2006-11-08

# Learning in Short-Time Horizons with Measurable Costs

Patrick Bowen Mullen
*Brigham Young University - Provo*

# LEARNING IN SHORT-TIME HORIZONS WITH MEASURABLE COSTS

by

Patrick B. Mullen

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

December 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Patrick B. Mullen

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

| | |
|---|---|
| Date | Kevin D. Seppi, Chair |
| Date | Dan A. Ventura |
| Date | Scott N. Woodfield |

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Patrick B. Mullen in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

| | |
|---|---|
| Date | Kevin D. Seppi<br>Chair, Graduate Committee |
| Accepted for the Department | Parris K. Egbert<br>Graduate Coordinator |
| Accepted for the College | Dana T. Griffen<br>Associate Dean, College of Physical and Mathematical Sciences |

# ABSTRACT

## LEARNING IN SHORT-TIME HORIZONS WITH MEASURABLE COSTS

Patrick B. Mullen
Department of Computer Science
Master of Science

Dynamic pricing is a difficult problem for machine learning. The environment is noisy, dynamic and has a measurable cost associated with exploration that necessitates that learning be done in short-time horizons. These short-time horizons force the learning algorithms to make pricing decisions based on scarce data.

In this work, various machine learning algorithms are compared in the context of dynamic pricing. These algorithms include the Kalman filter, artificial neural networks, particle swarm optimization and genetic algorithms. The majority of these algorithms have been modified to handle the pricing problem. The results show that these adaptations allow the learning algorithms to handle the noisy dynamic conditions and to learn quickly.

# ACKNOWLEDGMENTS

I want to thank those who helped me with my degree, and most especially with this thesis. It has been a long road and not one that I could have walked alone.

First I would like to thank my advisor, Kevin Seppi. His patience and guidance helped me when I did not feel like going on. I am not sure that I could have done this with any other advisor. I appreciate his advice and his earnest desire to see me graduate.

I would also like to thank Chris Monson. His code, tricks and tips literally saved me hundreds of hours in getting results and actually putting this document together. I used to think quite highly of myself until I sat next to him, so thank you Chris for also teaching me humility.

Mostly I would like to thank my wife, Karisa. First off, I appreciate her for actually reading this thesis even though she has no idea who Kalman is and why he filters things. Her help with the editing process saved me from aging prematurely. Also, her love and patience through the whole time, even though it always took longer than I said it would, is what kept me going.

# Contents

# Chapter 1

## Introduction

Russell and Norvig, in discussing the purposes behind the study of artificial intelligence, state that machine learning is important because "having a better idea of how the world works enables us to generate more effective strategies for dealing with it" [Russell and Norvig 2003]. This thesis studies machine learning applied to dynamic pricing on commercial websites, or making adjustments to prices in response to market changes. The pricing problem is difficult because it is both noisy and dynamic and since there is a real-time cost associated with learning, it must be done in short-time horizons.

From the commercial website perspective there are many factors that could lead to noise. Noise occurs when unmodeled occurrences affect the observation (number of products sold). On a day to day basis there are many things that could affect sales that are extremely difficult to model. If a major ISP has some problems, traffic on the Internet could drop and less customers do their purchasing for one day. Sudden shifts in the global economy, natural disasters or other major news headlines may affect demand positively or negatively. It is hard to include these events in a demand model.

Dynamics means that the market has changed or will change with time. A classic example of this is airline tickets. Demand for an airline ticket is completely different two months before a flight is scheduled than it is in the hours before departure. Airline companies know this and adjust prices to reflect the state of the market. Other examples include holiday sales or the start of a new semester at a college book store. Demand is going to change drastically and the company needs to adjust policies to reflect this change.

From the point of view of the seller noise and dynamics are very similar. The economic models used in this thesis, which will be discussed later, are simplified. Noise happens when un-modeled occurrences affect demand. The model becomes dynamic when the parameters being used change with time. In the examples of noise given above, the factors affecting demand could be modeled, in which case they would become dynamic instead of noisy - they are not modeled however because it is extremely difficult to do so. A major difficulty occurs when problems are both noisy and dynamic. When sales fluctuate with time it is not easy to differentiate between noise and change.

As mentioned above, another difficulty of dynamic pricing is that there is a real-time cost while learning takes place. Prices can be set to take advantage of all the information that is available (exploitation), however this can lead to a very narrow view of prices and no additional knowledge is gained at each time step (the amount of time a price is left unchanged). Instead prices can be purposefully set in a manner that the seller expects will bring in less revenue for the short term, but will increase knowledge of the market so that exploiting during future time steps will generate more revenue than it would have previously. The revenue lost because of this exploration is that real-time cost that has been mentioned. Excessive exploration will give a good understanding of demand and pricing but at a high cost to the business that may be unrecoverable. This catastrophic result leads to an important factor identified with dynamic pricing, which is the necessity to learn in short-time horizons. Pricing decisions need to be made with very little data so as to avoid the passage of too much time. Prices can not be adjusted forever and there comes a point when the retailer needs to maximize revenue. This necessitates a balance between exploring and exploiting. With the right amount of exploration, revenue may be higher than if there was no exploration done, but the time given for learning cannot be long because the real-time cost would be too high.

Other factors must be considered when changing prices. If the goal of the seller is to maximize revenue for the current time step, then an exploitation only strategy should

be chosen. If however, the seller wants to maximize total revenue over several time steps into the future, then policies can be implemented that will initially favor exploration and improve the understanding of prices and demand.

In summary of the dynamic pricing problem we assert that a good dynamic pricing algorithm will do the following:

- Maximize the revenue in the given number of time periods,
- Minimize the cost of exploration while learning as much as possible and
- Detect when demand has changed and make adjustments as necessary because of this change.

In order to solve this problem the following approaches have been implemented and adapted to dynamic pricing:

- Kalman Filter (KF)
- Artificial neural networks (ANN)
- Particle swarm optimization (PSO)
- Genetic algorithms (GA)

Each of these approaches will be examined and their learning characteristics compared in the context of dynamic pricing.

Some terms that may be useful to the reader unfamiliar with this problem follow. *Dynamic pricing* is defined as the "buying and selling of goods in markets where prices move quickly in response to supply and demand fluctuations, pricing strategy in which prices change either over time, across consumer or across product bundles" [Jayaraman and Baker 2003]. *Demand* is a function of price and can be seen as a continuous representation of how many products will be sold. The number of items sold at a given price is known as the *quantity demanded*. *Revenue* is the amount of money acquired,

which can be found by multiplying the set price with the quantity demanded at each time step.

## 1.1 Thesis Statement

We will investigate learning in short-time horizons in the context of a commercial website where prices can be changed automatically to maximize revenue over a given amount of time. Our market models provide an environment where learning costs are measurable. Some machine learning algorithms will be altered to improve performance in this noisy, dynamic environment while others will just be employed. We will compare well-known methods with our new algorithms and with an existing dynamic pricing strategy. We will show that our new algorithms increase total average revenue when compared to the existing algorithms.

## 1.2 Methods

This section discusses how short term learning with measurable cost will be analyzed. First the testing environment will be presented, including simple models that simulate demand. After that the algorithms to be applied to dynamic pricing, along with relevant previous work, will be given.

### 1.2.1 The Pricing Environment

Depending on the product that is being sold, there are distinct models for demand that can be used. It is important to have more than one model so that we can compare the pricing algorithms and solutions in different situations. An ideal situation would be to find a pricing algorithm that will perform well regardless of the demand model.

Demand will not be directly observable by the learning algorithms. Instead these algorithms will query the economic system with a price and receive a quantity demanded in

return. It should be noted that the goal is not to maximize quantity demanded *which could be done by setting the price to zero*, but to maximize revenue:

$$r = pd \tag{1.1}$$

where $p$ is the price and $d$ is the quantity demanded. The simplest demand model that will be considered is linear:

$$d = \alpha + p\beta + \epsilon. \tag{1.2}$$

In the above equation $\alpha$ and $\beta$ are demand parameters affecting the slope and elasticity (defines how much a change in price will change the quantity demanded). These parameters describe how much demand there is and which price would be optimal. Noise is represented by $\epsilon$. The noise represents small changes in the curve on an irregular basis. This corresponds to all factors that affect the demand such as money spent on advertising, competitors advertising, seasonal shifts in demand, etc. Note that one difficulty can be seen here, if only a few data points are available it is hard to figure out $\alpha$ and $\epsilon$.

Building on the linear model is the log-linear [Kalyanam 1996] model:

$$d = e^{(\alpha + p\beta + \epsilon)}. \tag{1.3}$$

In running the experiments, several variations on these models will be used. The different versions of the above models that will be used are:

1. Constant demand - Using the models as defined above.
2. Random walk - the parameters for the models change with time, numbers drawn from a Gaussian distribution with a mean of zero will be added to the parameters at each time step.

3. Random walk with overall trend - The mean for the Gaussian distribution for each parameter will be positive or negative depending on if an upward or downward trend is desired.
4. Random walk with intervention - sudden shift in demand, a large number will be added to each parameter at one time step during the simulation.

A model where two products are related is also used to test some of the learning algorithms in a more difficult setting. This multi-product model is more difficult for two reasons. First, the pricing space is larger. Instead of tracking one product and revenue, various prices need to be set. Additionally, adjusting one price not only affects the quantity demanded for that particular product, but also affects the quantities demanded of all other related products. There is a large growth in the number of parameters in the model as the number of products increase. The number of parameters is equal to $(n)(n+1)$ where $n$ is the number of products being modeled.

### 1.2.2 Pricing Solutions

This section presents the various approaches to be applied to the dynamic pricing problem. It will briefly introduce various ideas as well as discuss some of advantages and disadvantages of each.

#### Kalman Filter

Using a Kalman filter for the solutions assumes that the problem can be considered as a Markov Decision Process (MDP). MDPs have long been used for solving problems in speech recognition, natural language processing and bioinformatics. Hidden Markov Models (HMMs) and MDPs have also been recognized to have applications in economics and finance [Bhar and Hamori 2004]. One advantage of using a Markov model is that the hidden states in HMMs and MDPs (actually Partially Observable Markov Decision Processes) map well to the hidden variables in many economic situations. A disadvantage of using

MDPs is that there is a need for an internal representation of demand, presenting the challenge of creating an accurate model for demand.

Previous work in this area has been done by Harvey who sets up his model and then uses a Kalman filter to track the parameters of the demand [Harvey 1989]. Carvalho and Puterman [2003] improve on this idea and estimate the value of different prices using a one-step look-ahead function and then choose the price with the highest potential future revenue. This thesis will use a different method to cause the Kalman filter to explore and then compare the results from this new algorithm with the results from Harvey's and Puterman's algorithms.

### Artificial Neural Networks

ANNs are modeled after the way the brain works. Neurons in the brain collect, process and broadcast electrical signals to other nearby neurons. The brain's capacity for processing information is believed to come from networks of neurons [Cowan and Sharp 1988]. In order to create an artificial neural network a mathematical model of the neuron has was created that fires when a linear combination of its inputs passes some threshold [McCulloch and Pitts 1943]. ANNs are made up of multiple layers of these artificial neurons and are good function approximators that work well in noisy environments [Russell and Norvig 2003]. Some work has been done previously in using ANNs to forecast economic series [Kaastra and Boyd 1996].

In order to get the ANN to where it can learn demand and set prices, some training data will need to be produced. This neural network can then be queried and the price that maximizes revenue according to the neural network will be chosen. Some method of exploration will likely need to be included so that the neural network can detect changes to the function with time. One reason for using ANNs for this problem is to attempt to provide a solution where no prior knowledge of demand is necessary. The hope is that even with little knowledge of a product's particular demand neural networks will still work well.

### Particle Swarm Optimization

The pricing problem lends itself naturally to the optimization approach in which some unknown function needs to be maximized. Particle Swarm Optimization is based on bird flocking [Kennedy and Eberhart 1995]. In this approach each particle will be assigned a random starting price within a given price range. Each price will be set for one time period. After all particles have obtained revenue values for their price, they communicate the observed results and calculate new trial prices based on individual and global information. This algorithm lends itself naturally to exploration and has no explicit representation of demand. Like neural networks, PSO will have no need of prior knowledge of demand, unlike ANNS, though, this algorithm is naturally exploring.

### Genetic Algorithms

Another optimization algorithm that will be analyzed is a Genetic Algorithm (GA) [Holland 1975]. Genetic Algorithms use ideas taken from biology to search out an optimal or near optimal solution. Genetic Algorithms have been used for financial problems before, such as portfolio tracking in the stock market [Shapcott 1992]. Our use of GAs will require prior knowledge of the problem similar to the MDP approach. Exploration will be controlled with the parameters influencing mutation and crossover rates, among others.

## 1.2.3 Exploration versus Exploitation

Throughout this study the question of exploration and exploitation will be evaluated. This idea has been studied extensively in literature previously [Pant et al. 2002; Carvalho and Puterman 2003; Zhang et al. 2003; Warmuth et al. 2003]. We feel that the real-time measurable cost associated with pricing will allow this idea to be effectively studied because a dollar amount is a widely understood measurement of utility. Here it should be noted that in a real-time situation we cannot truly measure the cost of exploration since we cannot find out what would have happened with a different price. The given pricing

models permit the system to be re-initialized to a given starting point, which makes it possible to figure out what that cost might be.

One interesting comparison done in this work is between algorithms that lend themselves naturally to exploration (PSO and GA) and the algorithms where exploration must be forced (KF and ANN). One approach to forced exploration in the KF and the ANN is inspired by Puterman: instead of using a function to approximate the future value of a price, we will simulate the value of setting different prices, given that our models are accurate. The price that returns the highest simulated revenue will be chosen for the next time period. Since we explore the effect of different prices on the quantity demanded until there is little or no value in doing so, we consider this an "Optimal Sampling" approach.

## 1.3 Contribution to Computer Science

This thesis will make several contributions to computer science. First, it will evaluate the effectiveness of various learning algorithms in learning with short-time horizons and measurable costs. In most cases the algorithms need to be changed to handle noisy and dynamic environments. In addition, because of the real cost, we will be able to discuss the tension between exploration and exploitation in problems in which the penalty for too much exploration is catastrophic (bankruptcy).

# Chapter 2

## Dynamic Pricing on Commercial Websites: A Computationally Intensive Approach

*Published in Proceedings of JCIS 2005, pages 1001–1004*

## Abstract

With commercial websites now selling anything that can be obtained in a brick and mortar store it would be helpful to automate the monitoring and the adjustment of prices to maximize profit. We propose a computationally intensive, simulation-based, approach to dynamic pricing in this context. In this approach we simulate the effect of various pricing strategies based on what we know about demand and choose the pricing strategy that obtains the highest revenue under this assumption. Using this method we explore the tension that exists between exploring prices to better learn demand behavior and exploiting what we have already learned.

## 2.1 Introduction

It is a tedious task to maintain prices on commercial websites selling hundreds or thousands of items. It would be better to automate the process of monitoring prices, monitoring quantities sold, and adjusting prices as needed to increase profits. We propose an optimal sampling approach to solve this problem. We estimate the long run value of each potential pricing policy to determine whether we should take advantage of what we know about demand or if we should use different prices to explore the demand curve. If we choose to explore, we will choose a price that may yield lower initial revenue but will allow us to pick better prices in the future.

An important part of any type of pricing algorithm is modeling demand. According to Kalyanam [Kalyanam 1996] a log linear model may be used to model demand (we also use the simulation based approach taken in Kalyanam's work). In this case the quantity demanded, $d$, is modeled using two parameters, $\alpha$ and $\beta$, and the equation

$$d = e^{(\alpha + p\beta + \epsilon)}, \tag{2.1}$$

where $p$ is the chosen price, $\beta$ is strictly negative, and $\epsilon$ is noise drawn from a normal distribution with a mean of zero and variance $\sigma^2$. We assume that $\alpha$ and $\beta$ change with time. At each time step we draw a new $\alpha$ and $\beta$ from independent normal distributions with the old values as the mean and a variance of $\sigma^2_{\alpha\beta}$ (a random walk). We use a Kalman filter to track these variables as suggested by Harvey [Harvey 1989], however we use the

notation from Russell [Russell and Norvig 2003]:

$$
\begin{aligned}
\mu_{t+1} &= F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t) \\
\Sigma_{t+1} &= (I - K_{t+1}H)(F\Sigma_t F^T + \Sigma_x) \\
K_{t+1} &= (F\Sigma_t F^\top + \Sigma_x)H^T(H(F\Sigma_t F^T + \Sigma_x)H^T + \Sigma_z)^{-1} \\
x_t &= [\alpha_t, \beta_t]^T \\
H &= [1, p]^T \\
F &= I
\end{aligned}
$$

where $\mu$ is the mean of $\alpha$ and $\beta$, $\Sigma_0$ is the prior covariance of $\mu$, $\mu_0$ is the prior mean of $\mu$, $z$ is the log of the observed quantity demanded, $\Sigma_z$ is the variance of $z|\alpha, \beta$, $F$ is the system transition matrix and $\Sigma_x$ is the covariance of $x_{t+1}|x_t$. The Kalman filter attempts to estimate a hidden state of a system based on noisy observations over a period of time.

Using the filtered value of $\beta$, $\hat{\beta}$, we can attempt to maximize revenue, $pd$, by setting $p = -1/\hat{\beta}$. This approach tends to select prices in a narrow range. If the initial estimate of $\beta$ is incorrect we will not see much information that would allow us to discover our error (slopes, such as $\beta$, are difficult to estimate when all of the data points are close to each other). Selecting a price away from $-1/\hat{\beta}$ would help us better estimate the slope, but potentially at the cost of selecting a sub-optimal price in the current time step.

The Kalman filter provides a covariance matrix ($\Sigma_t$ which we will refer to from now on as $\Sigma_{\alpha\beta}$) in addition to means for $\alpha$ and $\beta$ ($\mu_t$ which we will break into its individual components, $\hat{\alpha}$ and $\hat{\beta}$). These values model our understanding of the true $\alpha$ and $\beta$. We can use these values to simulate what we expect to occur in the next time step. We can likewise simulate what will happen in all time steps through the end of the time period we are interested in.

In our dynamic pricing algorithm we run this simulation for each price under consideration. Initially our algorithm will simulate the expected results under three possible scenarios:

1. Pick the optimal price ($p = -1/\hat{\beta}$) at all time steps
2. Pick a low price for the next time step and then use the optimal price for all of the subsequent time steps
3. Pick a high price for the next time step and then use the optimal price for all of the subsequent time steps

Once these simulations are completed, the algorithm simply selects the pricing choice that yields the largest long term revenue. Since we explore the effect of high or low prices on the quantity demanded until there is little or no value in doing so, we consider this an "Optimal Sampling" approach.

