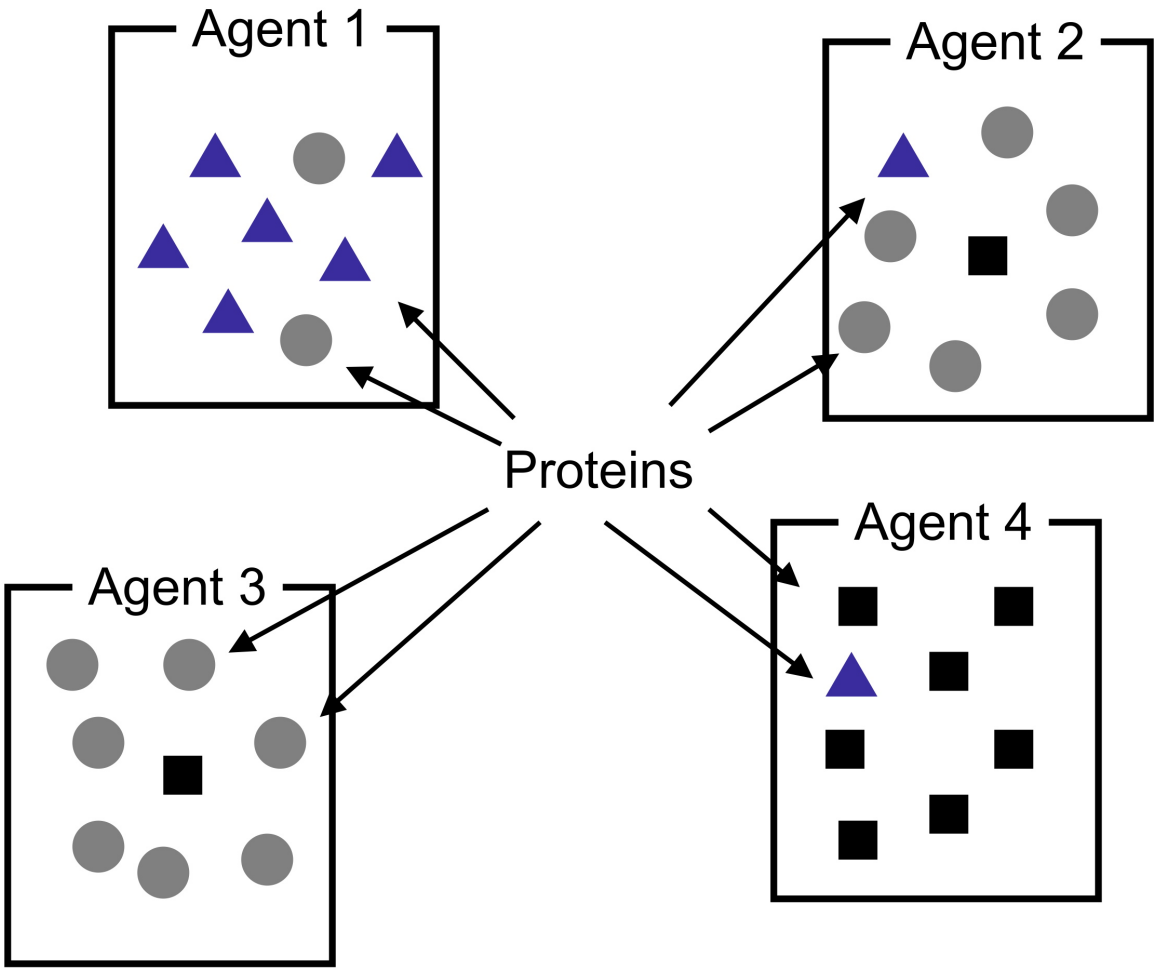

Figure 3.2: Type Agents. Each type agent determines the protein types in its cluster. Each agent makes decisions about the type of protein in its cluster independently so, proteins in the same cluster can have different types.

3.3 Agent Framework for Protein Model

Agents play an important role in determining the specialization of the breaker proteins and determining the properties of the clusters. Specifically, each cluster will be

assigned a "type" agent, whose actions determine the type of each protein in its cluster. Also each specialization of a breaker protein will be determined by the actions of a "breaker-specialization" agents. This section describes the details of the learning process and the actions taken by the agents.

3.3.1 Type Agents

The types of the proteins in a cluster are determined by an agent assigned to that cluster. In our model an agent takes a discrete action ranging from 1 to k, where k is the number of possible types. Every time an agent takes an action, its action determines the type of a single protein in its cluster chosen at random. Over time, an agent takes many actions, defining the types of more and more proteins within the cluster. The goal of the agent is to choose types that ultimately lead to a protocell of high global utility. To accomplish this task, our agents learn through a simple reinforcement learning. The system is modeled using a series of discrete time steps, where each agent takes an action after each time step. At the beginning of a time step each type agent takes an action determining the type of single protein in its cluster. This action is determined by an ϵ-greedy reinforcement learner with a learning table of size k, the number of protein types. Each entry in the table represents the agents expected utility when choosing the action associated with that entry. With probability $1-\epsilon$ an agent takes an action associated with the highest table entry. With probability ϵ it takes a random action. Note that with the ϵ-greedy learner, we would expect a cluster to have proteins of several different types. Most of the time the agent would set proteins to the type associated with highest expected utility, though sometimes it will choose random types with frequency depending on ϵ.

If an agent sets the protein type to something other than a combiner protein or a breaker protein, then nothing happens in the system until the next time step. However, if an agent chooses a protein to be a combiner or a breaker protein, the corresponding combining or breaking operations then take place. If the protein is a combiner, then two other proteins are chosen at random and combined together to form a new protein. If the protein is a breaker protein, then another protein is chosen at random for possible breaking. Whether this protein is actually broken depends on the specialization of the breaker protein, discussed next.

3.3.2 Breaker-Specialization Agents

Each protocell has several breaker-specialization agents associated with the proteins. A breaker-specialization agent will typically be associated with several proteins, but a protein will only be associated with one breaker-specialization agent. The actions of its breaker-specialization agent determine the specialization of any breaker proteins that are associated with it. At each time step, the breaker-specialization agent takes one of k actions chosen with an ϵ-greedy reinforcement learner. This action then defines the specialization of a breaker protein chosen randomly. This specialization determines the type of protein the breaker protein cannot break: If the protein it operates on during its breaking operation is of the same type as its specialization, then the operation does nothing.

3.3.3 Cluster Assignment of New Proteins

After a combining or breaking operation, the cluster that the new protein belongs to is determined by the clusters of the original proteins involved in the operation. In our model we want the child protein(s) that result from these operations to end up in cluster(s) that are "similar" to the cluster(s) the parent protein(s) came from. While there are many ways to define the similarity between clusters, our model will use the learning tables for the agents assigned to each cluster to define similarity (Figure 3.3). For a breaking operation, a protein p in cluster c is broken into two new proteins. To find the cluster c_{new} for one of the new proteins p_{new}, we first compute a vector, v as follows:

$$v = Q_c + e \ , \tag{3.2}$$

where Q_c is the learning table of the type agent associated with cluster c, and e is Gaussian noise. The cluster for the new protein, c_{new} is then chosen as the cluster associated with the agent that has the closest learning table to v, based on the Manhattan Norm (1-norm):

$$c_{new} = argmin_c \delta(Q_c, v) \ , \tag{3.3}$$

where $\delta(\cdot)$ is the Manhattan Norm.

For a combining operation a protein p_1 in cluster c_1 is combined with protein p_2 in cluster c_2 to form a single new protein, p_{new}. As with the breaking operation, to find the cluster c_{new} for the new protein we first compute a vector, v. However this time v is a function of two learning tables since p_{new} is created from two proteins. To handle this we combine the learning tables using the max operator $m(x, y)$, which

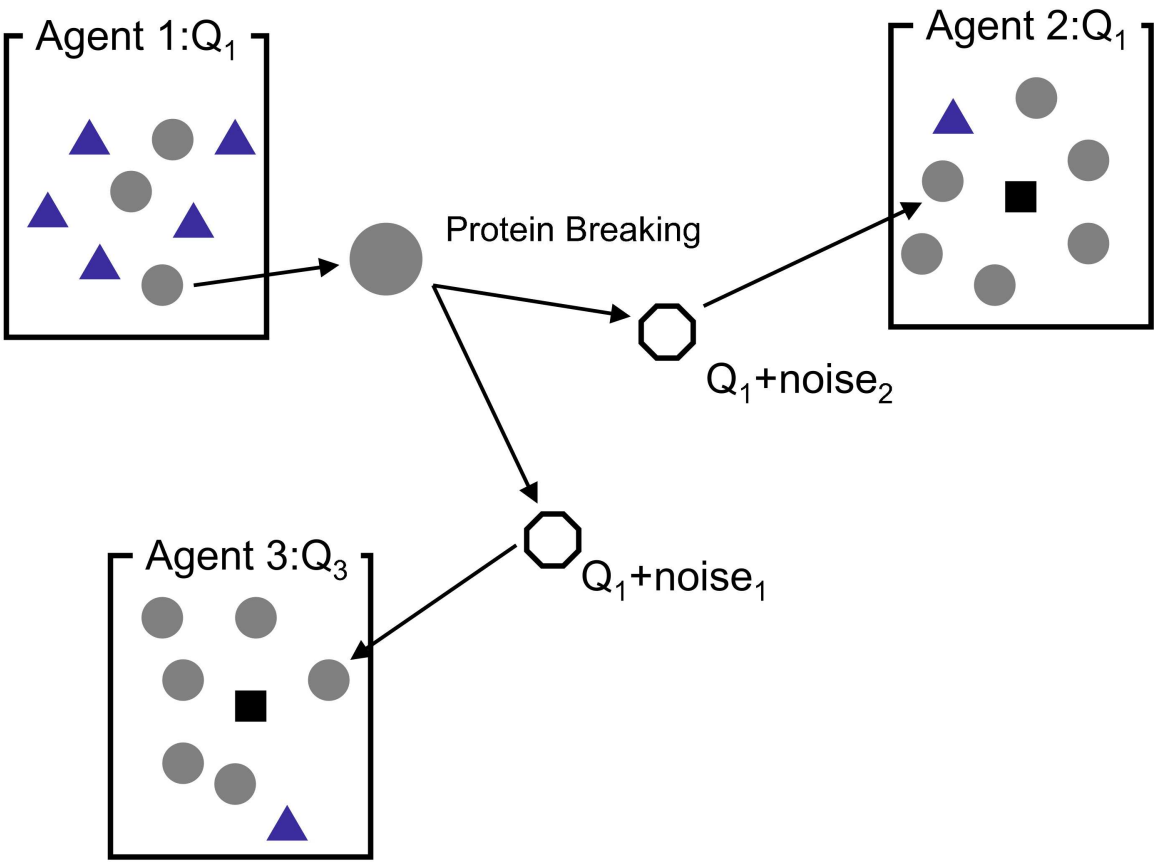

Figure 3.3: Breaking Proteins Apart. When a protein is broken apart, two child proteins are created. The types of each of the child proteins depend on the learning table of the agent for the cluster the original protein came from. For each child protein a new table is made by adding noise to this learning table. Each child protein is assigned to the the cluster whose agent has the learning table is closest to the proteins table.

assigns to each component of m_i the maximum of x_i or y_i: $m_i = \max(x_i, y_i)$. The vector v can then be computed as:

$$v = m(Q_{c_1}, Q_{c_2}) + e \ , \tag{3.4}$$

where Q_{c_1} and Q_{c_2} are the learning tables of the type agents associated with clusters c_1 and c_2, and e is Gaussian noise. As before the cluster for the new protein, c_{new} is then chosen as the cluster associated with the agent that has the closest learning table to v, based on the Manhattan Norm (1-norm):

$$c_{new} = argmin_c \delta(Q_c, v) \ , \tag{3.5}$$

where $\delta(\cdot)$ is the Manhattan Norm. The process for determining the breaker-specialization

agent associated with a new protein is parallel, with the learning tables of the breaker-specialization agents being used.

3.3.4 Agent Utilities

The goal of the agents is to maximize the global utility $G(x, y)$ as defined in Equation 3.1. However when there are many agents, it is difficult to maximize this utility. This difficulty arises from the fact that a single agent will only have a small impact on the value of G. If an agent takes an action and G increases, that agent does not know if the increases were due to its action or the actions of the other agents. When agents do not have significant influence over their utility as compared to the influence of all the other agents, it usually takes a very long time for them to learn to take actions that maximize it.

To alleviate this problem, each agent maximizes a "difference utility" instead of the global utility. A difference utility is specific to agent i, and can be computed as follows:

$$D_i(z) = G(z) - G(z_{-i}) \ , \tag{3.6}$$

where z is a vector of all the actions taken by all the agents and z_{-i} is a vector of all the actions except for the action taken by agent i. The second term of the difference equation is a counterfactual version of the global utility that can be seen as the value of the system without agent i. Therefore the difference utility can be seen as the agent's net contribution to the system. By subtracting this counterfactual second term from first term, the difference equation removes much of the noise caused by the actions of the other agents. The difference utility has been shown to greatly speed up reinforcement learning in many domains, including internet traffic routing, multi-robotic control and satellite communication [33, 35, 2].

The specific form of the difference utility for our global utility, $G(x, y)$, defined in this paper is:

$$D_i(x, y) = G(x, y) - G(x_{-i}, y) \ , \tag{3.7}$$

where x_{-i} represents the counts of the protein types for a protocell, had agent i not taken an action. The exact form of x_{-i} depends on the type of action made, especially if the agent made a protein a combiner or a breaker. For all actions let j be the type of protein chosen by the agent. For combiner actions let a and b be the types of proteins that were combined and let c be the type of protein produced. For breaker actions let a be the type of protein broken and let b and c be the types of proteins produced. Then the value of x_{-i} is as follows:

1. $x - e_j$ for non-combiner or breaker proteins ,
2. $x - e_j + e_a + e_b - e_c$ for combiner proteins ,
3. $x - e_j + e_a - e_b - e_c$ for breaker proteins ,

where e_j is a vector where element j has a value of 1 and the other elements are 0. Essentially for combiner and breaker proteins, the proteins created are removed from the counts and the proteins consumed are returned.