We acknowledge that this direct simulation approach is computationally intensive. We assume that commercial websites are constructed to meet peak or near peak demand and that there is ample off-peak computational capacity to re-assess pricing.

Through this approach we quantify the tension between setting the price in a way we believe will maximize the current time period's revenue and setting the price to some other value in order to learn more about our target market and thereby increase the revenue in future time periods. Another solution to the dynamic pricing problem is to analytically estimate the future value by looking one step into the future and scaling the effect to estimate the effect on future time periods [Carvalho and Puterman 2003]. This approach relies on the correct estimation of the scaling and other parameters.

## 2.2 Algorithm

For purposes of explanation we present the algorithm in two parts. Algorithm 2.1 contains the main logic of the algorithm. This algorithm uses Algorithm 2.2 ($UsePthenExploit(p)$)

to run a Kalman filter and return the expected revenue using the argument $p$ for the price in the first time step and the estimated optimal price for all subsequent time steps.

Algorithm 2.1 begins by obtaining the actual quantity demanded from the previous time step. This value is passed through a Kalman filter to produce filtered estimate of $\alpha$ and $\beta$, $\hat{\alpha}$ and $\hat{\beta}$, and a corresponding covariance matrix $\Sigma_{\alpha\beta}$. These values encode all that can be inferred about the true values of $\alpha$ and $\beta$ given the current observation of the quantity demanded, all previous observations of quantity demanded, and any prior knowledge of $\alpha$ and $\beta$. Since Algorithm 2.1 will simulate expected revenues, we must run the simulation multiple times to produce acceptably accurate estimates of the expected revenue (see the loop starting at line 4). This loop simulates new values of $\alpha$ and $\beta$ and then accumulates expected revenue under three polices. We calculate $rev_{optimal}$ by assuming that the estimated optimal price is used for all time periods, including the first one. We calculate $rev_{pmin}$ by assuming that the estimated optimal price is used for all time periods, *except* during the first time period when a low price is used ($p_{min}$, set by the administrator as a lower bound on the price). We calculate $rev_{pmax}$ by assuming that the estimated optimal price is used for all time periods, *except* during the first time period when a high price is used ($p_{max}$, set by the administrator as a upper bound on the price). Algorithm 2.1 concludes by returning the price, $-1/\hat{\beta}$ (the estimated optimal price), $p_{min}$, or $p_{max}$ which yielded the maximum simulated long-term revenue.

Algorithm 2.2 begins by creating a new instance of the Kalman filter (called KF). This filter will be used to simulate the computation of the optimal price in future time periods. The loop starting at line 2 of Algorithm 2.2 simulates the behavior of the market and pricing system for all time periods from the current time up to the end of the period of interest. Note that the price passed in as an argument is used as the price for the first time period. After the first time period the demand is simulated and used to update KF. We then use KF to produce an estimate of $\beta$ which is in turn used to compute an expected optimal price (see lines 10-14).

**Algorithm 2.1** Simulating to choose the best price

Get quantity demanded from previous time period
Use the Kalman filter to compute $\Sigma_{\alpha\beta}$, $\hat{\alpha}$ and $\hat{\beta}$
$rev_{pmax}, rev_{pmin}, rev_{optimal} \leftarrow 0$
**for** $j = 0$ to $j < NumberOfDraws$ **do**
    // Simulate the next $\alpha$ and $\beta$
    $\alpha_{temp} \sim N(\hat{\alpha}, \Sigma_{\alpha\beta})$
    $\beta_{temp} \sim N(\hat{\beta}, \Sigma_{\alpha\beta})$
    // Calculate Simulated Revenue using Algorithm 2.2
    $rev_{optimal} \leftarrow rev_{optimal}$+ ***UsePthenExploit***$(-1/\hat{\beta})$
    $rev_{pmax} \leftarrow rev_{pmax}$+***UsePthenExploit***$(p_{max})$
    $rev_{pmin} \leftarrow rev_{pmin}$+ ***UsePthenExploit***$(p_{min})$
**end for**
Set $p$ as the argument ($rev_{optimal}$, $rev_{pmax}$ or $rev_{pmin}$) with the highest total revenue

**Algorithm 2.2** UsePthenExploit(p): Simulated revenue using price $p$ for the first time period then use the estimated optimal price

Initialize a new Kalman filter (KF) for use inside of UsePthenExploit(p) using $\Sigma_{\alpha\beta}$, $\hat{\alpha}$ and $\hat{\beta}$
**for** $k = 1$ to $t <= TimePeriodsLeft$ **do**
    **if** k=1 **then**
        $price \leftarrow p$
    **else**
        $price \leftarrow -1/\beta_{KF}$
    **end if**
    //$\alpha_{temp}$ and $\beta_{temp}$ come from Algorithm 1
    $demand \leftarrow e^{(\alpha_{temp} + price * \beta_{temp})}$
    $revenue \leftarrow revenue + price * demand$
    Update KF based on simulated demand
    $\beta_{KF} \leftarrow getbeta(KF)$
**end for**
return revenue

Table 2.1: Initial Values for Variables

| Variable | Value |
|---|---|
| $\alpha$ | 9.0 |
| $\beta$ | $-1.3$ |
| $\sigma^2$ | 3.5 |
| $\sigma^2_{\alpha\beta}$ | .005 |
| $p_{max}$ | 3.0 |
| $p_{min}$ | 0.25 |

A variation of this algorithm has been created where once we exploited (used the estimated optimal price rather than either the lower or upper prices) we would always use the expected optimal price. We refer to this variant as *Once Exploit Always Exploit*. This has the benefit of running faster, since we do not need to simulate at every time step. Since we allow the true $\alpha$ and $\beta$ to vary, *Once Exploit Always Exploit* may diverge from the true $\alpha$ and $\beta$. To overcome this we can add in some random exploration. With a certain probability we choose a random price (uniform between $p_{min}$ and $p_{max}$) instead of using our knowledge of $\beta$ to choose the price optimally. We call this variant *With Random Exploration*.

Another variation is created by adding more price points to the simulation. We call this variant *Five Prices*. Instead of just choosing between an upper price, a lower price and setting the price optimally, we can have some other fixed prices in between (2 more, for a total of 5 choices in this case). This offers a nice compromise between learning and maximizing our revenue for the immediate time period. These extra price points allow us to learn something about the market without having to lose as much profit as if we used the boundary prices.

## 2.3 Simulation Setup

We have tested the behavior of our pricing algorithm using a simulated market. This simulation was run for 1000 times steps. The result presented below are averages over 4000 simulations of each of these 1000 time steps. We have constructed the simulated market using the parameters given in Table 1.

To initialize our priors for the Kalman Filter we assume that we have two quantities demanded, one for the highest allowed price and one for the lowest allowed price. We used these two data points to solve for $\alpha$ and $\beta$ using the two instances of Equation 1 assuming no noise:

$$\begin{aligned} log(d_{pmin}) &= \alpha + p_{min}\beta, \\ log(d_{pmax}) &= \alpha + p_{max}\beta, \end{aligned}$$

Our system covariance matrix ($\Sigma_x$) has the true variance ($\sigma_{\alpha\beta}$) in the diagonals, and $-\sigma_{\alpha\beta}/2$ in the off diagonals. Changing this covariance matrix within a certain amount did not have a large effect on the performance of our algorithm. The prior covariance matrix ($\Sigma_0$) was initialized as $(I\mu_0)/2$. The sensor covariance matrix ($\Sigma_z$) was initialized to $I\sigma^2$. The observation ($z_t$) is the natural log of the quantity demanded at each time step.

## 2.4 Results

Figure 2.1 shows the mean cumulative revenues for each of our variants. This is total revenue up to some time step, divided by this time step. *Optimal Pricing* refers to the revenue obtained if the true values of the parameters ($\alpha$ and $\beta$) were known at each time step. This is the best possible revenue. *Five Prices* uses the once exploit always exploit ideology with five prices instead of three. The other two policies are based on three prices. *With Random Exploration* simulates until the exploit price is chosen at which point it exploits with 99% probability and chooses the price randomly otherwise. From this graph we can see where the conflict between exploration and exploitation comes in. *Always Exploit* is the revenue obtained when we use the Kalman filter to estimate $\beta$ and exploit it at every time step, but never explore other prices nor simulate the potential revenue.

Note that if we are only setting prices for a short period of time it is best to use the *Exploit Always* policy, since this brings in more revenue short term. Note also that the

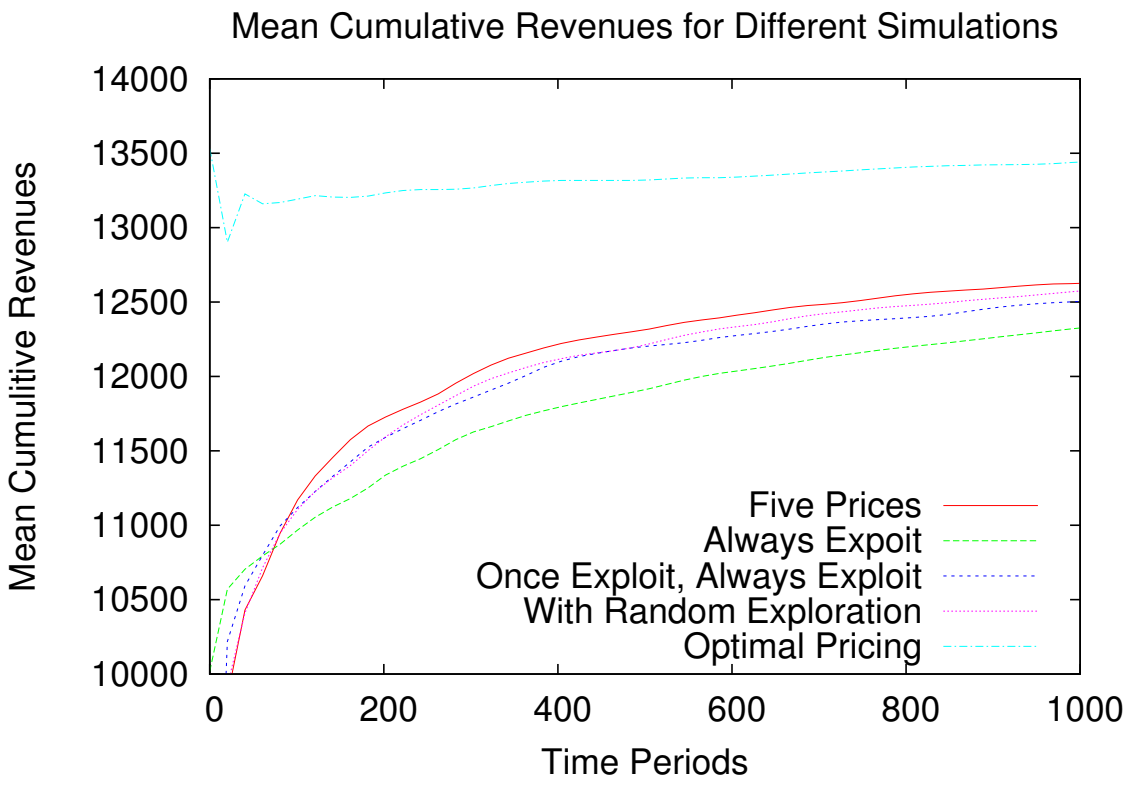


Figure 2.1: Optimal Pricing is what we reach for and Always Exploit is what we measure against.

*Always Simulate* policy is not shown on the graph because it never achieved an average revenue above 10000.

In the introduction we asserted that this computationally intensive approach to pricing was practical in the context of a commercial website. Table 2 shows the average time to run our pricing algorithm. It only took ten seconds to do a simulation for seven price points using 50 draws. Even though this is computationally intensive, it is very manageable on a server that is built to handle high volume websites.

Figure 2.2 shows a distribution of how long the simulation explored before exploiting the first time. Surprisingly 37% of the time there was no exploration the first time through the simulation. This is probably due to having good priors.

Table 2.2: Seconds to complete one simulation (300 time steps) depending on number of prices and draws.

| | Number of Draws | | |
|---|---|---|---|
| Number of Prices | **3** | **10** | **50** |
| **3** | .27 | .97 | 4.74 |
| **5** | .48 | 1.61 | 7.40 |
| **7** | .68 | 2.26 | 10.38 |

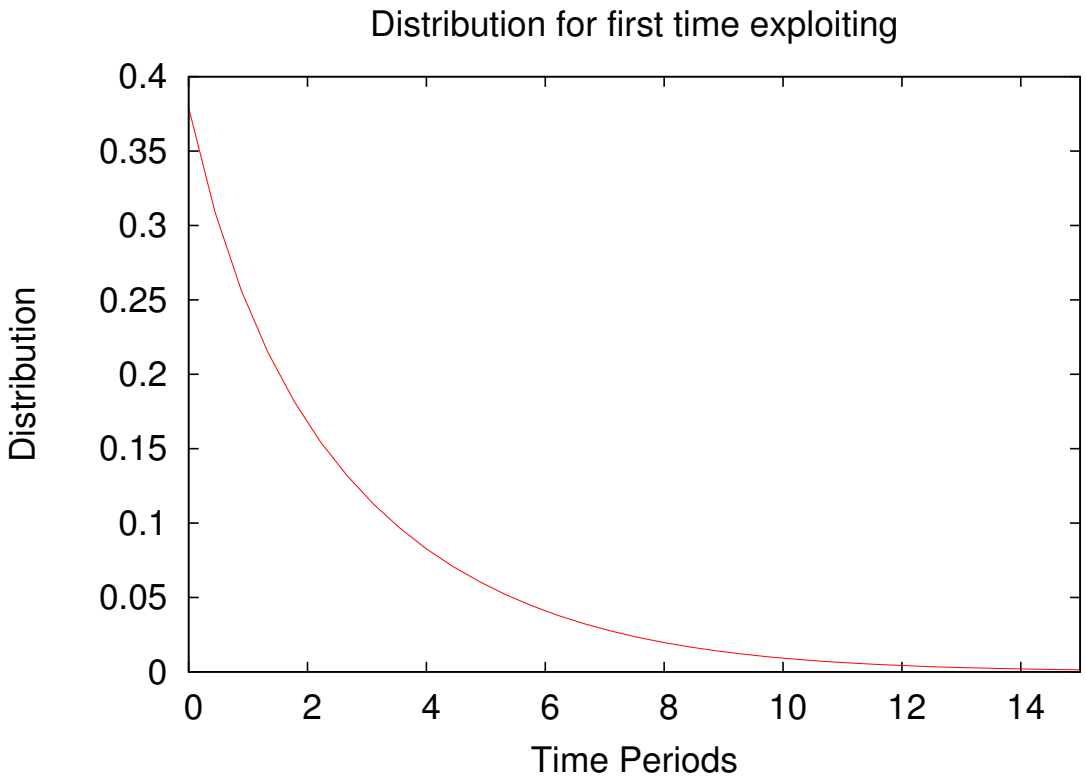


Figure 2.2: Distribution for first time exploiting based on three prices and three draws

## 2.5 Conclusion

For our experimental setup the *Five Prices* simulation performed the best. We were very disappointed to see that *Always Simulate* performed so poorly. However our modified methods do offer some improvement over using just the Kalman based estimate of the market. The *Once Exploit Always Exploit* variant yielded good gains over *Always Exploit*. Small increases were added to this by adding some random exploration (*With Random Exploration*) or by adding more price points to the simulation (*Five Prices*). We note that if we have very short term goals we may want to just apply our prior knowledge rather than exploring other price points.

We have also demonstrated that the computational resources required by our algorithm are manageable in the context of commercial website.

There are several avenues that we would like to explore in the future. The first is that our simulations assume that after the initial price exploitation we will exploit thereafter. This may not be the best policy and we would like to look at other options. In addition we may be able to quantify the number of time steps necessary to simulate. There should not be a need to run our simulation for the full 1000 time steps into the future as there is probably not much information gained after the first few. We would also like to calculate the number of prices and draws that could be done on some typical servers with average workloads.

# Chapter 3

## Particle Swarm Optimization in Dynamic Pricing

*Published in Proceedings of CEC 2006, pages 4375–4382*

## Abstract

Dynamic pricing is a real-time machine learning problem with scarce prior data and a concrete learning cost. While the Kalman Filter can be employed to track hidden demand parameters and extensions to it can facilitate exploration for faster learning, the exploratory nature of Particle Swarm Optimization makes it a natural choice for the dynamic pricing problem. We compare both the Kalman Filter and existing particle swarm adaptations for dynamic and/or noisy environments with a novel approach that time-decays each particle's previous best value; this new strategy provides more graceful and effective transitions between exploitation and exploration, a necessity in the dynamic and noisy environments inherent to the dynamic pricing problem.

## 3.1 Introduction

The popularity of the Internet as a medium for commerce creates unique opportunities to alter prices rapidly in response to changes in markets. While not responsible for its existence, this fluid medium lends increased importance to an interesting real-time learning problem known as *dynamic pricing*, "...the problem of setting prices dynamically to maximize expected revenues in a finite horizon model in which the demand parameters are unknown. [Carvalho and Puterman 2005]" In dynamic pricing, training examples are available in the form of one set of price-revenue pairs per time period.

In addition to being a real-time learning problem, dynamic pricing also has measurable associated costs that must be taken into account when determining an appropriate balance between exploration and exploitation. A pure exploitation strategy may produce good results for a time, but the dynamic nature of the environment may eventually cause its performance to degrade. Alternatively, exploration is likely to provide useful information about the true nature of the market, information that may facilitate more effective future exploitation. Because exploration and exploitation are indistinguishable from the perspective of a buyer (as both involve setting a price), liberal exploration carries with it the risk of *opportunity cost*, revenue lost at the current time period because the price was set away from the optimum.

One frequently applied approach to the problem is to model demand using a parametric model. As the true market demand is not known *a priori*, the parameters of the model are considered to be hidden; a price is chosen and a revenue observed, but the nature of the demand at every price point is generally unknown. Such a model lends itself well to Bayesian reasoning, making a case for the use of the Kalman Filter to track its hidden parameters [Harvey 1989].

The Kalman Filter is provided a prior (and generally subjective) belief about the hidden parameters, which it combines with observations to adjust and track its understanding of those parameters. This information is typically used to set a new price that optimizes

the expected revenue during the next time period. After setting the new price, the actual revenue is observed and the process is repeated. This "myopic pricing strategy" was enhanced by Carvalho and Puterman, who estimate the value of different prices using a one-step look ahead function and choose the price with the highest total expected revenue over a period of time [Carvalho and Puterman 2003].

The Kalman Filter is not unique in its ability to track hidden parameters, however, and these modifications, while allowing it to explore, do not provide compelling evidence that an optimal exploration/exploitation balance has been achieved. Because the dynamic pricing problem requires striking such a balance, PSO is an interesting alternative; it tends to naturally operate at the boundary between stability and chaos [Clerc and Kennedy 2002]. It is essentially an optimization technique based on social behavior that modifies each "particle" in a "swarm" by combining its current position in the search space $\vec{x_i}$, velocity $\vec{v_i}$, best remembered location $\vec{p_i}$, and the best location known among its neighbors $\vec{g}$ [Kennedy and Eberhart 1995]. This is typically done in the following way:

$$\vec{v_i} = \chi \left( \vec{v_i} + \varphi_1 \vec{U_1} \otimes (\vec{p_i} - \vec{x_i}) + \varphi_2 \vec{U_2} \otimes (\vec{g} - \vec{x_i}) \right) \tag{3.1}$$

$$\vec{x_i} = \vec{x_i} + \vec{v_i} \tag{3.2}$$

where each $\vec{U_i}$ is a vector whose elements are drawn from a standard uniform distribution at each time step, and the $\otimes$ operator performs element-wise multiplication. The constant $\chi$ is called the "constriction coefficient" and is calculated thus:

$$\chi = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \tag{3.3}$$

usually with $\varphi = \varphi_1 + \varphi_2 > 4$ [Clerc and Kennedy 2002].

Unfortunately, standard unmodified PSO is not suitable for application to static pricing problems due to the presence of observation noise; one abnormally favorable observation far from the optimal price can fix a particle's $\vec{p}$ (and often $\vec{g}$) to an overly opti-

mistic value, causing the entire swarm to converge quickly on that erroneous point. This occurs because PSO is essentially greedy: only the best position is remembered by each particle, and the swarm is generally attracted to the best of those (assuming the commonly used fully-connected sociometry). Noise alone is sufficient to cause problems for a particle swarm, but the situation worsens further in the presence of dynamic demand parameters. Adaptations to PSO must therefore be considered.

This work begins with a more detailed description of the dynamic pricing problem, including the formulation of a demand model. Approaches employing the Kalman Filter are then described prior to a discussion of popular adaptations of PSO to noisy and dynamic environments. This discussion motivates the creation of a new algorithm, the P-Best Decay PSO (PBDPSO). After presenting the experimental setup, the results for these various algorithms are shown and discussed.

## 3.2 Pricing Model

Various economic models are in common use, and this work will focus on Kalyanam's log-linear demand model [Kalyanam 1996]. In this model, demand is represented using the parameters $\alpha$ and $\beta$, and the equation

$$d = e^{\alpha + p\beta + \epsilon} \tag{3.4}$$

where $d$ is the quantity demanded for the chosen price $p$, $\beta$ is strictly negative, and $\epsilon$ represents Gaussian noise with parameters $\mu$ and $\sigma^2$. The noise represents small changes in demand that occur on an irregular basis and can be due to any factor that shifts the $d$, such as money spent on advertising, competitor's advertising, or seasonal shifts in demand.

Interestingly, in this simplified demand model the optimal price is $p^* = -1/\beta$ and is therefore independent of $\alpha$. It should be noted that maximization is performed over the *revenue* $r = pd$ in the pricing problem, not over the *quantity demanded* $d$ (since the only

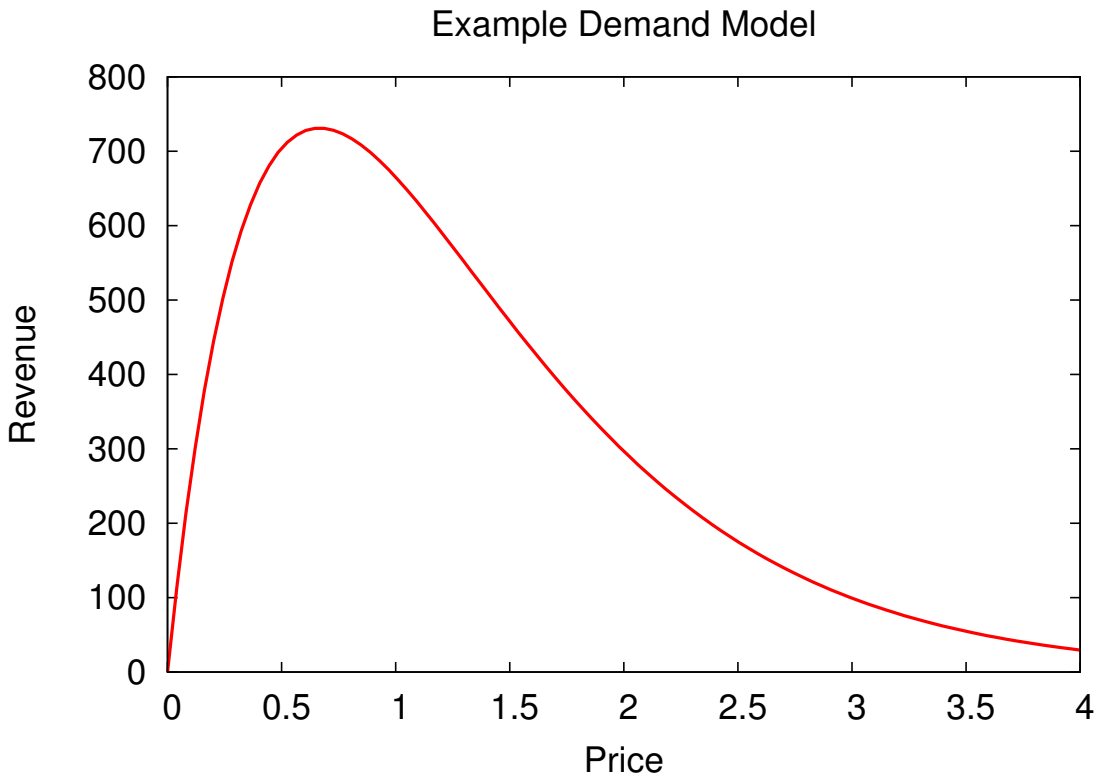


Figure 3.1: Log-linear demand model, $\alpha = 8.0, \beta = -1.5$ with no noise

requirement for maximization of the latter is a reduction of the price $p$ to 0). An example of the shape of the log-linear demand curve is given in Figure 3.1.

While simple, the log-linear model admits more sophisticated and realistic scenarios by simply assuming that $\alpha$ and $\beta$ vary with time. The following situations are of particular interest in this work:

- The parameters stay close to the original values,
- An event causes a sudden shift in the parameters, or
- A long-term trend is evident in the demand.

Results will be presented for experiments in which each of these situations is captured.

## 3.3 Kalman Filter and Pricing

The Kalman Filter estimates the hidden state of a system based on noisy observations over time using the following equations:

$$\mu_{t+1} = F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t)$$
$$\Sigma_{t+1} = (I - K_{t+1}H)(F\Sigma_t F^\top + \Sigma_x)$$
$$K_{t+1} = (F\Sigma_t F^\top + \Sigma_x)H^\top (H(F\Sigma_t F^\top + \Sigma_x)H^\top + \Sigma_z)^{-1}$$

Where $\mu = (\hat{\alpha}, \hat{\beta})^\top$ is the mean of the filtered estimates of the true demand parameters, $\mu_0$ and $\Sigma_0$ parameterize the prior Gaussian distribution over those parameters, $z$ is the log of the observed quantity demanded, $\Sigma_z$ is its covariance, $F$ is the system transition matrix, and $\Sigma_x$ is the covariance of the model parameters. Additionally, in this pricing model:

$$H = \begin{pmatrix} 1 \\ p \end{pmatrix} \qquad\qquad F = I \ .$$

This choice of $F$ and $\Sigma_x$ assumes the demand parameters $\alpha$ and $\beta$ follow a random walk.

Using the filtered estimate $\hat{\beta}$, it is possible to attempt to maximize revenue $r = pd$ by setting the price to the estimate of the optimal price $\hat{p}^* = -1/\hat{\beta}$. This approach tends to select prices within a narrow range, a strategy that provides little information about the true nature of demand; if the initial estimate of $\beta$ is incorrect, this strategy will never discover the error because $\beta$ essentially defines a slope, requiring more diverse samples to achieve a good estimate. Choosing sufficiently diverse samples, however, incurs *opportunity cost*: the difference between what *might have been* earned through exploitation (letting $p = \hat{p}^*$) and what was *actually* earned during exploration (by setting it to something else).

Carvalho and Puterman address this information issue by using a one-step look ahead function [Carvalho and Puterman 2003]:

$$F_t(p_t) = p_t e^{\alpha_{t-1}+p_t\beta_{t-1}} M_{t-1} + \frac{G(t)}{2} \frac{M_{t-1}e^{\alpha_{t-1}-1}}{\beta_{t-1}^3} \sigma_{B_t}^2(p_t) \ . \tag{3.5}$$

The price is chosen to maximize the objective function: $p_t = \arg\max_p F_t(p)$, where the first term is interpreted as the present estimated revenue and the second term defines the future estimated revenue (corresponding to minimizing the variance) given the price $p_t$. The variables $\alpha_{t-1}$ and $\beta_{t-1}$ are set to $\hat{\alpha}$ and $\hat{\beta}$ from the Kalman Filter, respectively. Furthermore, the value $M_{t-1} = e^{\sigma_{t-1}^2/2}$, where $\sigma_{t-1}^2$ is the estimate of $\sigma^2$ for time $t$ before demand is observed.

This algorithm assumes the availability of data points for $t \in \{-2, -1\}$. Even given these, however, at $t = 0$ sufficient information is lacking for a good estimate of $\sigma^2$. It is therefore initially set to a value known to be wrong for the purposes of experimentation: $\sigma_0^2 = \sigma^2/2$ at $t = 0$. The algorithm is not particularly sensitive to this choice [Carvalho and Puterman 2003].

After observing the revenue at $t = 1$, the ordinary least squares method is used to estimate $\sigma_t^2 = \frac{1}{t}[\hat{\epsilon}_{-2} + \hat{\epsilon}_{-1} + \sum_{k=1}^{2} \hat{\epsilon}_k^2$ where $\hat{\epsilon}_k = log(d_k) - \alpha_t - \beta_t p_k$, $k = -2, -1, 1, \ldots, t$. The term $G(t)$ in equation 3.5 defines the weight given to exploration, and several different functions are used in the original work[Carvalho and Puterman 2003]. In this paper the piecewise linear function is used

$$G(t) = \begin{cases} T_c - t & \text{if } t < T_c \\ 0 & \text{otherwise} \end{cases} \ . \tag{3.6}$$

Note that $T_c$ is not necessarily the end of the considered time horizon, and $G(t) = 0$ produces the myopic pricing strategy.

## 3.4 Particle Swarms and Pricing

The unmodified Kalman Filter has some obvious deficiencies in this setting. First, it does not naturally explore: in using (3.5), Carvalho and Puterman were able to address this issue to some extent, forcing the Kalman Filter to explore. Second, the Kalman Filter requires precise knowledge or assumptions about the demand function: since it estimates the demand parameters, it must know exactly how those parameters relate to the quantity demanded; success within a log-linear model will not translate into success within other models (which may have many more parameters).

These particular deficiencies are not shared by PSO, which explores naturally and does not require explicitly-stated prior information about the target function for successful operation. Assuming that the revenue curve (e.g., Figure 3.1) is generally unimodal and smooth, PSO can be expected to work well in a variety of demand environments without problem-specific tuning. Even so, standard PSO has its own deficiencies in dynamic and noisy contexts, and these must be addressed before it can be successfully applied to the dynamic pricing problem.

Standard PSO has occasionally been tested in noisy environments, particularly in comparison with other evolutionary algorithms and differential evolution on common benchmarks [Krink et al. 2004], as well as with non-linear least squares algorithms on problems of determining parameters for traditional system identification tasks [Voss and Feng 2002]. In both cases PSO was competitive with existing methodologies for solving those problems. Speciation [Parrott and Li 2004; Li 2004] is a variation of PSO that works well in dynamic environments but is tailored to multimodal functions. Charged swarms use repulsive fields to encourage exploration in dynamic settings [Blackwell and Bentley 2002]. Parsopoulos and Vrahatis claim that noisy functions allow particles to avoid local optima while converging on the global optimum. Their study, however, uses very small amounts of additive noise [Parsopoulos and Vrahatis 2001a]; the pricing problem, in contrast, is subject to large amounts of noise.

Another study by Parsopoulos and Vrahatis applies PSO in situations with higher noise levels, but PSO requires thousands of iterations to converge even on two-dimensional problems [Parsopoulos and Vrahatis 2001b]. The dynamic pricing problem requires significantly more agility than this, where convergence is expected to occur in a much smaller number of price settings or function evaluations ($< 1000$).

One adaptation stands out that improves PSO performance in noisy environments: Noise-Resistant PSO [Pugh et al. 2005]. This algorithm re-evaluates all $\vec{p}$ locations after each iteration and either averages or takes the min of the new values. It appears to work well in a robotic obstacle avoidance setting but requires too many additional function evaluations to be suitable for dynamic pricing; another approach is needed.

Two things must be detected by PSO in a noisy and dynamic environment:

- Premature convergence due to noise, and
- Environmental changes that move the global optimum.

Once detected, an appropriate response to these changes must be developed that allows PSO to track the location of the optimum [Li and Dam 2003].

### 3.4.1 Detection Methods

Several detection strategies have been developed for dynamic environments, all of which make use of a variable $N$, describing a number of iterations. One approach triggers tracking if $\vec{g}$ has not moved but its *value* has changed after $N$ iterations (NewG$_N$), while another triggers if the *location* of $\vec{g}$ has *not* changed after $N$ iterations (FixedG$_N$) [Hu and Eberhart 2002]. Yet another approach employs specialized *sentry particles* in the same manner as NewG$_N$, re-evaluating these points during search to detect changes in the function [Carlisle and Dozier 2002]; two variants of this approach involve choosing random points as sentries (Sentry$_N$) and choosing the $\vec{p}$ of a random particle as a sentry point (SentryP$_N$). A simpler but more blunt detection tool is sometimes applied, triggering a change notification after $N$ iterations without regard to the state of the swarm (Fixed$_N$).

Table 3.1: Methods used to detect changes in the pricing environment

| Label | Detection Method Description |
|---|---|
| $\text{Fixed}_1$ | No Detection Method, Response called every time |
| $\text{NewG}_{10}$ | Re-evaluate global best, check every 10 iterations, 20% threshold |
| $\text{NewG}_{20}$ | Re-evaluate global best, check every 20 iterations, 20% threshold |
| $\text{FixedG}_{10}$ | Monitor time since global best changed, 10 iterations without change triggers response |
| $\text{FixedG}_{20}$ | Monitor time since global best changed, 20 iterations without change triggers response |
| $\text{FixedG}_{30}$ | Monitor time since global best changed, 30 iterations without change triggers response |
| $\text{Sentry}_{10}$ | Sentry, check every 10 iterations, 20% threshold |
| $\text{Sentry}_{20}$ | Sentry, check every 20 iterations, 20% threshold |
| $\text{SentryP}_{10}$ | Sentry-p-best, check every 10 iterations, 20% threshold |
| $\text{SentryP}_{20}$ | Sentry-p-best, check every 20 iterations, 20% threshold |
| $\text{Fixed}_{10}$ | Fixed-iteration, 10 iterations |
| $\text{Fixed}_{20}$ | Fixed-iteration, 20 iterations |
| $\text{Fixed}_{30}$ | Fixed-iteration, 30 iterations |

In consideration of the noisy pricing environment in which the algorithms will be running, these detection algorithms have been altered. When employing $\text{NewG}_N$, $\text{Sentry}_N$, or $\text{SentryP}_N$ in noisy environments, it is highly unlikely that re-sampling a position will produce the same value twice even if the environment has not changed, making the unmodified re-evaluation technique trigger too often. Therefore, instead of merely detecting a change in value, notification is only triggered when a value change exceeds some minimum percentage of variation, here taken to be $20\%$. Additionally, the addition of the $N$ parameter is new to many of the previously described methods (with the exception of $\text{FixedG}_N$), as by default they do their evaluations after every iteration. These altered approaches are outlined in Table 3.1.

### 3.4.2 Response Methods

Response methods vary as well, but generally fall into one of two categories. The first involves recalculating the value at each $\vec{p}_i$: if the value is better at the particle's current

Table 3.2: Methods used to respond to changes in the pricing environment

| Label | Response Method Description |
|---|---|
| NoResp | No Response Method |
| Rand1 | Re-randomize 1 particle |
| Rand2 | Re-randomize 2 particles |
| Rand1+ | Re-randomize 1 particle, reset others |
| Rand2+ | Re-randomize 2 particles, reset others |
| Reset | Reset all particles |
| RandG | Re-randomize g-best |
| RandG+ | Re-randomize g-best, reset others |

position $\vec{x}_i$, then $\vec{p}_i$ is replaced with $\vec{x}_i$ (Reset) [Carlisle and Dozier 2002, 2000]. Another strategy is to re-randomize a subset of particle locations and velocities (possibly including a reset of $\vec{p}$), thereby selectively erasing some particles' memory (Rand) [Eberhart and Shi 2001; Hu and Eberhart 2002]; a variant of this method only re-randomizes particle $i$ if $\vec{p}_i = \vec{g}$ (RandG). These approaches are summarized in Table 3.2. In this table a '+' is used when the reset methodology is used in conjunction with other response techniques. For comparison purposes we also include NoResp where no changes are made to the swarm.

While interesting and useful in many dynamic situations, these approaches still suffer in the presence of noise; as mentioned above, re-evaluation of any location will almost always produce a different value. Additionally, if the locations in need of re-evaluation are far from the optimum, re-evaluation can incur significant opportunity cost with little potential for acquiring useful information. Worse still, in the pricing environment the majority of changes are gradual, so resetting the particles in the system discards all previously-obtained information about the state of the system; given the short time constraints under which the swarm is operating, throwing away so much information is unwise: the new optimum is likely to be near its previously location, implying that previous information may still have value. These issues are addressed by a new PSO variant, described below.