3.4 Results

To test the ability of our agent-based system to produce effective models, we ran a number of experiments where agents had to create models that led to protocells with high utility starting with different protein distributions. This task was non-trivial as agents had to create a model with the proper distribution of combiner and breaker proteins, as well as ensuring that the breaker proteins had the proper selectivity profile. In addition agents had to ensure that the functions of the other proteins were in the proper distribution.

In all the experiments there were thirty agents. In each experiment there were five different types of proteins that were desired in the final outcome besides combiner and breaker proteins (e.g., proteins for energy collection, energy transport, membrane building, and nucleus building). In each experiment, we tested different desirable combinations of these proteins by selecting an optimal number of each these five proteins, y_1, y_2, y_3, y_4, y_5, along with a corresponding optimal total $y = y_1 + y_2 + y_3 + y_4 + y_5$. The first three experiments tested the ability of the agents to create a model that reached a good final equilibrium, starting from approximately y, 2, and $3y$ proteins respectively. Starting from 2 proteins, we intended to determine how little is needed for the process to start. Intuitively, this experiment tested whether a protocell can develop from two proteins and generate the necessary proteins to perform all the required tasks. At the other extreme the experiments starting with $3y$ proteins, we tested whether the protocell can cut through clutter and elimate unnecessary proteins. The fourth and last experiment tested the ability of the protocell to adapt to a suddenly different environment where the optimal number of each type of protein is changed halfway through a simulation.

In the first experiment the protocell started with 30 proteins. The optimal number of proteins was 31 $(1+2+4+8+16)$, with $y_1 = 1, y_2 = 2, y_3 = 4, y_4 = 8$, and $y_5 = 16$. Since the starting and optimal number of proteins was almost the same, this problem

was relatively simple and the only task of the agents was to create a model that forms the proper distribution and maintains it. However, despite the simplicity of the task, Figure 3.4 shows that agents directly maximizing the global utility could not produce an adequate model. In fact the initial actions of the agents produced a model that was worse than the starting one, and after 5000 learning trials they still did not manage to reach their initial starting performance. This result is not surprising since many choice of model parameters can be very destructive, such as creating too many breaker proteins. Since it was very difficult for the agents to learn from the global utility, they had difficulty avoiding these destructive choices. In contrast, agents using the difference utility were able to produce adequate models for this task. By changing the distribution they were able to create a model that achieved a utility of 3.7, up from the initial utility of 2.8. The agents using the difference utility were more effective than the agents using the global utility because the difference utility was much more learnable. The significance of this model to non-genomic evolution, however is minimal. It shows that the proper distribution can be maintained, intuitively meaning that there should be few combiner and breaker proteins.

In the second experiment, the protocell started with 2 proteins. The optimal number of proteins was 31, with $y_1 = 1, y_2 = 2, y_3 = 4, y_4 = 8$, and $y_5 = 16$. This presented a much more difficult setting than the previous one, since starting with 2 proteins, the agents had to create a model that increases the number of proteins by a factor of 15, while providing that the proteins are in the proper distribution. This requires an initial increase in the distribution of breaker proteins to create enough total proteins and a shift in the distribution, as well, to maintain the proper protein levels. The results show that again agents using the difference utility are able to create a model that leads to highly fit protocells. In this experiment, the performance of the protecell goes from 0.8 to 3.5, a significantly larger increase than in the previous experiment. This larger increase is expected since the initial protocell with two proteins is in a much worse state than the one starting with 30. This experiment shows that a model exists where a protocell with a small number of proteins can grow to one with a large number of proteins and form the proper protein distribution. This has significant insight into the required initial conditions for the formation of protocells. This result demonstrates that with the proper conditions (simulated here through the learning agents building the model) a protein ecology can be formed starting from only 2 proteins.

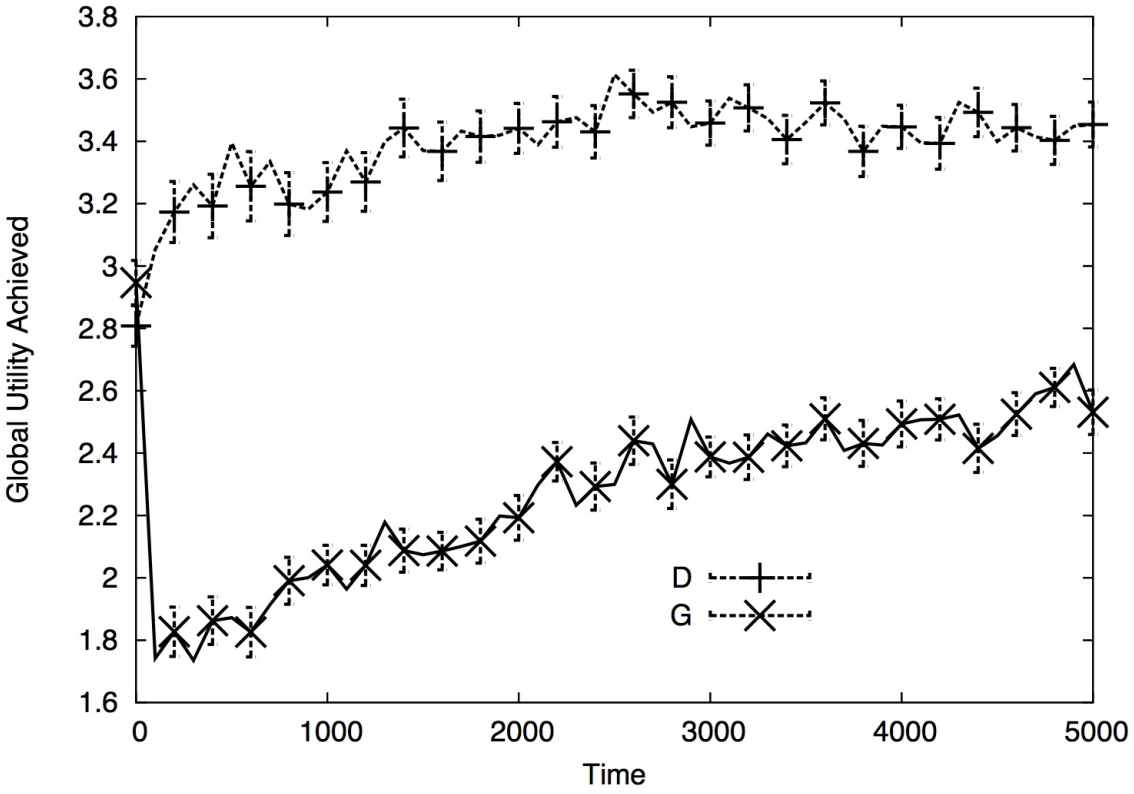

Figure 3.4: Performance of model for protocell starting with close to the optimal number of proteins. Here global utility is unusable since it cannot even recover to original performance. Agents using difference utility form adequate model by changing protein distributions to fit targets.

In the third experiment the protocell started with 95 proteins. The optimal number of proteins was again 31, with $y_1 = 1, y_2 = 2, y_3 = 4, y_4 = 8$, and $y_5 = 16$. This experiment requires an initial increase in the distribution of combiner proteins so that proteins in the initial high population are combined together to form a smaller population. As before results show that agents using the difference utility are able to create a model that leads to highly fit protocells. In this experiment the performance of the protocell quickly shoots from 2.2 to 3.6, but then goes down slightly. This slight reduction in utility is probably due to the model producing too many combiner proteins that causes the system to overshoot its protein reduction. This experiment shows that in situations where there are too many proteins in a protocell, a mechanism can be created to reduce the number and create a proper protein distribution.

In the final experiment, the protocell started with 35 proteins with an optimal number of proteins of 31. The optimal number of proteins was thirty one, with

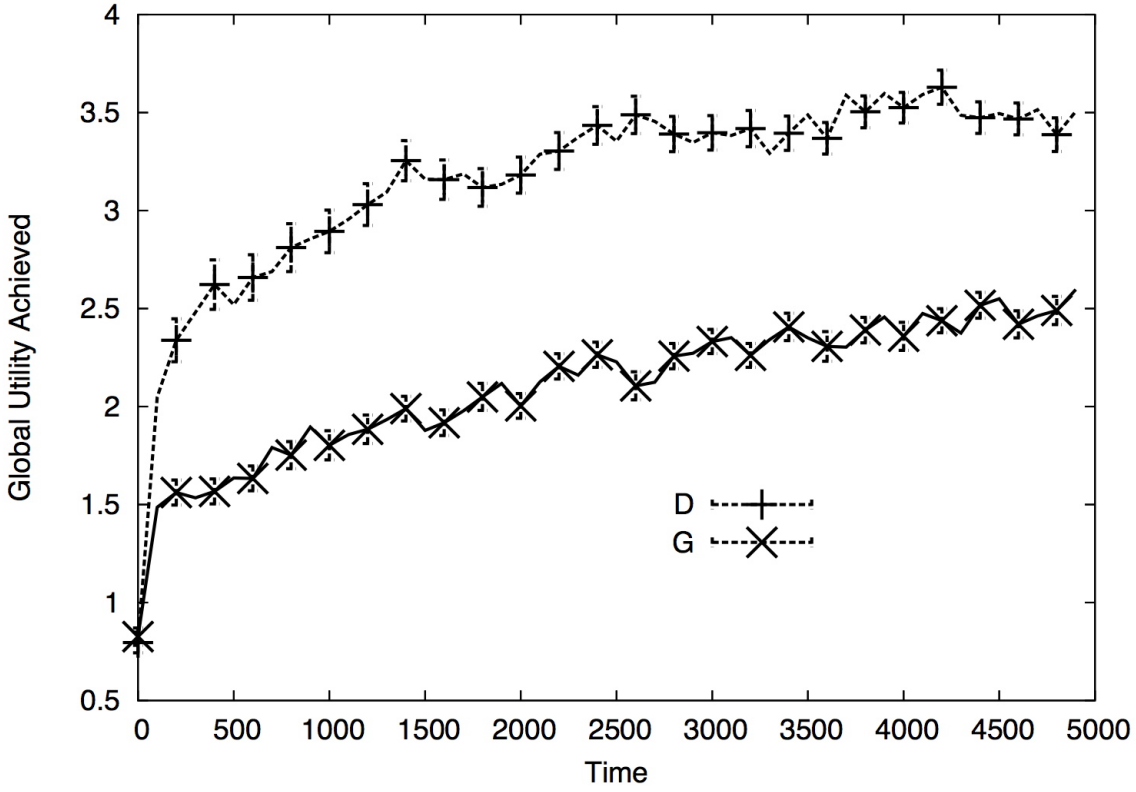

Figure 3.5: Performance of model for protocell starting with one fifteenth of the optimal number of proteins. Agents using difference utility are able to form model where combiner and breaker proteins are formed that lead protocell to an equilibrium where there are enough proteins, and the proteins are in the correct distribution.

$y_1 = 1, y_2 = 2, y_3 = 4, y_4 = 8, y_5 = 16$, but at 5000 episodes is changed to $y_1 = 16, y_2 = 8, y_3 = 4, y_4 = 2$, and $y_5 = 1$. This experiment requires the agents to adapt to new model requirements after they have already developed an existing model. The results shown in Figure 3.7 show that the agents are able to quickly adapt to this change in model requirements.

3.5 Discussion

We explore models to simulate potential ways in which early non-genomic cells can evolve to perform the tasks required for survival. We show that multi-agent systems help model such systems and provide simulation results important to the field of astrobiology. Through the use of learning agents, we create an abstract dynamical model for a primitive protocell of considerably greater complexity than previous models. In particular our models include proteins that had many different properties and had

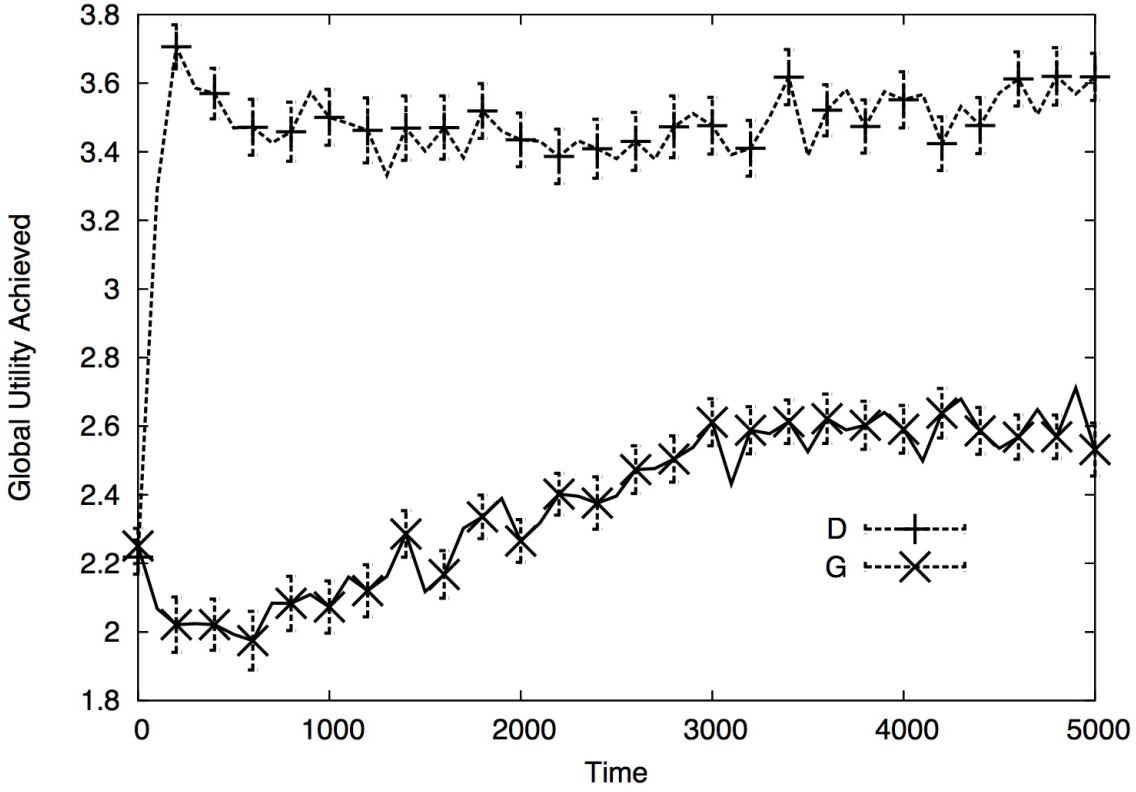

Figure 3.6: Performance of model for protocell starting with three times the optimal number of proteins. Agents using difference utility are able to form model where combiner and breaker proteins are formed that lead protocell to an equilibrium where there are enough proteins, and the proteins are in the correct distribution.