## 3.5 P-Best Decay PSO

The P-Best Decay PSO (PBDPSO) is designed to address the issues with adaptations of PSO for noisy and dynamic environments. It is essentially constricted PSO with the addition that the stored value $y_{\vec{p}_i}$ of each $\vec{p}_i$ is decayed by multiplying it by a decay rate $\gamma$. In the case of maximization, $\gamma$ is within the interval $(0, 1)$. PBDPSO will replace each stored $y_{\vec{p}_i}$ with $\gamma y_{\vec{p}_i}$ (thus automatically decaying $\vec{g}$) when response is triggered by any of the various detection algorithms (Table 3.1).

The assumption behind this approach is that each $y_{\vec{p}_i}$ is likely to be either abnormally favorable or invalid because the function has changed. By decaying $y_{\vec{p}_i}$, PBDPSO allows particles to be attracted to new areas of the space even though the noisy samples in those areas may not appear to be as good as the previous, potentially abnormal values. This allows the particles to simultaneously make use of previous information while discounting possibly noisy or dynamic data.

## 3.6 Experimental Setup

The experiments that follow require the selection and setting of various strategies and parameters. The true demand (as well as the Kalman demand model, whose parameters are learned over time) is log-linear, and its equation is initialized with $\alpha = 8.0$, $\beta = -1.5$, $\mu = 0$ and $\sigma^2 = 4.0$. To this model, one of the following four dynamic contexts is applied to the parameters of (3.4), described below. In all of the following scenarios, $\alpha$ and $\beta$ vary with time and have a random component represented as additive Gaussian noise, parametrized by $(\mu_\alpha, \sigma_\alpha)$ and $(\mu_\beta, \sigma_\beta)$. Unless otherwise specified, this is represented by the following equations:

$$\alpha_t = \alpha_{t-1} + N(\mu_\alpha, \sigma_\alpha) \tag{3.7}$$

$$\beta_t = \beta_{t-1} + N(\mu_\beta, \sigma_\beta) \ . \tag{3.8}$$

In all cases, $\sigma_\alpha^2 = .05$ and $\sigma_\beta^2 = .015$. The scenarios follow.

**Parameters remain near original values:**

$$\mu_\alpha = \mu_\beta = 0$$

**Large change at $t = 300$:**

$$\alpha_{300} = \alpha_{299} + 1 + N(\mu_\alpha, \sigma_\alpha)$$
$$\beta_{300} = \beta_{299} + 0.4 + N(\mu_\beta, \sigma_\beta)$$

**Overall upward trend:**

$$\mu_\alpha = 0.025 \qquad \mu_\beta = 0.0075$$

**Overall downward trend:**

$$\mu_\alpha = -0.025 \qquad \mu_\beta = -0.0075$$

Sellers usually have a prior belief about the optimal price for their products. They also know the lowest price at which they are willing to sell and generally have a reasonable estimate of a maximum supportable price in their market, and these data will be supplied to the algorithms where needed. It is also assumed that revenue observations for the latter two price points are available, supplying the Kalman Filter with needed seed values and the particle swarm with necessary initialization bounds. The upper and lower bounds are always set at $3.0$ and $0.33$, respectively.

Parameters for (3.5) are taken from Carvalho and Puterman [Carvalho and Puterman 2003], with exploration time $T_c = 30$. They found that af-

ter the exploration period is complete, some random exploration of the pricing system can improve the Kalman Filter's ability to track the parameters. Therefore, when $t > T_C$ a price is chosen according to a uniform distribution between the established bounds with probability $0.01$.

Given the short time horizon, the PSO swarm size is set to $4$ in all PSO algorithms, and a fully-connected sociometry is used. The chosen swarm size is large enough to admit testing responses that require more information while being small enough to avoid too many function evaluations. Additionally, $\varphi_1 = \varphi_2 = 2.05$ in all PSO experiments.

Each simulation proceeds for $1000$ function evaluations, and all results shown are the mean of $1000$ independent experiments. While total revenue for each algorithm is particularly important, in the interests of space the cumulative mean revenue is depicted instead, defined as $\sum_{n=1}^{t} r_n/t$, where $t$ is the current time step and $r_n$ is the revenue earned at $t = n$. For PSO, each detection method was run in conjunction with each response method, and these results are compared to the one-step look ahead function and unmodified Kalman Filter.

## 3.7 Results

In the graphs presented here, the use of a Kalman Filter is denoted "KF", and the Kalman Filter employing the one-step look ahead function is denoted "KF-OSL". The remainder of the lines are indicated using the detection and response abbreviations in tables 3.1 and 3.2. The additional notation PBDPSO$_n$ indicates that PBDPSO is applied with $\gamma = n/100$. In order to simplify the presentation, only the best-performing PSO algorithms are presented in each graph. For the curious reader more complete cumulative results are included in Tables 3.3, 3.4, and 3.5 for all experiments except those involving trends.

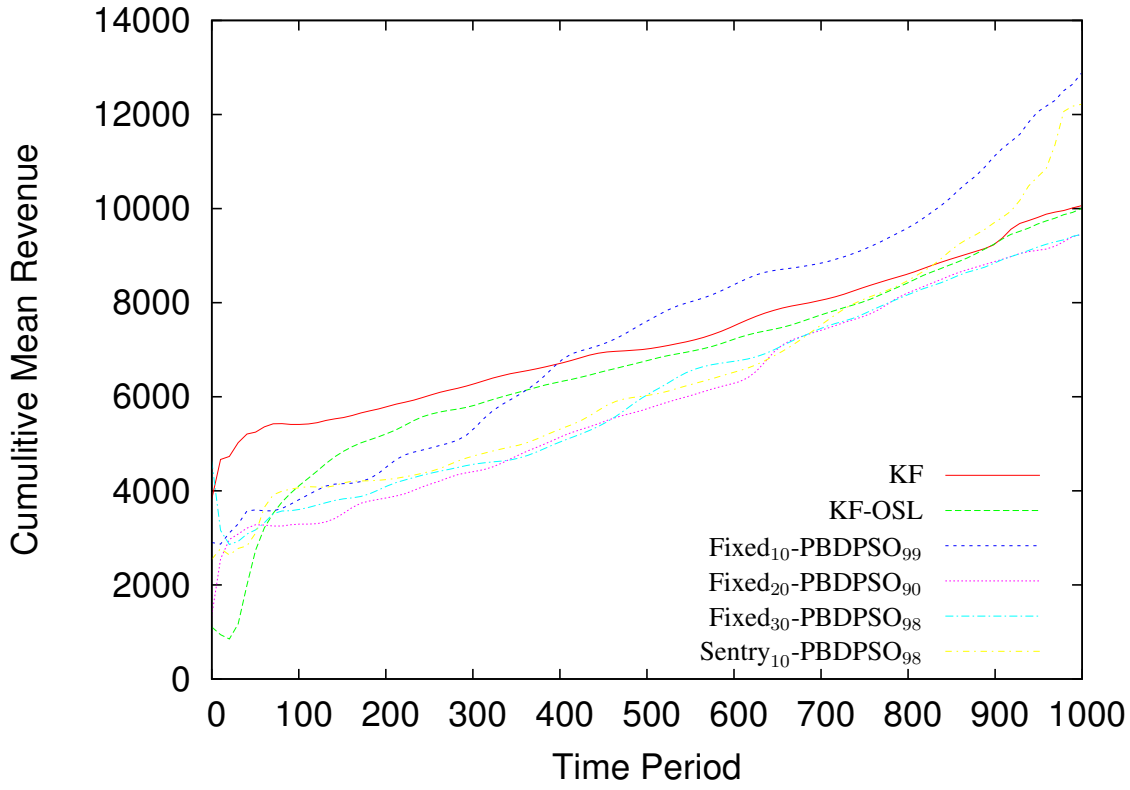


Figure 3.2: Mean cumulative revenue earned with demand parameters changing with time

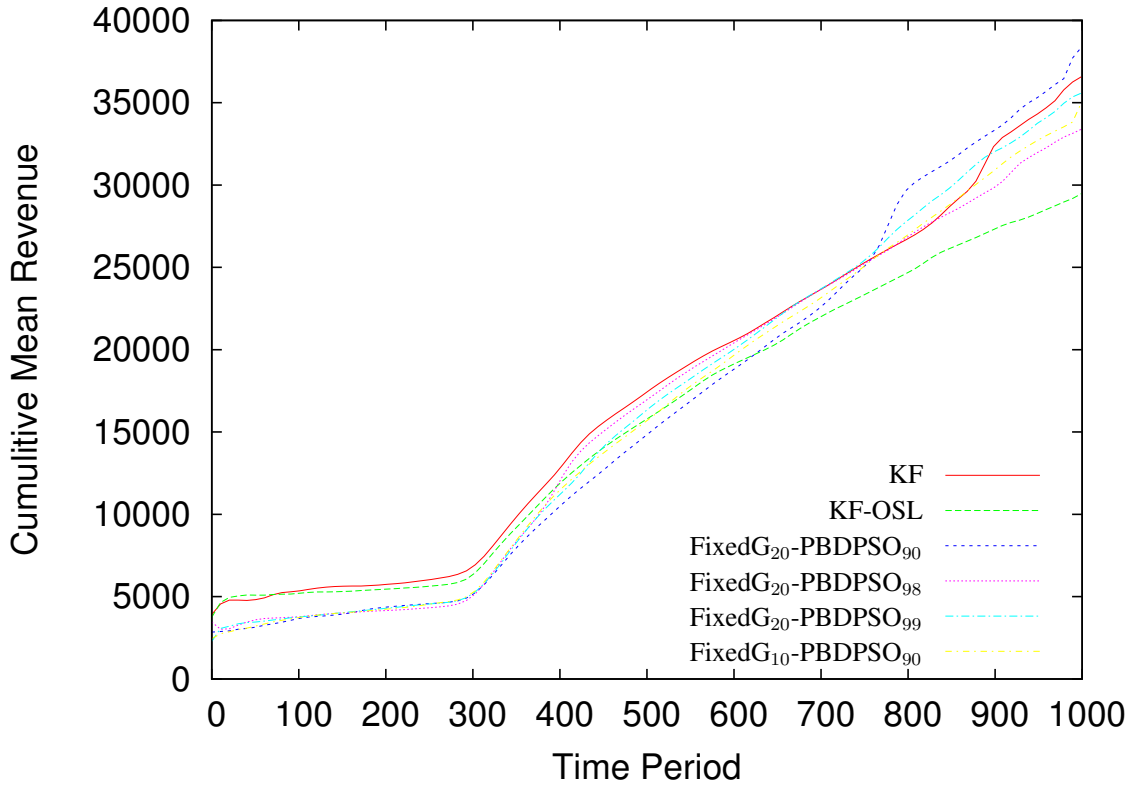


Figure 3.3: Mean cumulative revenue earned with demand parameters jumping at $t = 300$

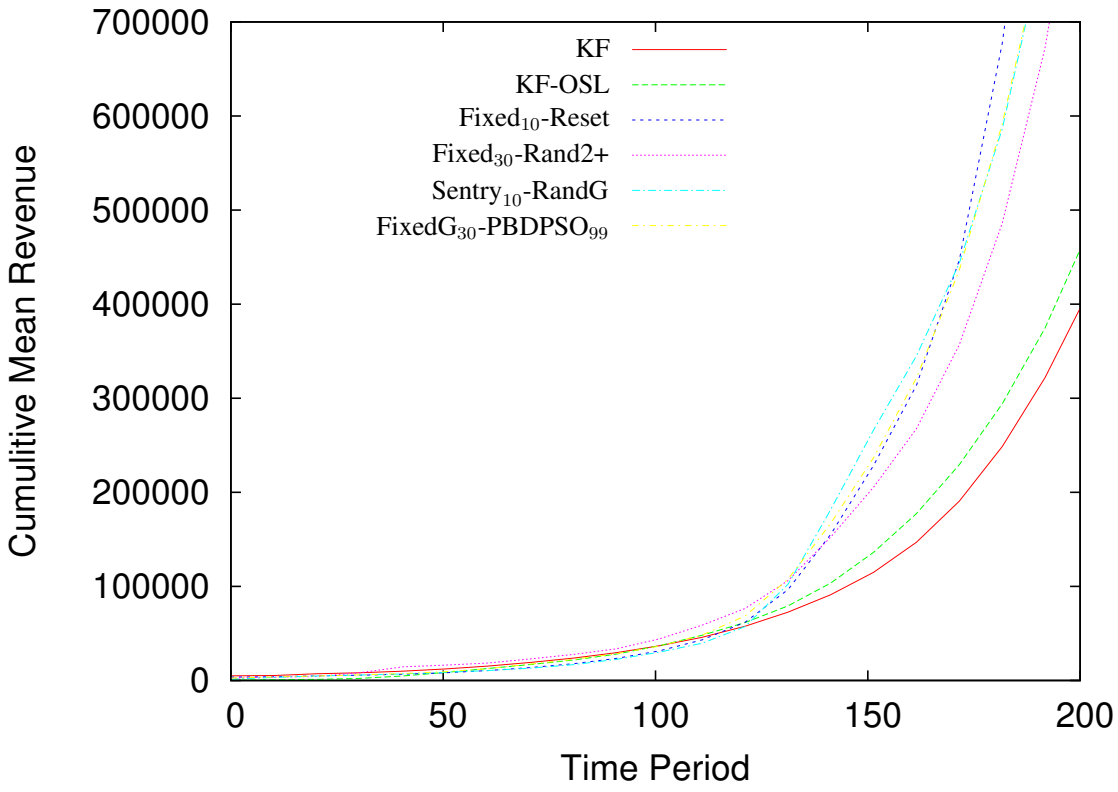


Figure 3.4: Mean cumulative revenue earned with demand parameters changing with time with overall upward trend

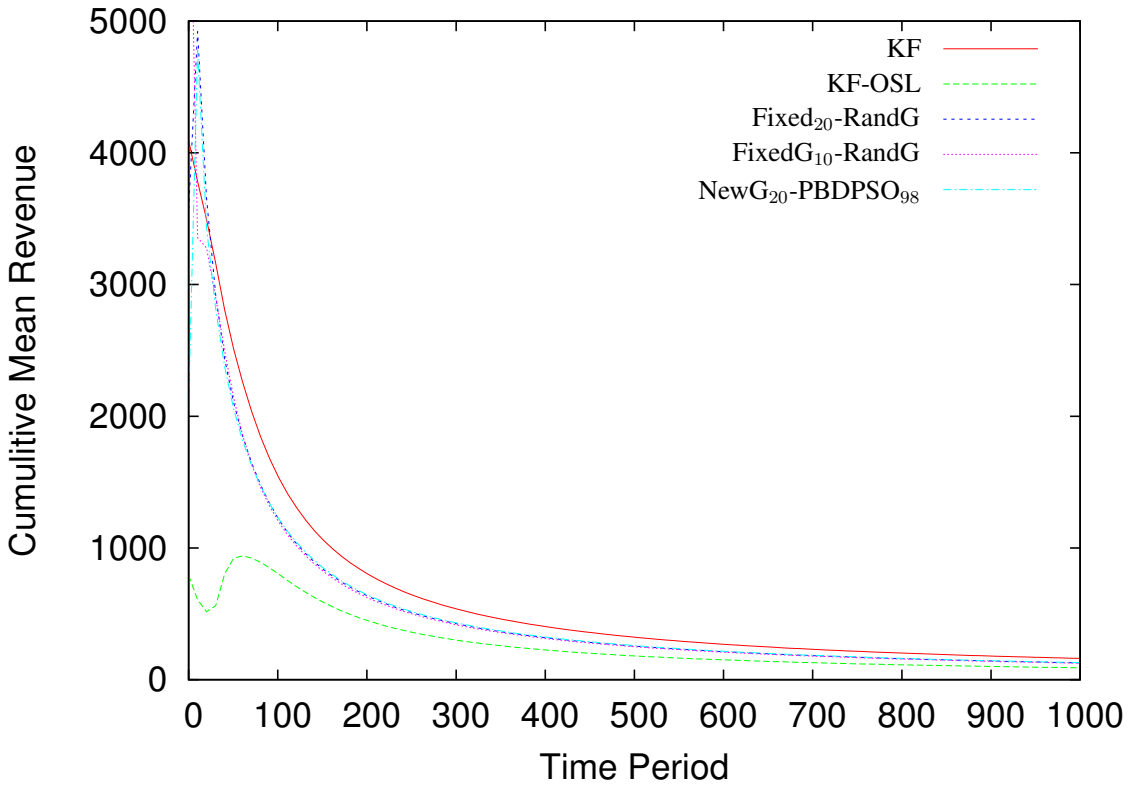


Figure 3.5: Mean cumulative revenue earned with demand parameters changing with time with overall downward trend

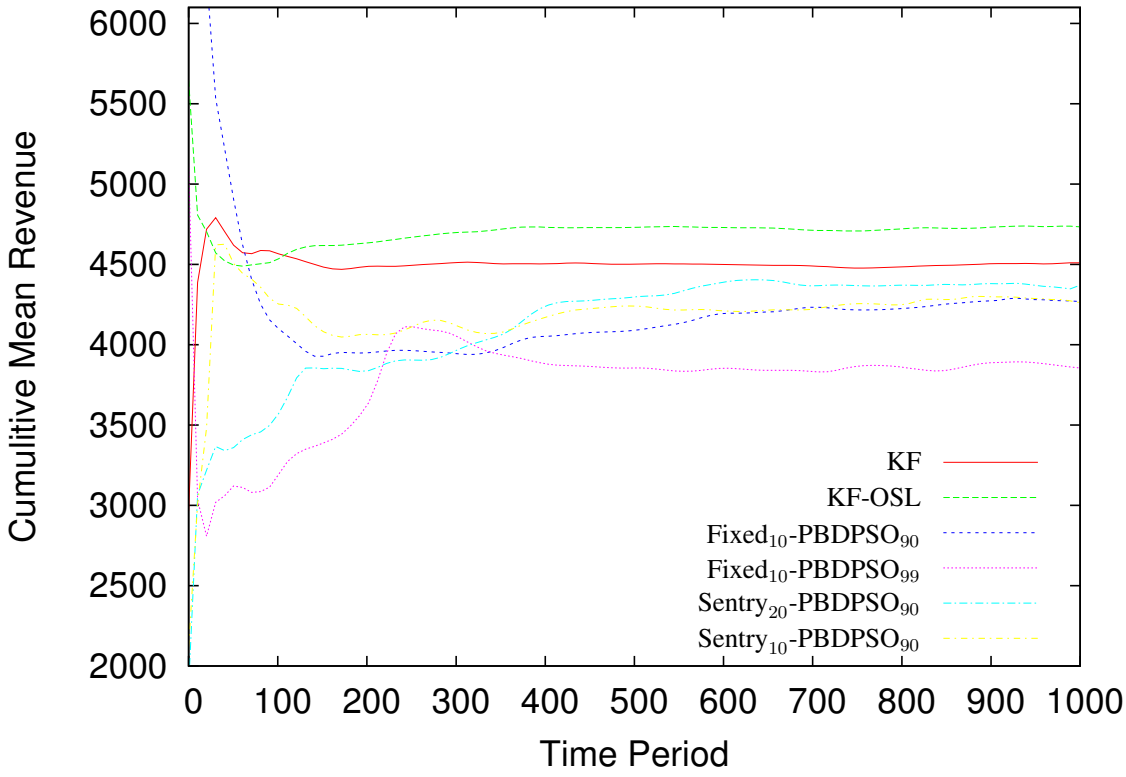


Figure 3.6: Mean cumulative revenue earned with static demand

Table 3.3: Complete PSO results for static demand. Numbers are average hundreds of dollars earned at each time period.