more complex dynamics for protein combining and breaking. In addition, while the fitness utility used in other models directly reflects the dynamics of the system, such as efficiency, our fitness utility is only indirectly influenced by the dynamics. Therefore our fitness utility is more realistic and provides many more degrees of freedom for the system to evolve. With our framework, we show that models exists that can produce highly fit protocells from a wide variety of starting conditions.

While our use of multi-agent systems allow us to create models of non-genomic evolution that are more complex than previous models, they are still abstract and highly simplified. One issue with the model is that only combiner and breaker proteins affect the dynamics of the system. Another issue is that the optimal protein distributions used in our global utility of the protocell are somewhat arbitrary. However our framework allows other distributions to be plugged in.

Even though many details need to be added to our models to increase their biological plausibility, the multi-agent framework provides the power and flexibility to do so

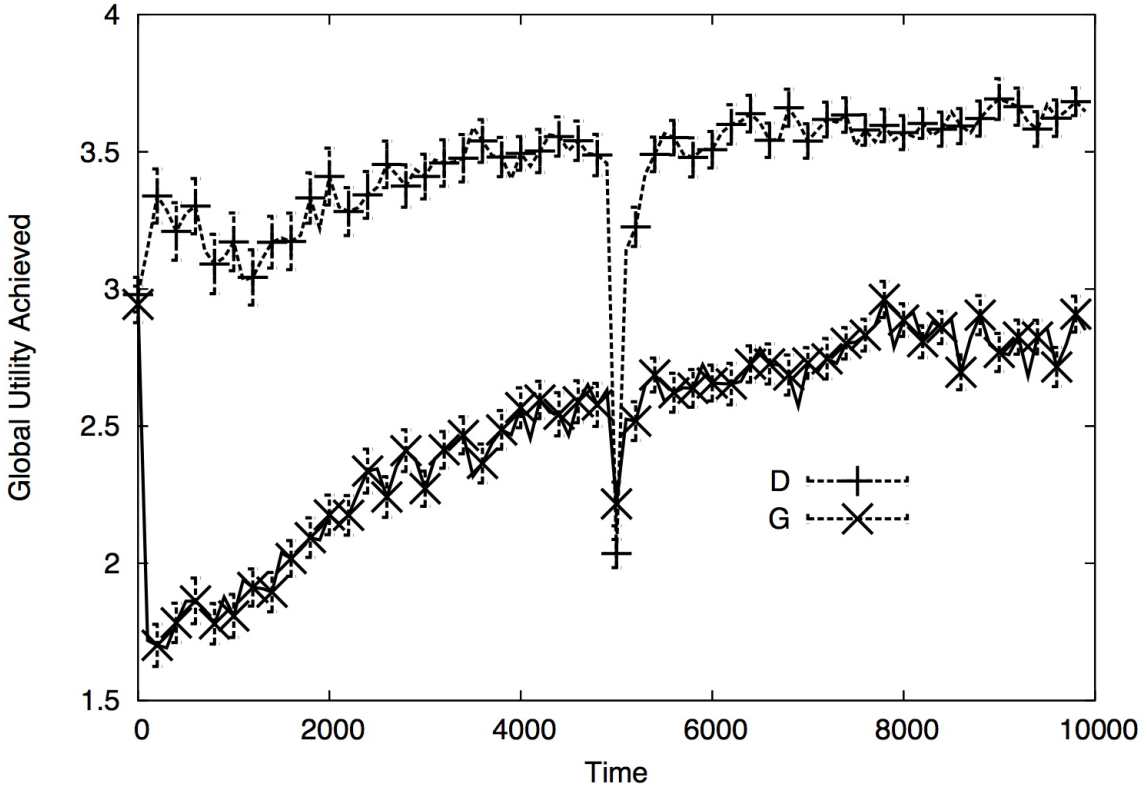

Figure 3.7: Model Formation for Changing Distribution. When requirements for the distribution of proteins changes at time step five thousand, agents are able to form a model that adjust to change.

and naturally matches the distributed properties of protein-only ecologies. In general more complex evaluation of protocells can simply be implemented through the agent utilities. In addition more complexity can be introduced by adding different types of agents or changing the agents' action space. Due to their flexibility it is likely that multi-agent systems will become an important tool in the study of astrobiology.

Chapter 4

Conclusions

We have shown a way to forecast time-series by parameterizing an artificial market model such as the Minority Game [8], using an iterative numerical optimization technique. A method is also provided for how to follow an effective forecasting scheme leading to pockets of predictability. These concepts are meant only to serve as an introduction to such methods of forecasting. When coupled with a more appropriate underlying market model, these techniques could provide useful insight into the composition of the underlying multi-trader population, and hence yield superior prediction estimates of the resulting financial time-series which is generated by this population.

We have also shown how protein ecologies can be modeled as a complex system in order to learn protein distributions leading to a viable protocell. Such modeling can lead to robust results explaining protein replication without the assistance of nucleic acids. In our model, we make use only of the protein functions corresponding to ligation and hydrolysis leaving us with a rather primitive model. In addition, our global utility function has little practical meaning in the simulations (but can be adapted quite easily). Again, this work is meant only to serve as an introduction and hopefully bring about the realization that agent-based modeling is very applicable to this problem. Once more detail is added to the system, we may be able to come up with a class of initial distributions of proteins that must be present to lead to viable protocells, one of the main questions at the root of astrobiology.

Chapter 5

Future Work

At the moment, I am looking down a number of different pathways for future research. These ideas are generally focused on the applications of complex systems to time-series analysis, especially those of financial markets. I will briefly mention a few of these avenues below.

5.1 Magnitude v. Direction

In the artificial market model for financial markets we provided earlier (MG), the agents score strategies based on the sign of the past series (which represents the winning decision). Using only information about the sign, however, throws away a great deal of information in the time-series. We can think of each element of the series as a combination of a magnitude and direction added on to the previous element. In the case of the MG, we lose magnitude information entirely and retain perfect information for direction.

It would be easy to imagine the following cases:

- Two time-series for which the direction between two consecutive elements in one is identical to the direction between the same two consecutive elements in the other. However, they both produce quite different looking time-series. For example, see Figure 5.1.

- Two time-series for which the magnitude between two consecutive elements in one is identical to the magnitude between the same two consecutive elements in the other. However, they both produce quite different looking time-series. For example, see Figure 5.2.