| | $Fixed_1$ | $NewG_{10}$ | $NewG_{20}$ | $FixedG_{10}$ | $FixedG_{20}$ | $FixedG_{30}$ | $Sentry_{10}$ | $Sentry_{20}$ | $SentryP_{10}$ | $SentryP_{20}$ | $Fixed_{10}$ | $Fixed_{20}$ | $Fixed_{30}$ | **Avg.** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NoResp | 40 | 39 | 38 | 36 | 38 | 39 | 39 | 38 | 38 | 39 | 38 | 37 | 40 | 38 |
| Rand1 | 32 | 33 | 33 | 33 | 33 | 31 | 32 | 32 | 33 | 31 | 33 | 32 | 33 | 32 |
| Rand2 | 31 | 31 | 32 | 32 | 33 | 30 | 31 | 31 | 32 | 31 | 32 | 31 | 31 | 31 |
| Rand1+ | 33 | 34 | 33 | 33 | 33 | 34 | 34 | 33 | 34 | 33 | 33 | 33 | 34 | 33 |
| Rand2+ | 32 | 32 | 32 | 32 | 32 | 35 | 33 | 33 | 32 | 32 | 34 | 32 | 31 | 32 |
| Reset | 38 | 40 | 40 | 39 | 39 | 41 | 42 | 39 | 39 | 39 | 40 | 40 | 40 | 40 |
| RandG | 29 | 29 | 31 | 29 | 30 | 28 | 29 | 27 | 28 | 29 | 29 | 30 | 29 | 29 |
| RandG+ | 31 | 31 | 31 | 31 | 31 | 31 | 31 | 30 | 32 | 30 | 30 | 31 | 31 | 31 |
| $PBDPSO_{99}$ | 41 | 38 | 39 | 41 | 37 | 39 | 42 | 41 | 39 | 42 | 39 | 39 | 40 | 40 |
| $PBDPSO_{98}$ | 39 | 39 | 41 | 41 | 41 | 40 | 39 | 41 | 40 | 40 | 43 | 38 | 41 | 40 |
| $PBDPSO_{90}$ | 41 | 42 | 39 | 41 | 40 | 42 | 41 | 44 | 41 | 40 | 43 | 41 | 41 | 41 |
| **Avg.** | 35 | 35 | 35 | 35 | 35 | 35 | 36 | 35 | 35 | 35 | 36 | 35 | 36 | |

Table 3.4: Complete PSO results for changing demand with time. Numbers are average hundreds of dollars earned at each time period.

| | $\text{Fixed}_{1}$ | $\text{NewG}_{10}$ | $\text{NewG}_{20}$ | $\text{FixedG}_{10}$ | $\text{FixedG}_{20}$ | $\text{FixedG}_{30}$ | $\text{Sentry}_{10}$ | $\text{Sentry}_{20}$ | $\text{SentryP}_{10}$ | $\text{SentryP}_{20}$ | $\text{Fixed}_{10}$ | $\text{Fixed}_{20}$ | $\text{Fixed}_{30}$ | **Avg.** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NoResp | 75 | 60 | 67 | 104 | 70 | 67 | 77 | 98 | 74 | 93 | 92 | 76 | 86 | 80 |
| Rand1 | 62 | 59 | 69 | 61 | 76 | 76 | 72 | 78 | 68 | 72 | 80 | 62 | 66 | 69 |
| Rand2 | 76 | 69 | 60 | 74 | 65 | 54 | 95 | 58 | 70 | 55 | 79 | 64 | 48 | 67 |
| Rand1+ | 79 | 69 | 81 | 64 | 63 | 62 | 61 | 60 | 74 | 67 | 71 | 89 | 66 | 70 |
| Rand2+ | 73 | 56 | 73 | 86 | 62 | 57 | 95 | 76 | 68 | 53 | 69 | 68 | 54 | 68 |
| Reset | 72 | 100 | 72 | 78 | 72 | 75 | 76 | 89 | 70 | 78 | 65 | 78 | 87 | 78 |
| RandG | 73 | 53 | 84 | 55 | 69 | 60 | 72 | 62 | 80 | 77 | 65 | 72 | 81 | 70 |
| RandG+ | 73 | 71 | 54 | 68 | 82 | 62 | 63 | 63 | 93 | 51 | 82 | 62 | 75 | 69 |
| $\text{PBDPSO}_{99}$ | 74 | 83 | 103 | 102 | 84 | 87 | 87 | 77 | 95 | 73 | 129 | 75 | 96 | 90 |
| $\text{PBDPSO}_{98}$ | 66 | 94 | 114 | 90 | 101 | 87 | 72 | 82 | 69 | 91 | 96 | 70 | 95 | 87 |
| $\text{PBDPSO}_{90}$ | 64 | 92 | 85 | 64 | 65 | 93 | 100 | 108 | 68 | 93 | 75 | 94 | 75 | 83 |
| **Avg.** | 72 | 73 | 78 | 77 | 74 | 71 | 79 | 77 | 75 | 73 | 82 | 74 | 75 | |

Table 3.5: Complete PSO results for changing demand with time and sudden shift in demand at time period 300, reported in average hundreds of dollars earned at each time step.

| | $Fixed_{1}$ | $NewG_{10}$ | $NewG_{20}$ | $FixedG_{10}$ | $FixedG_{20}$ | $FixedG_{30}$ | $Sentry_{10}$ | $Sentry_{20}$ | $SentryP_{10}$ | $SentryP_{20}$ | $Fixed_{10}$ | $Fixed_{20}$ | $Fixed_{30}$ | **Avg.** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NoResp | 289 | 303 | 327 | 326 | 294 | 312 | 341 | 320 | 299 | 306 | 345 | 323 | 313 | 315 |
| Rand1 | 256 | 259 | 287 | 257 | 254 | 267 | 271 | 255 | 247 | 257 | 279 | 251 | 259 | 262 |
| Rand2 | 266 | 288 | 242 | 272 | 243 | 277 | 286 | 287 | 329 | 257 | 264 | 281 | 277 | 274 |
| Rand1+ | 313 | 279 | 265 | 270 | 289 | 307 | 269 | 253 | 250 | 300 | 269 | 247 | 296 | 277 |
| Rand2+ | 283 | 266 | 264 | 280 | 275 | 264 | 288 | 288 | 278 | 256 | 239 | 274 | 245 | 269 |
| Reset | 309 | 283 | 291 | 279 | 322 | 297 | 300 | 322 | 309 | 288 | 276 | 312 | 296 | 299 |
| RandG | 256 | 247 | 231 | 234 | 262 | 236 | 263 | 256 | 275 | 250 | 237 | 245 | 242 | 249 |
| RandG+ | 280 | 256 | 270 | 258 | 285 | 263 | 259 | 264 | 247 | 276 | 262 | 255 | 254 | 264 |
| $PBDPSO_{99}$ | 351 | 314 | 291 | 309 | 356 | 320 | 325 | 311 | 310 | 318 | 291 | 312 | 310 | 317 |
| $PBDPSO_{98}$ | 342 | 344 | 323 | 320 | 334 | 308 | 340 | 326 | 285 | 326 | 297 | 317 | 319 | 322 |
| $PBDPSO_{90}$ | 325 | 307 | 309 | 350 | 384 | 327 | 312 | 313 | 336 | 297 | 302 | 332 | 330 | 325 |
| **Avg.** | 297 | 286 | 282 | 287 | 300 | 289 | 296 | 290 | 288 | 285 | 278 | 286 | 285 | |

PSO is able to overtake and surpass the performance of the Kalman Filter in the majority of the dynamic environments. Figure 3.2 indicates that the Kalman filter is able to quickly identify the demand parameters, but once the parameters begin to deviate from their original values the particle swarms are better equipped to adapt. The best response algorithms in this scenario are those employing the PBDPSO as a response methodology. The best-performing detection algorithms are the Fixed$_N$ and Sentry$_N$ methods.

Figure 3.3 depicts the results of applying various algorithms to "sudden shift" scenario, where parameters change sharply at $t = 300$. In this case FixedG$_N$ is the best detection method and PBDPSO provides the best response. It is interesting to note that immediately after the parameter shift the particle swarm quickly overtakes the Kalman Filter algorithms.

In the presence of a constant upward trend, PSO significantly outperformed the Kalman Filter variants, as shown in Figure 3.4. Different response methods appear to be appropriate in this setting than in the previous experiments. Among response methods, the top performers are variations on Rand and Reset, with PBDPSO remaining highly competitive. When tested on the downward trend scenario, Figure 3.5 shows that PBDPSO does no better nor worse on average than any of the other top performers. In fact, none of the top performers are easily distinguished in this scenario.

Figure 3.6 displays the results in a noise-free and purely static demand context. In this scenario, PBDPSO is unable to approach the performance of the either of the Kalman Filter variants. It does, however outperform the other response algorithms in this setting.

Table 3.3 shows the result of applying the PSO variants to a static demand scenario, and has some interesting characteristics. In general, PBDPSO is the best response method applied, but unmodified PSO (Fixed$_1$-NoResp) performs surprisingly well in comparison.

## 3.8 Conclusions

PBDPSO performs well against the Kalman Filter variants as well as outperforming some existing PSO adaptations for noise and dynamic functions, except in the static scenario. While Carvalho and Puterman developed the one-step look ahead Kalman variant with the express purpose of improving performance in a dynamic setting, their published results are limited to a noisy but purely static environment. Unchanging demand parameters, while providing a good algorithm testing ground, do not represent a realistic pricing scenario, and this is the only setting in which the Kalman Filter variants dominate the results.

The results in this work indicate that PBDPSO is a well-rounded algorithm for application to the dynamic pricing problem. The exploratory nature of particle swarms, especially with the proposed adaptations, allows them to track changes in this noisy and dynamic system, thereby earning higher total revenues over time.

That no PSO variant's performance is distinguishable from that of the other methods outlined here for the case with a downward trend is unsurprising after deeper consideration. Even with the proposed decay method, PSO remains a fundamentally greedy algorithm: it continues to favor attractors with higher value, and therefore struggles with functions whose maximum is constantly decreasing. In an attempt to improve the performance of PBDPSO in this situation, more aggressive detection and response methods have been implemented, including lower thresholds and iteration constants for the detection methods, and larger re-initialized subsets and decay values as low as $0.75$ in the response methods. None of these approaches improved performance. Additionally, drawing random $\varphi_i$ values from a uniform distribution on the interval $[1, 3]$ at each time step failed to improve its ability to track a downward trend. We also implemented charged swarms [Blackwell and Bentley 2002] which reach a nice balance between exploring and exploiting. The results were competitive but did not earn enough simulated revenue to be included in the results.

Many promising directions are under consideration for future research. The first relates to the dynamic pricing problem itself. More experiments are needed in the downward

trend scenario to determine whether any kind of particle swarm may be effectively applied; instead of completely re-randomizing particles, it may help to retain their current positions while randomizing only their velocities. The problem of the downward trend is especially interesting because improvements to PSO in that environment are likely to extend to the others; noise as defined in these experiments is symmetric about the true revenue, and PBDPSO was designed exclusively for the overly optimistic case. That noise represents abnormally favorable *and* abnormally unfavorable values suggests that improvement in the presence of a downward trend will translate to improvement elsewhere.

Another opportunity for future research is a more direct and less application-centric comparison of PBDPSO with other PSO variants designed to cope with noisy or dynamic functions.

## Acknowledgments

This work was supported in part by the BYU bookstore and the Roll ins Center for eBusiness at Brigham Young University.

# Chapter 4

## Dynamic Pricing with Artificial Neural Networks

*Published as a tech report, Applied Machine Learning Laboratory, Department of Computer Science, Brigham Young University, September 2006*

## Abstract

Dynamic pricing is a machine learning problem with scarce data and a real-time learning cost. The Kalman Filter has been employed successfully to track the hidden demand parameters and extensions to it can facilitate exploration for accelerated learning. Using a Kalman filter, however, requires prior knowledge about the nature of the demand curve. Artificial Neural Networks perform well as function approximators in noisy environments and do not require that same prior knowledge. This paper explores the performance of neural networks in this real-time learning environment. The results show that while the neural network does well at tracking the demand in non-dynamic situations, it has trouble with the sparsity of data in this noisy and dynamic environment.

## 4.1 Introduction

The popularity of the Internet as a medium for commerce creates unique opportunities to alter prices rapidly in response to markets changes. While not responsible for its existence, the Internet lends increased importance to an interesting real-time learning problem known as *dynamic pricing*, "...the problem of setting prices dynamically to maximize expected revenues in a finite horizon model in which the demand parameters are unknown [Carvalho and Puterman 2005]."

Dynamic pricing also has measurable associated costs that must be taken into account when determining an appropriate balance between exploration and exploitation. A pure exploitation strategy may produce good results for a time, but the dynamic nature of the environment may eventually cause its performance to degrade. Alternatively, exploration is likely to provide useful information about the true nature of the market, and this information may facilitate increased revenue by exploiting during future time steps. Liberal exploration carries with it the risk of *opportunity cost*, which is revenue lost at the current time period because the price was set away from the optimum.

One frequently applied approach to the problem is to model demand using a parametric model. As the true market demand is not known *a priori*, the parameters of the model are considered to be hidden; a price is chosen and a revenue observed, but the nature of the demand at every price point is generally unknown. Such a model lends itself well to Bayesian reasoning, making a case for the use of the Kalman Filter to track its hidden parameters [Harvey 1989].

The Kalman Filter is provided a prior (and generally subjective) belief about the hidden parameters, which it combines with observations to adjust and track its understanding of those parameters. This information is typically used to set a new price that optimizes the expected revenue during the next time period. After setting the new price, the actual revenue is observed and the process is then repeated [Kalyanam 1996]. This "myopic pricing strategy" was enhanced by Carvalho and Puterman, who estimate the value of different

prices using a one-step look ahead function and choose the price with the highest total expected revenue for a given number of time steps [Carvalho and Puterman 2003].

One drawback of using a Kalman filter to track the parameters is that prior knowledge about the demand curve is needed to setup the necessary models. It would be beneficial to have an algorithm that would work well in this environment when knowledge about the nature of the demand curve is not available. An artificial neural network (ANN) is a good function approximator in noisy situations [Russell and Norvig 2003]. ANNs are modeled on the way the brain works. Neurons in the brain collect, process and then broadcast electrical signals to other nearby neurons. The brain's capacity for processing information is believed to come from networks of neurons [Cowan and Sharp 1988]. A mathematical model of the neuron has been created that fires when a linear combination of its inputs passes some threshold [McCulloch and Pitts 1943]. ANNs are made up of multiple layers of these artificial neurons.

This work begins with a more detailed description of the dynamic pricing problem, including the formulation of the demand models. Approaches employing the Kalman Filter are then described followed by a discussion of ANNs during which an algorithm attempting to balance exploration and exploitation in ANNs is presented. After presenting the experimental setup, the results of applying these various algorithms are shown and discussed.

## 4.2 Pricing Models

Various economic models are commonly used, and this work will use two of these models for the experiments in addition to one that we created to test the robustness of the presented algorithms. The first of these models is a linear demand model, represented using the parameters $\alpha$ and $\beta$, and the equation

$$d = \alpha + p\beta + \epsilon \tag{4.1}$$

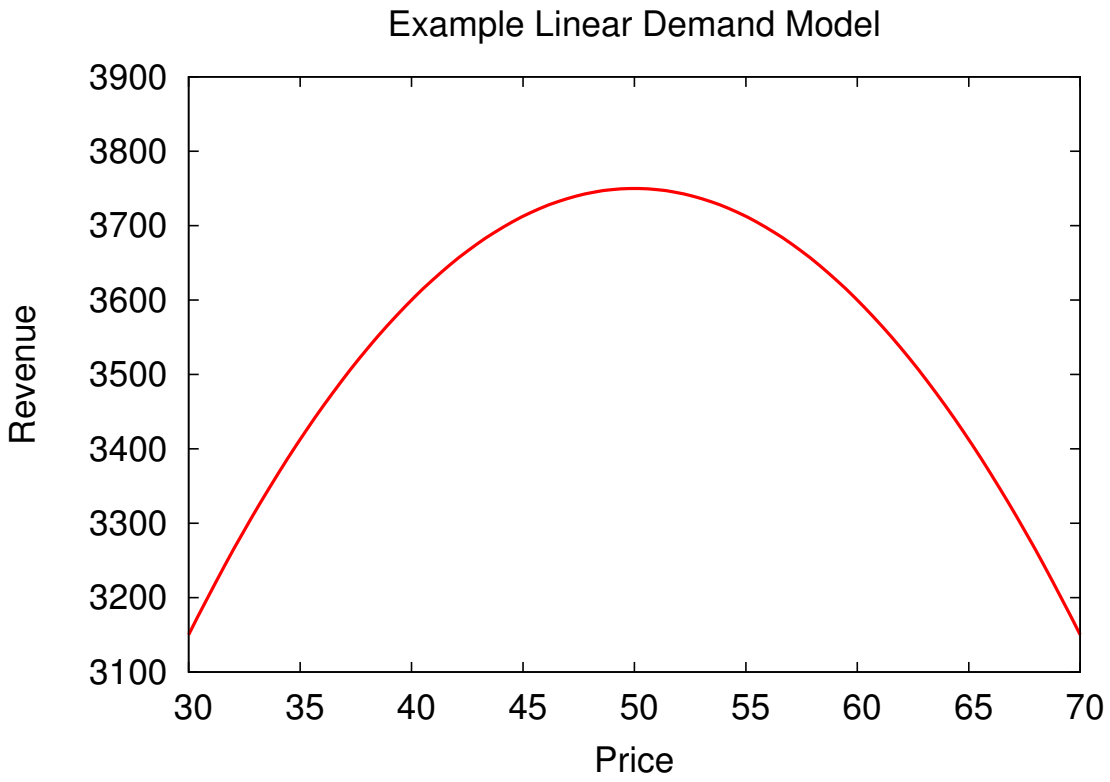


Figure 4.1: Linear demand model, $\alpha = 150.0, \beta = -1.5$ with no noise

where $d$ is the quantity demanded for the chosen price $p$, $\beta$ is strictly negative, and $\epsilon$ represents Gaussian noise with parameters $\mu$ and $\sigma^2$. The noise represents changes in demand that occur on an irregular basis and can be due to any factor that alters demand that is not accounted for in the supplied model. An example pricing model with linear demand is shown in Figure 4.1.