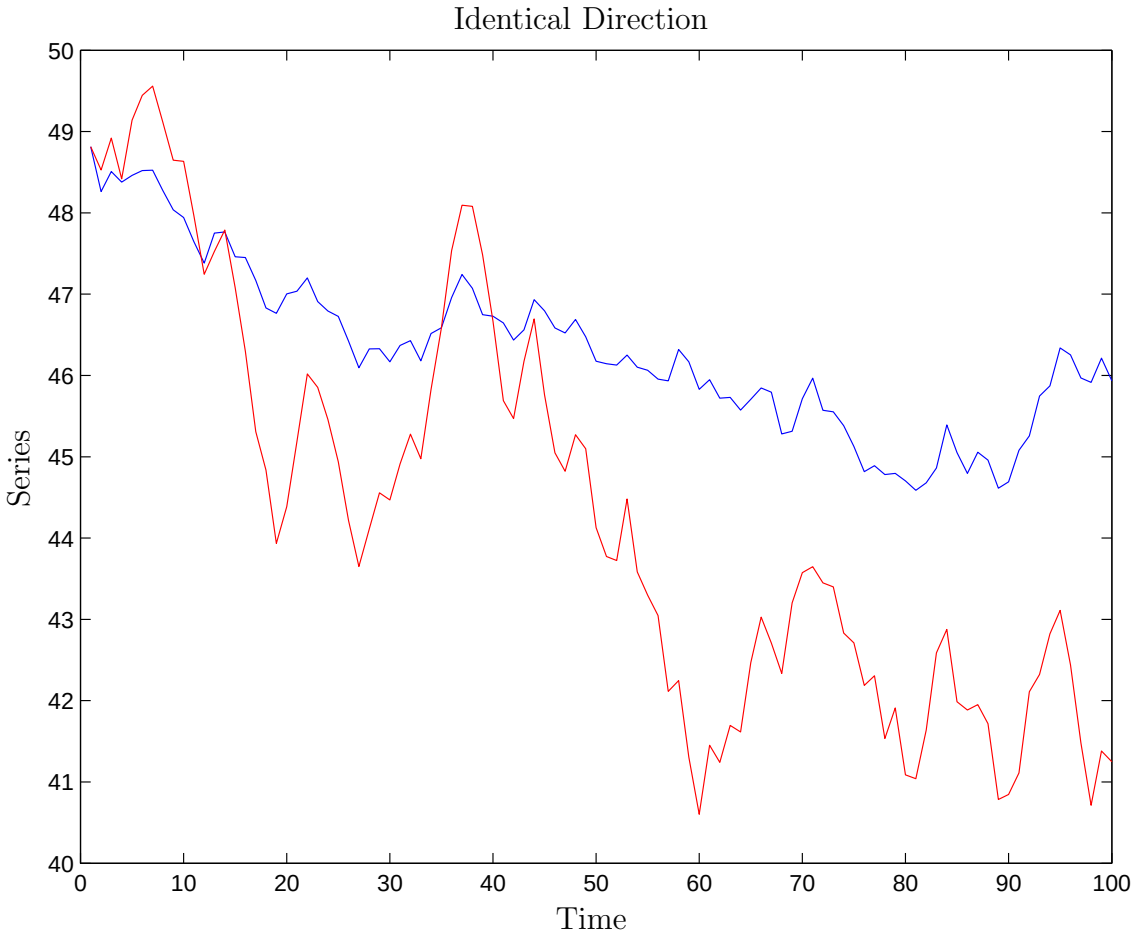

Figure 5.1: The two time-series above have identical directions for each fluctuation.

- Two time-series which do not necessarily match the direction and magnitude between consecutive elements well, but do look similar (of course similar and different are relative terms here, which can more formally be defined in a number of ways, e.g., thresholding the integral over the difference between two time-series). For example, see Figure 5.3

So it should be clear by these figures that capturing only direction or magnitude will not guarantee that we know much about the time-series. Of course in certain instances, the trader may truly be interested in only one or the other. However, in the majority of cases, utilizing only one would surely be a poor approximation.

In Figure 5.3, not only are we creating a time-series that seems to be quite similar to the original (77 of the 99 directions are the same as well), but we are only using about half the number of points to construct the time-series. In order to generate

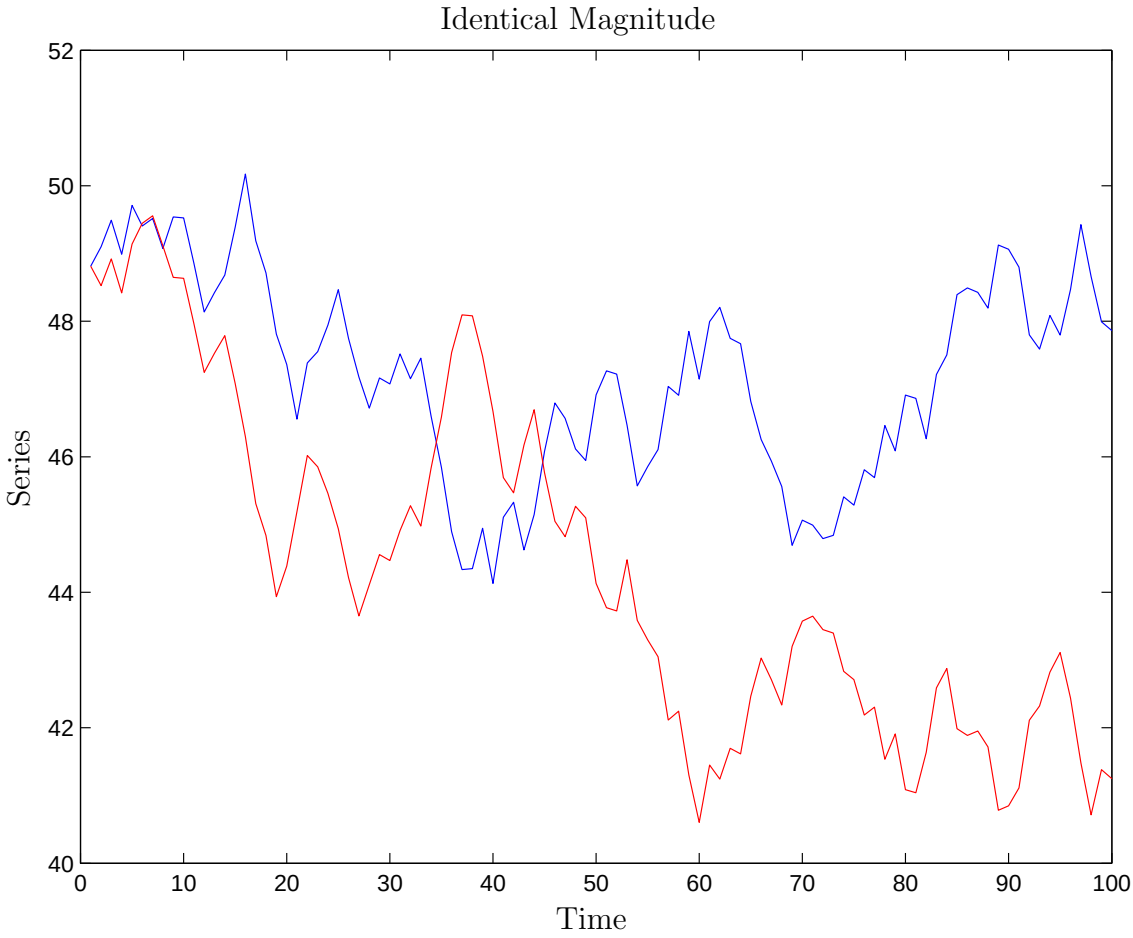

Figure 5.2: The two time-series above have identical magnitude for each fluctuation.