An additional model that is used is Kalyanam's log-linear demand [Kalyanam 1996], also using $\alpha$ and $\beta$ and the equation

$$d = e^{\alpha + p\beta + \epsilon} \tag{4.2}$$

An example of the maximization problem with a log-linear demand is given in Figure 4.2. In the log linear demand model the optimal price $p^*$ is $-1/\beta$ and calculated independent of $\alpha$. In contrast to the linear model which requires knowledge of $\alpha$ to cal-

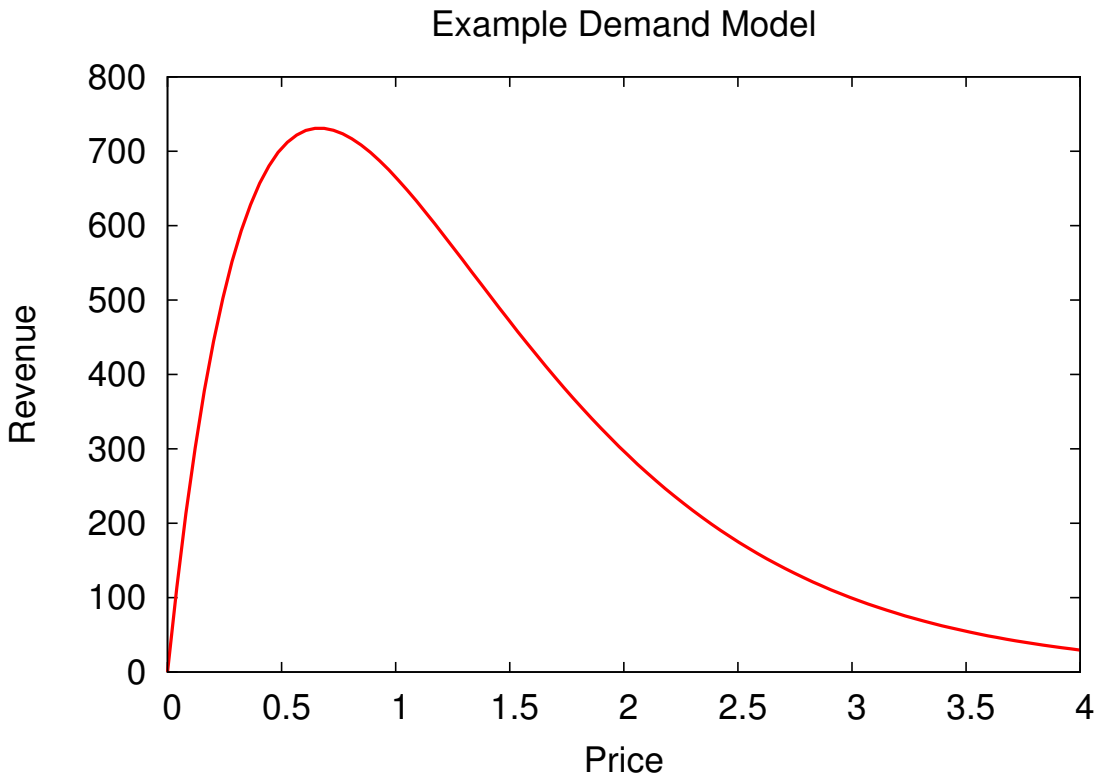


Figure 4.2: Log-linear demand model, $\alpha = 8.0, \beta = -1.5$ with no noise

culate the optimal price with ($p^* = -\alpha/2\beta$). It should be noted that the maximization is performed over the *revenue* $r = pd$ in the pricing problem, not over the *quantity demanded* $d$ (the only requirement for maximizing the latter is to reduce the price $p$ to $0$).

A third demand will also be used in a limited fashion to verify whether or not the neural networks are superior when there is little or no information about the true nature of demand. This demand is not based on any economic models but is presented here to add to the the discussion on adaptability. This will be modeled as follows:

$$q = (p + \alpha + \epsilon)^{\beta} \tag{4.3}$$

Where $q$ in this case gives the number of items purchased *per capita* and the final quantity demanded, $d$ is given by

$$d = \chi q \tag{4.4}$$

with $\chi$ being the population size. Additionally $\alpha$ must be positive and $\beta$ negative.

While simple, these pricing models admit more sophisticated and realistic scenarios by simply assuming that $\alpha$ and $\beta$ vary with time. The following situations are of particular interest in this work:

- The parameters stay close to the original values,
- An event causes a sudden shift in the parameters, or
- A long-term trend is evident in the demand.

Results will be presented for experiments in which each of these situations are simulated.

## 4.3 Kalman Filter and Pricing

The Kalman Filter estimates the hidden state of a system based on noisy observations over time using the following equations:

$$\begin{aligned}
\mu_{t+1} &= F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t) \\
\Sigma_{t+1} &= (I - K_{t+1}H)(F\Sigma_t F^\top + \Sigma_x) \\
K_{t+1} &= (F\Sigma_t F^\top + \Sigma_x)H^\top (H(F\Sigma_t F^\top + \Sigma_x)H^\top + \Sigma_z)^{-1}
\end{aligned}$$

Where $\mu = (\hat{\alpha}, \hat{\beta})^\top$ is the mean of the filtered estimates of the true demand parameters, $\mu_0$ and $\Sigma_0$ parameterize the prior Gaussian distribution over those parameters. The observation, $z$, is the quantity demanded in the case of the linear demand. When it appears that the demand may follow a log-linear curve then the *log* of the quantity demanded is used. $\Sigma_z$ is the covariance of $z$, $F$ is the system transition matrix, and $\Sigma_x$ is the covariance of the model parameters. $H$ is the sensor model and in the pricing problem:

$$H = \begin{pmatrix} 1 \\ p \end{pmatrix} \qquad\qquad F = I \ .$$

This choice of $F$ assumes the demand parameters $\alpha$ and $\beta$ follow a random walk.

Using the filtered estimate of $\hat{\alpha}$ and $\hat{\beta}$, it is possible to attempt to maximize revenue $r = pd$ by setting the price to the estimate of the optimal price $\hat{p}^*$. This approach tends to select prices within a narrow range, a strategy that provides little information about the true nature of demand; if the initial estimates of $\alpha$ and $\beta$ are incorrect, this strategy may never discover the error because $\beta$ essentially defines a slope, requiring more diverse samples to achieve a good estimate. Choosing sufficiently diverse samples, however, may incur an *opportunity cost*: the difference between what *might have been* earned through exploitation (letting $p = \hat{p}^*$) and what was *actually* earned during exploration by setting it to something else.

Carvalho and Puterman address this information issue in the log-linear demand situation by using a one-step look ahead function [Carvalho and Puterman 2003]:

$$F_t(p_t) = p_t e^{\alpha_{t-1}+p_t\beta_{t-1}} M_{t-1} + \frac{G(t)}{2} \frac{M_{t-1}e^{\alpha_{t-1}-1}}{\beta_{t-1}^3} \sigma_{B_t}^2(p_t) \ . \tag{4.5}$$

The price is chosen to maximize the objective function: $p_t = \arg\max_p F_t(p)$, where the first term is interpreted as the present estimated revenue and the second term defines the future estimated revenue (corresponding to minimizing the variance) given the price $p_t$. The variables $\alpha_{t-1}$ and $\beta_{t-1}$ are set to $\hat{\alpha}$ and $\hat{\beta}$ from the Kalman Filter, respectively. Furthermore, the value $M_{t-1} = e^{\sigma_{t-1}^2/2}$, where $\sigma_{t-1}^2$ is the estimate of $\sigma^2$ for time $t$ before demand is observed.

This algorithm assumes the availability of data points for $t \in \{-2, -1\}$. Even given these, however, at $t = 0$ sufficient information is lacking for a good estimate of $\sigma^2$. It is therefore initially set to a value known to be wrong for the purposes of experimentation: $\sigma_0^2 = \sigma^2/2$ at $t = 0$. The algorithm is not particularly sensitive to this choice [Carvalho and Puterman 2003].

After observing the revenue at $t = 1$, the ordinary least squares method is used to estimate $\sigma_t^2 = \frac{1}{t}[\hat{\epsilon}_{-2} + \hat{\epsilon}_{-1} + \sum_{k=1}^{2} \hat{\epsilon}_k^2$ where $\hat{\epsilon}_k = log(d_k) - \alpha_t - \beta_t p_k$, $k = -2, -1, 1, \ldots, t$. The term $G(t)$ in equation 4.5 defines the weight given to exploration, and several different functions are used in the original work[Carvalho and Puterman 2003]. In this paper the piecewise linear function is used

$$G(t) = \begin{cases} T_c - t & \text{if } t < T_c \\ 0 & \text{otherwise} \end{cases} . \tag{4.6}$$

Note that $T_c$ is not necessarily the end of the considered time horizon, in the experiments presented here we will use the value $30$, and $G(t) = 0$ produces the myopic pricing strategy.

## 4.4 Artificial Neural Networks

The benefit of using ANNs over a Kalman filter is that there is no need specify which type of demand model to use. This is beneficial when the simple models provided here do not exactly capture the *true* demand. ANNs have previously been used in dynamic pricing, however this was in an economy with a limited quantity and finite demand [Kong 2004]. The problem with ANNs in this setting is the lack of exploration (the same problem that Puterman overcame with the one step look ahead function with the Kalman Filter). A workaround is also possible with ANNs.

In order to initialize the ANN some training data needs to be generated. This data will be gathered by generating some number of random prices, observing the true quantity demanded from those prices, and using this data to train the network. These data points are part of the time horizon given to the algorithm to learn and set prices. A sliding window approach is used to handle the dynamic environments. At each time point the oldest data instance from the previous training is thrown away, the latest observed price and quantity

**Algorithm 4.1** Simulating to choose the best price; determining exploitation or exploration. $T$ is the end of the time period for which revenue should be maximized and $t$ is the current time step. In this case $NumberOfPrices$ is 3: the price that maximizes revenue according to the ANN and two random prices within our allowed price range for exploration.

```
Get quantity demanded from previous time period
Adjust data set for new data point and retrain ANN
//Find price in given range that maximizes expected revenue:
p[0] ← p_max
Choose two random prices in the given range
p[1], p[2] ← randomPrice()
for i = 0 to i < NumberOfPrices do
    Create a copy of the true ANN
    for j = t to T do
        if j = t then
            price ← p[i]
        else
            //Use price that maximizes revenue according to copy of ANN:
            price ← price_max
        end if
        Observe expected quantity demanded from ANN copy with chosen price
        Adjust training data for the copy ANN
        Retrain copy of ANN with new data set based on expected value
    end for
end for
Set price that returned the highest expected revenue from simulation
```

demanded pair is added to the data, and the neural network is re-initialized and then trained on this new data set.

An ANN can be set up to track the demand, but if the price chosen always maximizes revenue according to the ANN, no exploration will be done and prices will always be selected from a narrow range. The price that maximizes the expected revenue is found by inputting all allowed prices into the neural network, calculating the revenue according to what the knowledge the network has and choosing the price with the highest revenue. In order to track the changing parameters of demand, some algorithm other than a myopic pricing strategy with ANNs may be needed.

Algorithm 4.1 is based on a similar idea that was presented for the Kalman filter [Mullen et al. 2005] and attempts to take a value of information approach to the explore versus exploit idea. The myopic pricing strategy can be used to maximize revenue accord-

ing to the information that the neural network has accrued. Alternatively, another price can be chosen that may increase the information known about demand and increase revenue for the remaining time periods. First, the algorithm makes a copy of the ANN that is tracking demand. The price that maximizes the expected revenue according to the ANN is used in addition to two chosen uniformly random prices from the given price range. The first price is set and the copy of the ANN is retrained. From this point forward in the simulation, the price that maximizes the ANN's estimated revenue at each time step is chosen. The initial price which leads to the highest simulated revenue for the rest of the objective period $T$ is chosen for the current time period. This price is set and the process is repeated at the next time period.

One possible variation that will be adopted with this algorithm is to add Gaussian noise to the expected observation (on line 16) of the simulation's neural network, using error from the expected revenue relative to the true demand to calculate variance. Initial experiments show that the ANN has difficulty with the dynamics of the system, in order to keep the neural net current only the previous sixty time steps will be used to train the network at each time step.

Experiments will also be performed to test if different training data will help with the dynamic nature of the problem. Instead of having one data point for each time step in our training data, a linearly increasing number of data points will be used as the data gets closer to the current time period. With our sliding window size of sixty, there will be sixty values from the previous time step, fifty nine from the one before, and so on until there is only one from sixty time steps into the past. In other words the number of data points $n_t$ generated for $t$ time steps in the past is

$$n_t = \begin{cases} 61 - t, & \text{if } t \leq 60 \\ 0, & \text{otherwise} \end{cases} \tag{4.7}$$

Small amounts of noise will be added to the observation portion of each of the extra data points for each time period. The neural network requires a large number of data points to train correctly. However, as the demand parameters move with time, the older data points will not be representative of the current state of demand. The hope is that this change will allow the neural network more data to train on, with greater weight given to the more recent data points.

## 4.5 Experimental Setup

The neural nets for these experiments were performed using the Fast Artificial Neural Network Library (FANN), version 2.0 [Nissen 2006], with most of the default settings except as stated below. Three layers were used with one input node, one output node and two hidden nodes. The learning rate was set to $.91$, the maximum number of epochs to $10000$ and the desired error to $10^{-10}$. The sigmoid activation function was used for each layer and the incremental training was also used. Since the output layer uses a sigmoid function and FANN is optimized for values between zero and one, all inputs and outputs for training were scaled to this range. The prices were normalized using a low and high price for each model, $0.33$ and $3.0$ for the log-linear model and the additional model presented here, and $40.0$ and $60.0$ for the linear model. The training of the neural network is started with fifty random data points in the pricing space.

The true demand in the linear model was initialized with $\alpha = 150.0$, $\beta = -1.5$, $\mu = 0$ and $\sigma^2 = 20.0$. To this model, one of the following four dynamic contexts is applied to the parameters of 4.1, described below. In all of the following scenarios $\alpha$ and $\beta$ vary with time and have a random component represented as a Gaussian random walk, parameterized by $(\mu_\alpha, \sigma_\alpha)$ and $(\mu_\beta, \sigma_\beta)$ respectively. Unless otherwise specified, this is

represented by the following equations:

$$\alpha_t = \alpha_{t-1} + \delta_\alpha \tag{4.8}$$

$$\beta_t = \beta_{t-1} + \delta_\beta. \tag{4.9}$$

Where $\delta_\alpha \sim \mathcal{N}(\mu_\alpha, \sigma_\alpha)$ and $\delta_\beta \sim \mathcal{N}(\mu_\beta, \sigma_\beta)$. In all cases with linear demand $\sigma_\alpha = 1.0$ and $\sigma_\beta = 0.015$. The scenarios follow.

**Parameters remain near original values:**

$$\mu_\alpha = \mu_\beta = 0$$

**Large change at $t = 300$:**

$$\mu_\alpha = \mu_\beta = 0$$
$$\alpha_{300} = \alpha_{299} + 10.0 + \delta_\alpha$$
$$\beta_{300} = \beta_{299} + 0.2 + \delta_\beta$$

**Overall upward trend:**

$$\mu_\alpha = 0.05 \qquad \mu_\beta = 0.0003$$

**Overall downward trend:**

$$\mu_\alpha = -0.05 \qquad \mu_\beta = -0.0003$$

The log linear demand model is setup in the same way with $\alpha = 8.0$, $\beta = -1.5$, $\mu = 0$ and $\sigma^2 = 4.0$. Equations 4.8 and 4.9 are used with this demand as with the linear

demand. In this case $\sigma_\alpha = .0005$ and $\sigma_\beta = .001$. The same four scenarios are also used as described below.

**Parameters remain near original values:**

$$\mu_\alpha = \mu_\beta = 0$$

**Large change at $t = 300$:**

$$\mu_\alpha = \mu_\beta = 0$$
$$\alpha_{300} = \alpha_{299} + 0.4 + \delta_\alpha$$
$$\beta_{300} = \beta_{299} + 0.2 + \delta_\beta$$

**Overall upward trend:**

$$\mu_\alpha = 5 * 10^{-6} \qquad \mu_\beta = 5 * 10^{-4}$$

**Overall downward trend:**

$$\mu_\alpha = -5 * 10^{-5} \qquad \mu_\beta = -1 * 10^{-3}$$

The demand given in equation 4.4 will only be used with constant parameters and parameters that stay close to the original values. The noise will be Gaussian with $\mu = 0$ and $\sigma = 0.2$. The initial parameters are $\alpha = 1.0$ and $\beta = -2.0$. The population size $\chi$ will be $10000$. In similar fashion to the previous two models for the changing parameters, a Gaussian random walk is used with the parameters $\mu_\alpha = \mu_\beta = 0$ and $\sigma_\alpha = \sigma_\beta = 0.05$.

Each experiment was run to maximize revenue over $1000$ time steps and the results averaged over $500$ runs.

## 4.6 Results

In any discussion about results, when a claim is made that one algorithm outperforms another this refers to the total revenue earned by the algorithms for the 1000 time step period. In order to keep the graph scales at a readable level, the mean cumulative revenue is presented which, graphically, shows the same thing as total revenue. Mean cumulative revenue is defined as $\sum_{n=1}^{t} r_n/t$ where $t$ is the current time step and $r_n$ is the revenue earned at $t = n$. Some discussion will be given to cases where one algorithm begins progressing slowly, only to end up surpassing another, with an attempt to reason why this may occur. The abbreviations given in the result graphs are given in Table 4.1. Some of the variations presented in Table 4.1 did not perform well at all and so were removed from most graphs to avoid cluttering the results, however these algorithms are presented in a couple of graphs to aid the final discussion. Each graph has an optimal performance line on it that is the revenue that could have been earned if everything about demand had been known at each time step.