the series in blue, we use 51 scalars each of which carries a special meaning. These are the coefficients of the approximation portion of a discrete wavelet transform using the 'db2' wavelet [31, 24, 25]. Using a discrete wavelet transform, we easily find a breakdown of the series in terms of coefficients for a given wavelet function denoted by the approximation and detail coefficients using the low-pass and high-pass filter, respectively. In Figure 3, we used only the approximation coefficients, but one can imagine picking certain detail coefficients also (which represent the localized high-frequency fluctuations).

We are interested in applying these ideas to the strategies of agents in multi-agent games representing an artificial market model. Rather than looking at the direction of movement from one time-step to the next, we propose comparing it to strategies given by coefficients of a wavelet transform. For example, we can have an agent with two strategies, each given by $n + 2n$ scalars. The n scalars can represent the

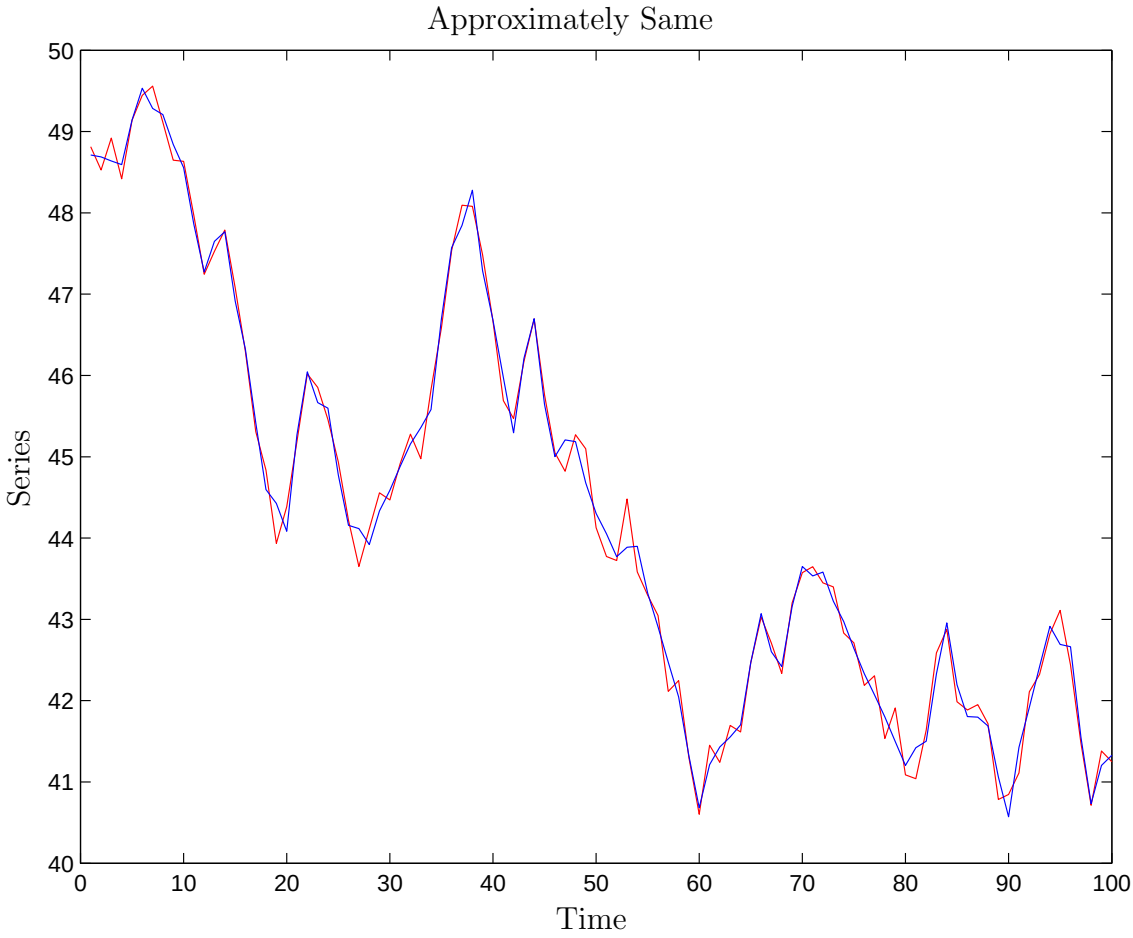

Figure 5.3: The two time-series above are similar to the naked eye although they do not depict identical direction or magnitude fluctuations. In fact one is a wavelet approximation of the other.

coefficients (or amplitudes) of certain wavelets, while the $2n$ can perfectly determine which wavelet in the time and scale domain to choose from. We can then see which of the two strategies generates the time-series (using an inverse wavelet transform) that is the nearest neighbor to the real time history. We can imagine that the nearest neighbor can be determined in a number of different way (e.g., based on the integral of the difference between the strategy and the real history).

This technique of choosing strategies might provide a much more effective use of the history time-series.

5.2 Forecasting using an Artificial Multi-Market Model

So far in this thesis, we have concentrated solely on cases where we would like to model one time-series. However, this isn't very practical for real models. As soon as we enter a multi-market model [27], our agents might be most interested in placing their money in the market which they believe will give the highest return although they may believe a number of them will give positive returns. In order to reduce risk in their portfolio, they may spread weighted proportions across some markets they believe will provide positive return. This presents a much more realistic problem which has some very different behavior. We would like to carry out this natural extension and see how the agents may interact differently now. For example, rather than having strategies over fluctuations in the time-series, the agents might now want to look at the fluctuations in the past log return series since they would like to invest in the market with highest log return at every time-step.

5.3 Approximating a Continuously Adjusting Minority Game

Another idea I would like to look into is the idea of removing the binary nature entirely from the Minority Game. At the moment a strategy in the minority game is a map $s : \{-1, +1\}^n \to \{-1, 1\}$. We could imagine replacing this with $s : \mathbb{R}^n \to \mathbb{R}$ and choosing from a number of possible ways to score these strategies (e.g., something similar to the wavelet idea in Section 5.1). Further, we would like to allow these maps over time to perturb slowly responding to the possible changes in the mind of the agent as the agent watches what strategies perform well (e.g., using machine learning or noisy weighted averaging).

If we were able to come up with a nice method for this, the next step might be to approximate the space of strategies of each agent so each agent doesn't hold exactly 2 or 3 strategies but some probability measure over a number of strategies. Perhaps, we could even approximate over the space of all agents similarly.

I'm not yet sure how feasible this is. It could turn out to be very easy or very hard, but it is certainly something I am hoping to look into as it would surely be quite exciting.

5.4 The Effects of News

In research of financial markets, there has been a recent trend to study the effects of news on the fluctuations of a time-series [11, 32]. News is often times an unexpected source of noise, but could be considered quantifiable. We might be able to add an exogenous state representing the news into our current prediction scheme so a trader could input exogenously the beliefs of news effects. Further, if we assume news reverberates around for a while, we could realize the sudden change and adapt our predictions in response. We could also imagine classifying news as surprising or unsurprising, relevant or irrelevant, and good or bad, and then adapt in some prescribed way according to these features.

5.5 Evolving the Model for Protein Ecologies

I am planning to continue the modeling of protein ecologies. The most important next step would be to continue speaking with biologists about protein functions and how they relate to other proteins. The idea would be to incorporate these functions into the global utility and also implement any internal effects they might have over other proteins. As we extend the current model, this project might show some very interesting results to explain some of the phenomena of this complex system. The feasibility of this extension is largely dependent on what biological reactions can be modeled using reasonable computation. We could also imagine at some stage removing the reinforcement learning and adapting different techniques which might be considered more realistic.

References

[1] http://www.unifr.ch/econophysics/.

[2] A. Agogino and K. Tumer. Efficient evaluation functions for multi-rover systems. In *The Genetic and Evolutionary Computation Conference*, pages 1–12, Seatle, WA, June 2004.

[3] Jørgen Vitting Andersen and Didier Sornette. The $-game. *The European Physical Journal B*, **31**:141, 2003.

[4] Jørgen Vitting Andersen and Didier Sornette. A mechanism for pockets of predictability in complex adaptive systems. *Europhysics Letters*, **70**(5):697–703, 2005.

[5] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley and Sons, Inc., 2001.

[6] George Casella and Roger L. Berger. *Statistical Inference*. Thomson Learning, 2002.

[7] Damien Challet. Inter-pattern speculation: beyond minority, majority and $-games. *e-print physics/0502140 at xxx.lanl.gov*, February 2005.

[8] Damien Challet, Matteo Marsili, and Yi-Cheng Zhang. *Minority Games*. Oxford University Press, 2005.

[9] Y. T. Chiang, L. S. Wang, F. R. Chang, and H. M. Peng. Constrained filtering method for attitude determination using gps and gyro. *IEE Proceedings - Radar, Sonar, and Navigation*, **149**(5):258–264, October 2002.

[10] Bernard Friedland. Treatment of bias in recursive filtering. *IEEE Transactions on Automatic Control*, **14**(4):359–367, August 1969.

[11] Lee Gillam, Khurshid Ahmad, Saif Ahmad, Matthew Casey, David Cheng, Tugba Taskaya, Paulo C F de Oliveira, and Pensiri Manomaisupat. Economic news and stock market correlation: A study of the uk market. In *Workshop at the International Conference on Terminology and Knowledge Engineering, Nancy, France*, August 2002.

[12] Nachi Gupta, Adrian Agogino, and Kagan Tumer. Efficient agent-based models for non-genomic evolution. In *Submitted to Autonomous Agents and Multi Agent Systems 2006*, 2005.

[13] Nachi Gupta, Raphael Hauser, and Neil F. Johnson. Deducing the multi-trader population driving a financial market. In *Proceedings of SPIE: Complex Systems*. The International Society for Optical Engineering, 2005.

[14] Nachi Gupta, Raphael Hauser, and Neil F. Johnson. Using artificial market models to forecast financial time-series. In *Workshop on Economic Heterogeneous Interacting Agents 2005, e-print physics/0506134 at xxx.lanl.gov*, June 2005.

[15] A.J. Hager, J.D. Pollard, and J.W. Szostak. Ribozymes: Aiming at RNA replication and peptide synthesis. *Chemistry and Biology*, **3**:717–725, 1996.

[16] P. Jefferies, M.L. Hart, P.M. Hui, and N.F. Johnson. From market games to real-world markets. *The European Physical Journal B*, **20**:493–501, 2001.

[17] Neil F. Johnson, David Lamper, Paul Jefferies, Michael L. Hart, and Sam D. Howison. Application of multi-agent games to the prediction of financial time-series. *Physica A: Statistical Mechanics and its Applications*, **299**(1–2):222–227, October 2001.

[18] G.F. Joyce. Ribozymes - building the RNA world. *Current Biology*, **6**:965–967, 1996.

[19] David Lamper. *Problems in Mathematical Finance : Market Modelling and Derivative Pricing*. PhD thesis, University of Oxford, 2002.

[20] D.H. Lee, J.R. Granja, J.A. Martinez, K. Severin, and M.R. Ghadiri. A self-replicating peptide. *Nature*, **382**:525–528, 1996.

[21] D.H. Lee, K. Severin Y. Yokobayashi, and M.R. Ghadiri. Emergence of symbiosis in peptide self-replication through a hypercyclic network. *Nature*, **390**:591–594, 1997.

[22] A De Martino, I Giardina, and G Mosetti. Statistical mechanics of the mixed majority–minority game with random external information. *Journal of Physics A: Mathematical and General*, **36**(34):8935–8954, 2003.

[23] Peter S. Maybeck. *Stochastic Models, Estimation and Control*, **2**. Academic Press, Inc., 1982.

[24] Yves Meyer. *Wavelets*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1993. Algorithms & applications, Translated from the French and with a foreword by Robert D. Ryan.

[25] Michel Misiti, Yves Misiti, Georges Oppenheim, and Jean-Michel Poggi. *Wavelet Toolbox for use with MATLAB*. The MathWorks, Inc., 2005.

[26] M. H. New and Andrew Pohorille. An inherited efficiencies model of non-genomic evolution. *Simulation Theory and Practice*, **8**:199–208, 2000.

[27] Tadeusz Platkowski and Michal Ramsza. Multimarket minority game. *Advances in Complex Systems*, **8**(1):65–74, March 2005.

[28] R.W. Roberts and J.W. Szostak. RNA-peptide fusions for the in vitro selection of peptides and proteins. In *Proceedings of the National Academy of Science USA*, pages 12297–12302, 1997.

[29] K. Severin, D.H. Lee, J.A. Martinez, M. Vieth, and M.R. Ghadiri. Dynamic error correction in autocatalytic peptide networks. *Angewandte Chemie International Edition*, **37**:126–128, 1998.

[30] Dan Simon and Donald L. Simon. Kalman filtering with inequality constraints for turbofan engine health estimation. Technical Report A491414, National Aeronautics and Space Administration, John H. Glenn Research Center at Lewis Field, February 2003.

[31] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[32] Omer Suleman, Mark McDonald, and Neil F. Johnson. (to be published).

[33] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, New York, 2004.

[34] L. S. Wang, Y. T. Chiang, and F. R. Chang. Filtering method for nonlinear systems with constraints. *IEE Proceedings - Control Theory and Applications*, **149**(6):525–531, November 2002.

[35] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, **4**(2/3):265–279, 2001.

[36] S. Yao, I. Ghosh, R. Zutshi, and J. Chmielewski. Selective amplification by auto and cross-catalysis in a replicating peptide system. *Nature*, **396**:447–450, 1998.