The first set of graphs are all from the linear demand model given in Equation 4.1. The first set of results, with constant parameters, given in Figure 4.3, shows that the neural network with exploration performs the best. This implies that the neural network has a very accurate model of the demand. The exploration falls apart, however, when dynamics are added to the system. Figure 4.4 has results with the parameters changing with time and shows that the NN-Sim algorithm begins quite well, but as the system moves further from the original points, the regular neural network surpasses the simulation algorithms. When the simulator is able to predict accurately, the chosen prices end up helping total revenue. However, when it cannot predict accurately, it cannot make good decisions. In the case with the large jump in parameters (Figure 4.5), the regular neural network again earns more revenue than any other algorithm. However one note of interest is that the variance added to the simulated observations allowed the simulation algorithm to handle the large jump in demand better than the alternative with no variance. Figures 4.6 and 4.7 show

results with the trends up and down. The regular neural network also performs the best for these variations, again because the dynamics in the model cause the explorations to go further astray.

The next set of graphs is for the log-linear results, although only a small sub-set of the results are shown because each one showed the same final result. The results from the constant parameters and those slightly changing with time (Figures 4.8 and 4.9, respectively) show that the Kalman filter algorithm with one-step look-ahead earns the most revenue in both cases. In the log-linear case the noise is greater than in the linear case because it is in the exponent. Even small amounts of noise from the draw can have large effects on the end quantity demanded. The increased unpredictability can be seen in Figure 4.9 where adding the Gaussian noise to the simulation observations causes the neural network performance to plunge. This occurs even if the neural network has the demand approximated accurately because the difference between the predicted values and the actual values are very different.

Figures 4.10 shows the results from the last demand model used. Here the two neural network algorithms start the same, and once the gathering of the initial training data is done, the regular neural network improves over every algorithm. The NN-Sim algorithm is spending too much time exploring and so stays at the same level as when the prices were set randomly. Using the one-step look-ahead function causes the Kalman Filter to fail terribly. Once exploration was finished the KF-OSL performed at the same level as the KF algorithm; however, it could not make up for the revenue gained by KF during the initial time frames. Figure 4.11 gives the results for demand changing with time, in which the neural network performs the best.

The NN-Sim algorithm presented here does not do as well as expected. In looking more closely at the results we discovered that the simulation algorithm only exploits thirty percent of the time, and this is consistent for all models and all cases. With seventy per-

Table 4.1: Key for graph abbreviations

| Label | Description |
|---|---|
| KF | Kalman filter, in the case of linear and log-linear this tracks the true demand. For the alternate demand this assumes log-linear. |
| KF-OSL | Kalman filter with one step look ahead. This algorithm always assumes log-linear demand. |
| NN | Neural network that always exploits. |
| NN-Sim | Neural Network with exploration as explained in algorithm 4.1. |
| NN-Sim-Var | Neural network with exploration as explained in algorithm 4.1 with variance added into observations. |
| GaussTrain | Neural network with Gaussian noise and extra training examples added at more recent time periods. |

cent exploration it is no wonder that the simulation algorithm does not perform as well as expected.

## 4.7 Conclusions and Future Work

Neural networks in dynamic pricing have one advantage over Kalman filters in that no prior knowledge is necessary to set up the pricing algorithm. If there is little or no prior information about demand, then neural networks provide good tracking when there is an ample amount of diverse data. The weakness of using ANNs for this type of problem is the need for more initial data than the other algorithms presented here. This occurs because the neural network is attempting to approximate the entire function. When ANNs use an exploitation strategy, the information decays because the points are all in a narrow price range. In addition, the random prices set at the beginning of using an ANN would not translate well to a real-life problem where there is an explicit cost in gaining an observation.

There are other variations that may or may not improve performance for the neural network such as soft max (exploit with some random exploration). Obviously the simulation algorithm presented here does not work as expected and part of that reason is too

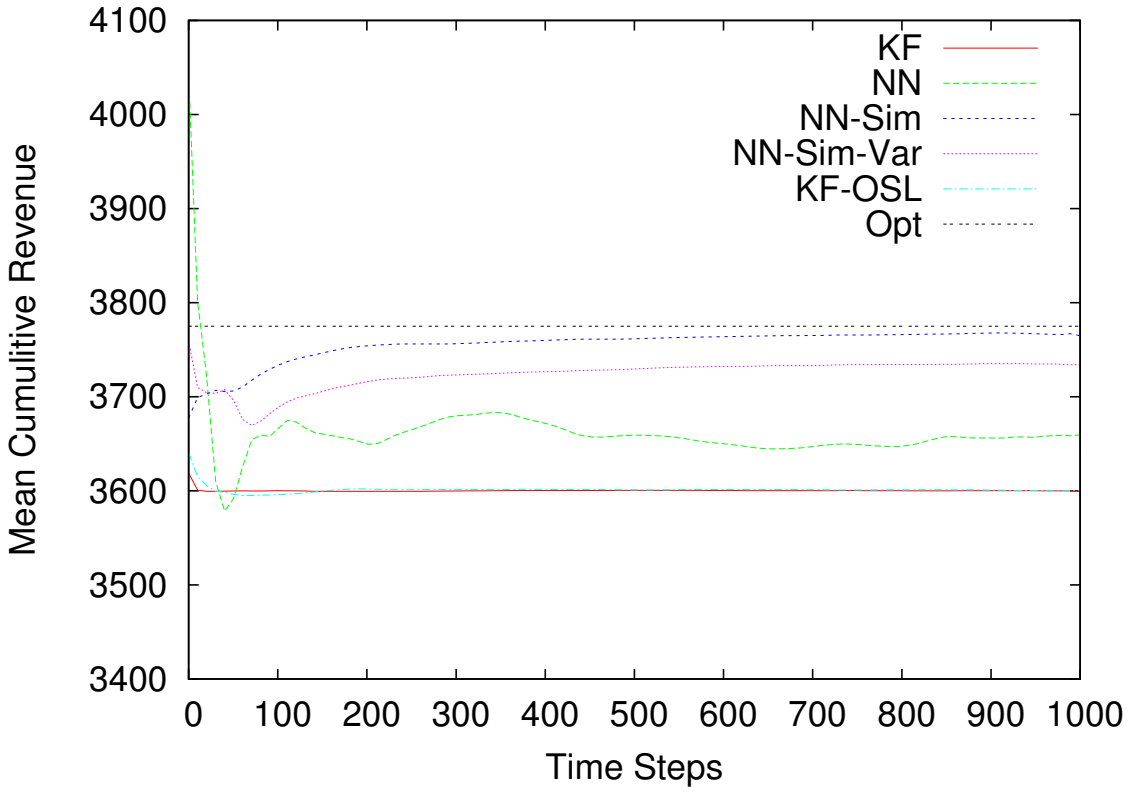


Figure 4.3: Linear Demand with Constant Parameters

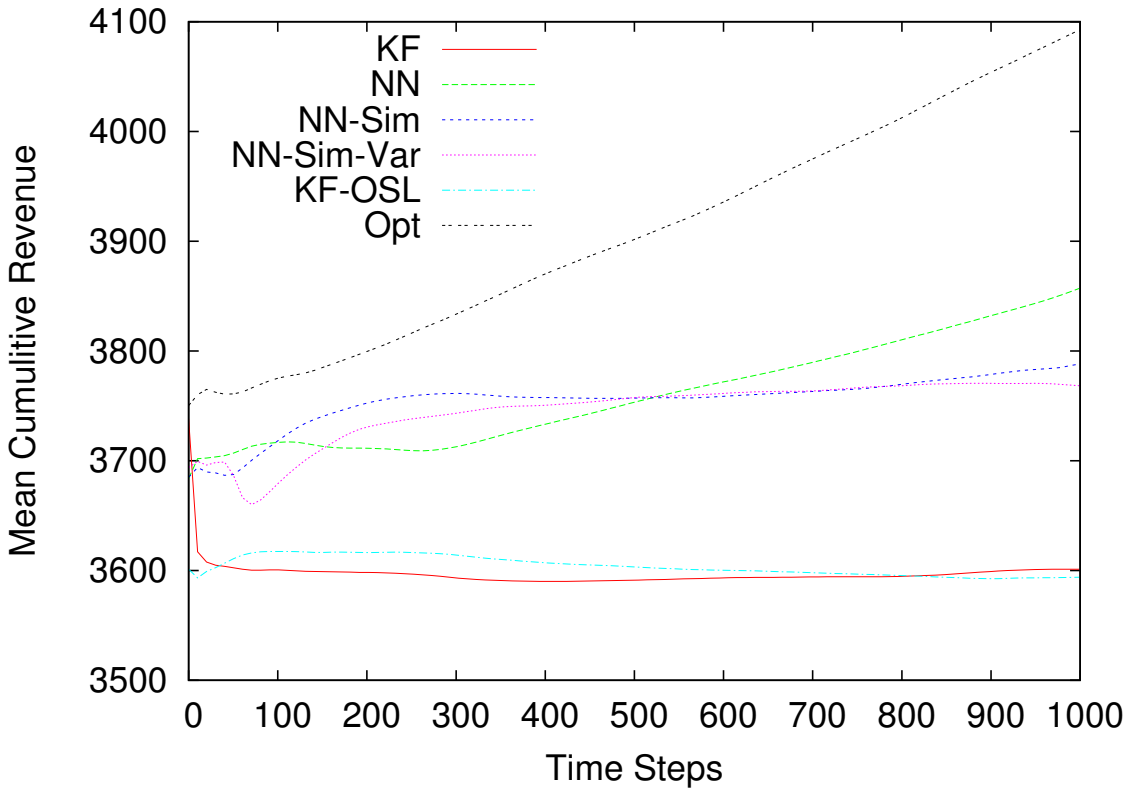


Figure 4.4: Linear demand with random walk–The trend upward happens because with an equal shift in the demand parameters up and down, the upward shift causes a larger jump in revenue than the equal shift down.

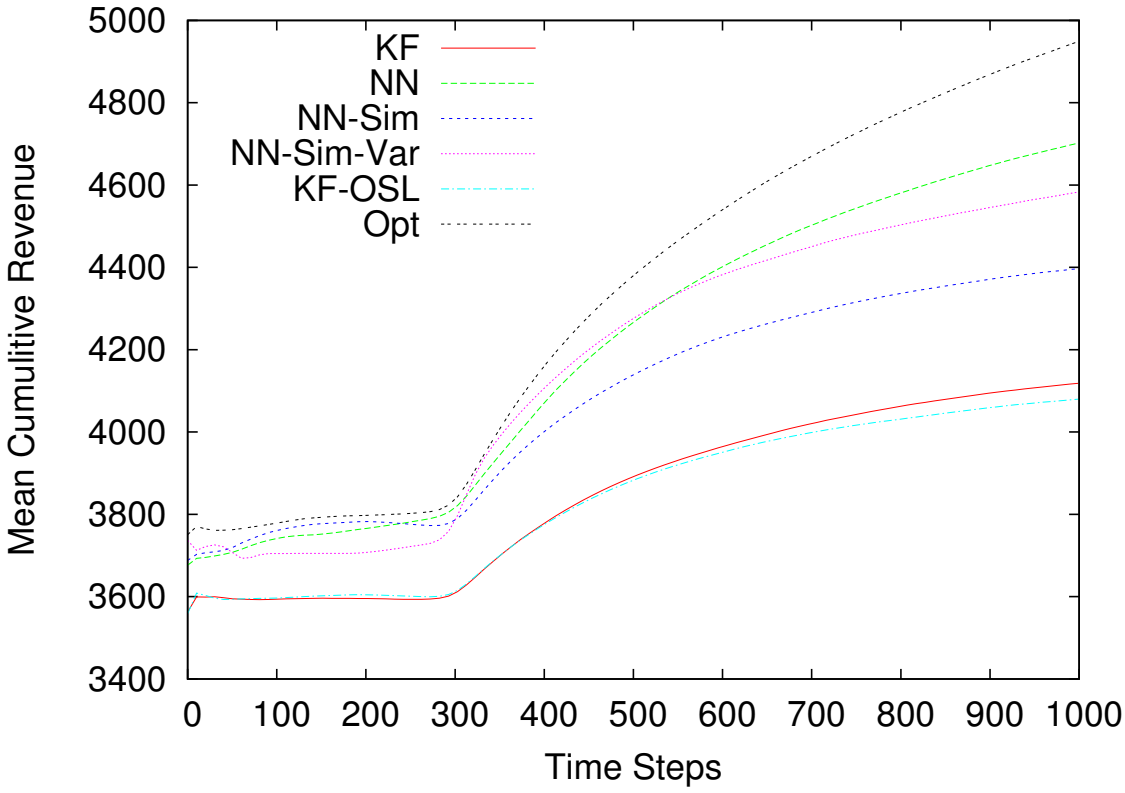


Figure 4.5: Linear Demand with Intervention

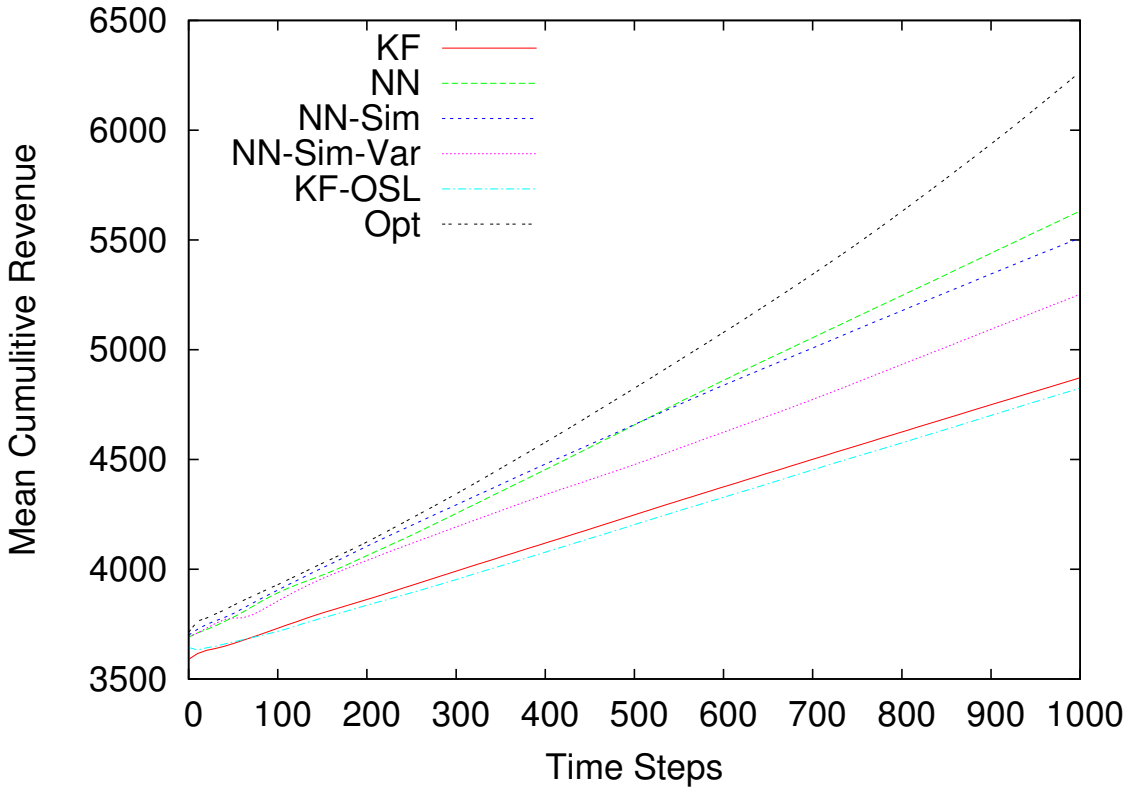


Figure 4.6: Linear Demand with Trend Up

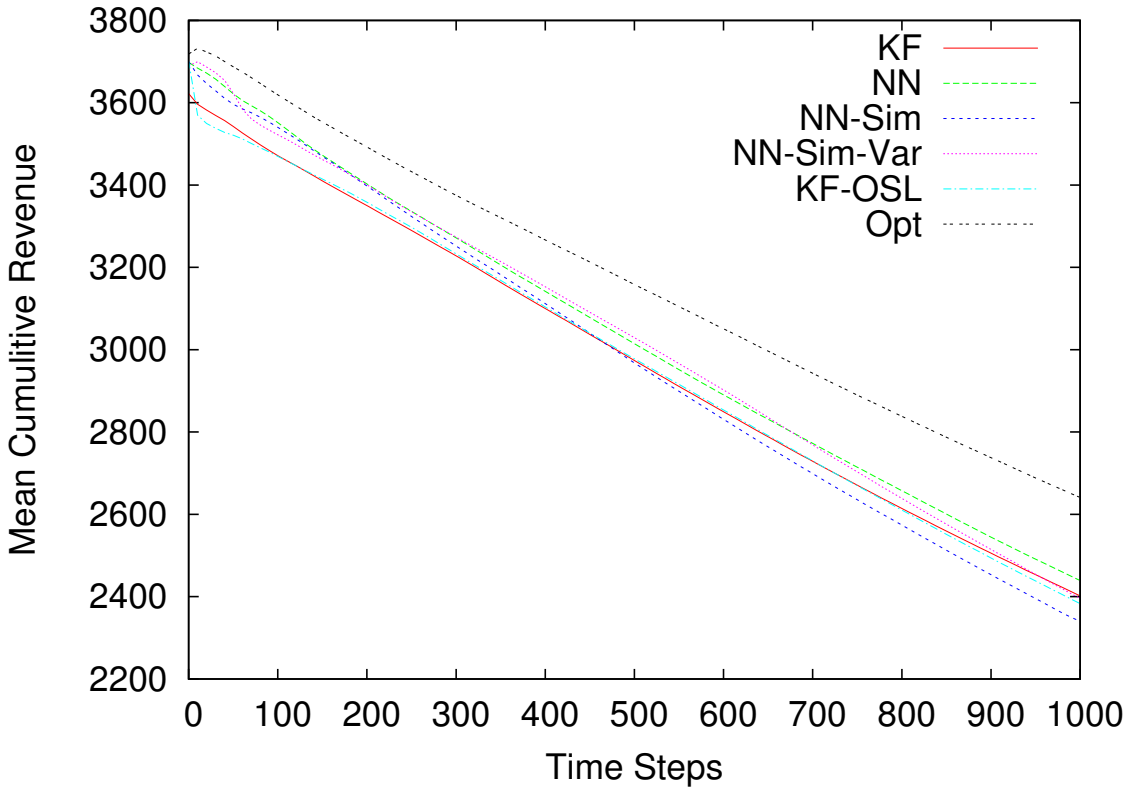


Figure 4.7: Linear Demand with Trend Down

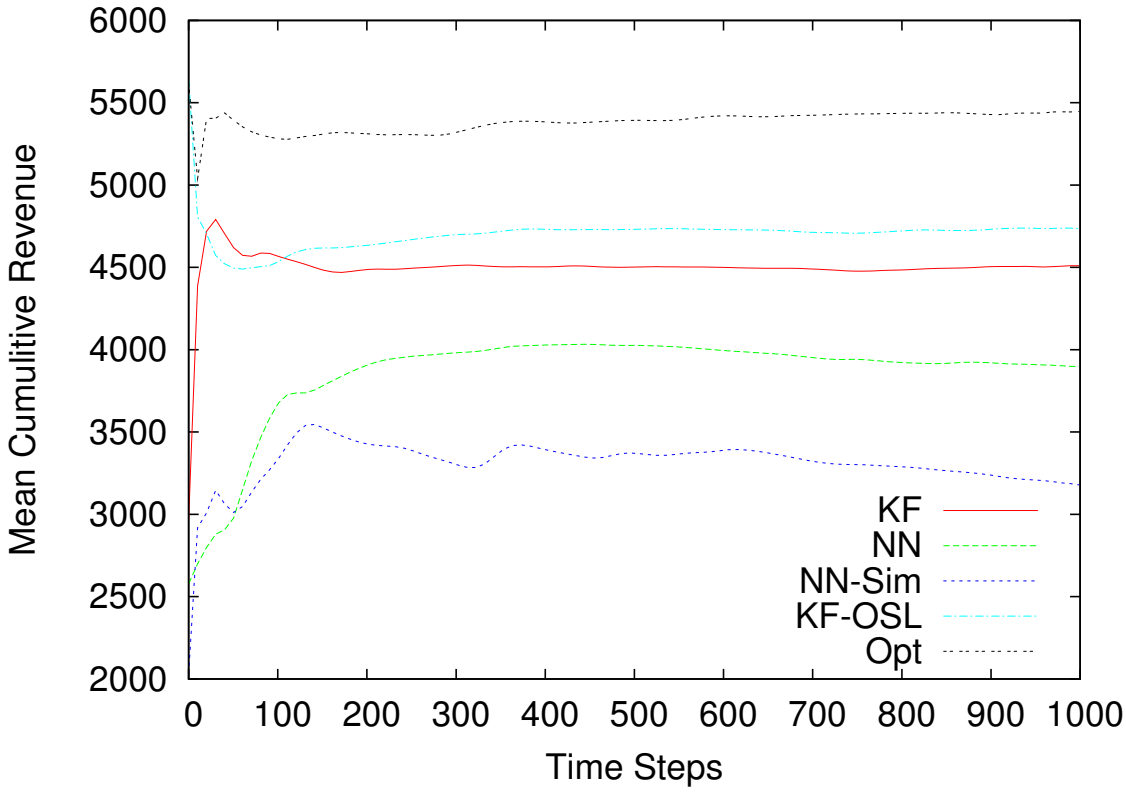


Figure 4.8: Log Linear Demand with Constant Parameters

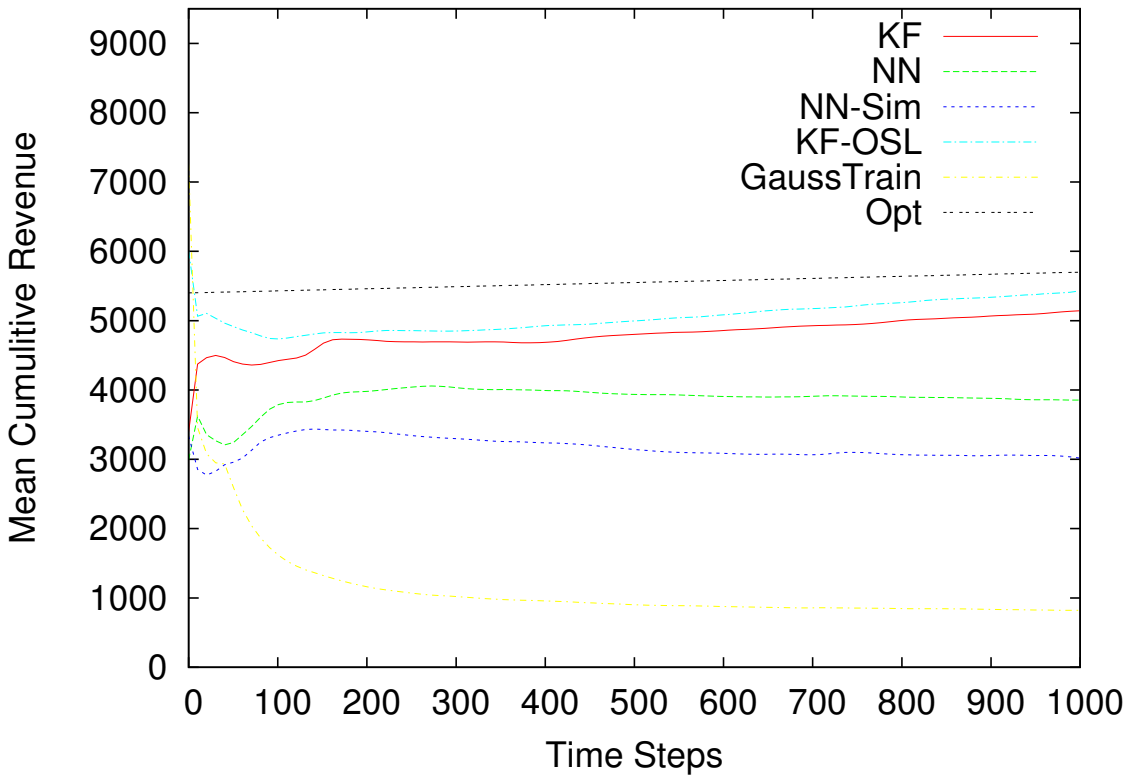


Figure 4.9: Log Linear Demand with Random Walk–The upward trend occurs because of the exponential demand. An increase in parameters causes a larger shift in the revenue than an equal downward change in the same parameters.

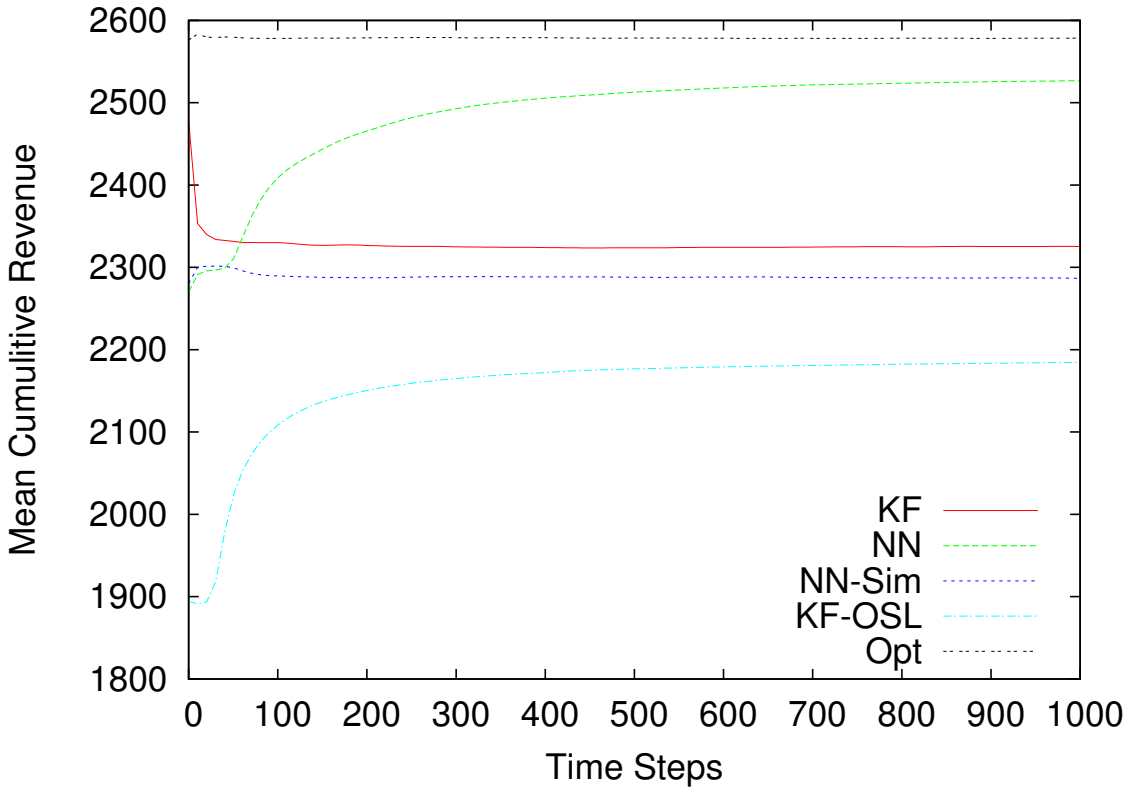


Figure 4.10: Alternate Demand with Constant Paramters

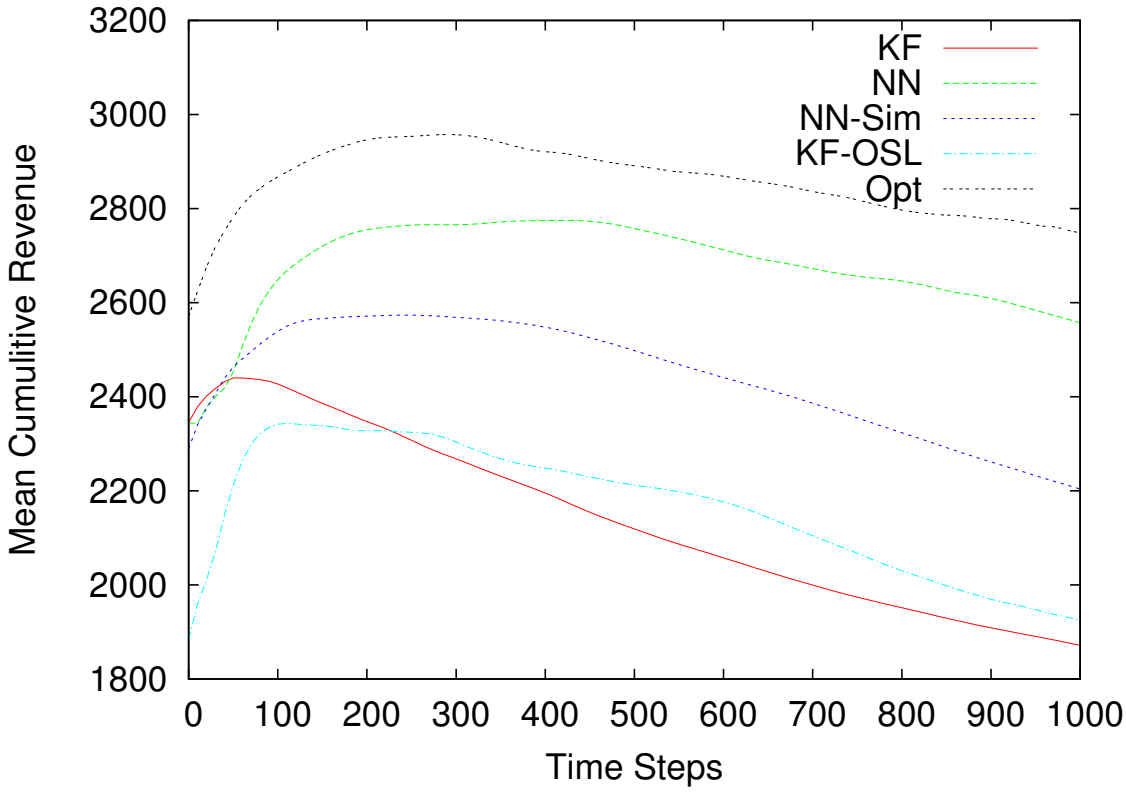


Figure 4.11: Alternate Demand with Random Walk

much exploration. Soft max would provide a compromise between the heavy exploration of NN-Sim and the pure exploiting of NN. While these changes to the ANN may provide some improvement, the drawbacks to the ANN lead to the conclusion that the neural network would be better used in conjunction with some other machine learning algorithm. The ideas behind using an ANN for the dynamic pricing problem still have merit. Instead of trying to modify the way the neural networks behave, it may be beneficial to have an ANN learning in the background without setting prices until some error threshold has been reached while another algorithm determines the exploration pattern of the prices and then switches to the ANN for exploitation.

# Chapter 5

## Results

To facilitate the final discussion of results and conclusions, this chapter will present complete graphs not given in any of the chapters thus far. In order to compile the results the experiments needed to be run with the same parameters. For the linear results the experimental setup from Section 4.5 is used. In order to get the results for the log linear demand, the experimental setup for the market from Section 3.6 is used. In addition, results from a basic genetic algorithm are presented, a brief description of the setup for this algorithm follows. Results from a multiple product environment are also given and an explanation of that market model is also presented.

In order to make graphs in this section easy to interpret, few algorithms are actually presented. A Kalman filter tracking parameters is shown as the myopic pricing scheme for linear demand; it is referred to as KF in the results. In the case of log-linear demand the Kalman filter with a one-step look-ahead function (KF-OSL) is used [Carvalho and Puterman 2003]. This algorithm performs well for the log-linear demand and so is presented for comparison purposes. The Kalman filter simulation algorithm (KF-Sim) presented in Section 2.2 is the five price algorithm (following the once exploit, always exploit philosophy), the Fixed$_{10}$-PBDPSO$_{99}$ from Section 3.7 (PSO) represents particle swarms, and the neural network with pure exploitation from Section 4.5 (NN) is presented for the ANNs. Results from genetic algorithms (presented in Section 5.1) are included as well (GA). These algorithms, while not the best in every case from their respective chapters, present a good idea of what the main algorithms (PSO, GA, NN) can do, and will

allow a good discussion on strengths and weaknesses. The optimal revenue is also given on each graph (Opt), this is the revenue that could be earned if everything were known about demand at each time step.

## 5.1 Genetic Algorithm Setup

In addition to the algorithms presented so far, genetic algorithms with no alterations are included for comparison purposes. A genetic algorithm package for python [Dequnes 2003] was used to generate the results presented. For a complete description on how the following parameters are used please see the package documentation. The initial organisms were created with mutation set to $0.95$ (at first they had a ninety five percent chance of mutating), with a mutation amplification set to $0.05$ (controls how far from values the numbers can mutate). These parameters are part of the organisms gene set and so change with time like the other parameters. The initial high mutation rate was set to cause quick exploration at the beginning, and as the population settled closer to a better solution the mutation rate would drop. Each organism also had an $\alpha$ and a $\beta$ that they were trying to learn. For the linear demand model this was initially a uniform random number between $-2.0$ and $-1.0$ for $\beta$ and between $140$ and $160$ for $\alpha$. For the log linear model, $\beta$ was drawn from the same range and $\alpha$ between $7.0$ and $9.0$.

To start the life cycle, elitism was set to one (only the best survive from each generation), the initial population size is $2$, the number of generations is set to $200$, the number of children to $4$ and the number of organisms to survive in each generation is also set to $4$.

## 5.2 Multi-Product Economic Setup

The demand for the multi-product economy is based on the log-linear demand presented earlier (see Equation 1.2) to which a relationship parameter, $\gamma$, is added for each product

Table 5.1: Parameters for multiple product economy

| Variable | Value |
|---|---|
| $\alpha_1$ | $9.0$ |
| $\beta_1$ | $-1.0$ |
| $\mu_1$ | $0.0$ |
| $\sigma_1^2$ | $1.0$ |
| $\gamma_{12}$ | $-0.3$ |
| $\alpha_2$ | $8.5$ |
| $\beta_2$ | $-2.0$ |
| $\mu_2$ | $0.0$ |
| $\sigma_2^2$ | $0.55$ |
| $\gamma_{21}$ | $-0.3$ |

that is to be included in the model. This gives us

$$d_i = \exp\left(\alpha_i + \beta_i p_i + \epsilon_i + \sum_{j \in 1..n, j \neq i} \gamma_{ij} p_j\right). \tag{5.1}$$

Where each product has its own $\alpha$, $\beta$ and parameters $\sigma$ and $\mu$ for the normal noise $\epsilon$. The $\gamma$ term defines the relationship between the products $i$ and $j$. If $\gamma$ is positive the products are competing, raising the price on product $j$ will increase the quantity demanded for product $i$. On the other hand, if $\gamma$ is negative the products are cooperating and lowering the price on product $j$ will increase the quantity demanded for product $i$. Setting $\gamma_{ij}$ to zero implies that the price of product $j$ does not affect product $i$.

In this case a two-product model is used where the two products are cooperating (i.e. hamburgers and french fries–lowering the price on hamburgers will sell more french fries). The actual parameters are given in Table 5.1. This model was run with constant parameters and with the parameters changing with time. In the case where the parameters changed with time, a normal random number was added to each $\alpha$, $\beta$ and $\gamma$. The parameters for the random walk are given in Table 5.2.

The algorithms used to set prices in the two-product environment were an ANN and PSO. The neural network was set up using two input nodes, one hidden layer with four nodes and one output node. The rest of the parameters were set as described in Section 4.5.

Table 5.2: Parameters for random walk of variables in multiple product economy

| Variable | Value |
|---|---|
| $\mu$ | 0.0 |
| $\sigma_\alpha$ | .1 |
| $\sigma_\beta$ | .01 |
| $\sigma_\gamma$ | .005 |

The ANN used a myopic algorithm to set the prices. The parameters for the particle swarm were as described in Section 3.6, the only difference is that this swarm is exploring a two dimensional space. The swarm used the same detection and response algorithm ($\text{Fixed}_{10}$-$\text{PBDPSO}_{99}$) as the rest of the swarms described in this chapter.

## 5.3 Result Graphs

This section contains the compiled results graphs. Figures 5.1 through 5.5 show the results from the linear demand model. Figures 5.6 through 5.10 give the results from the log-linear demand model. Figures 5.11 and 5.12 contain the the results from the multiple product economy. Results are once again give with mean cumulative revenue, defined as $\sum_{n=1}^{t} r_n/t$ where $t$ is the current time step and $r_n$ is the revenue earned at $t = n$.

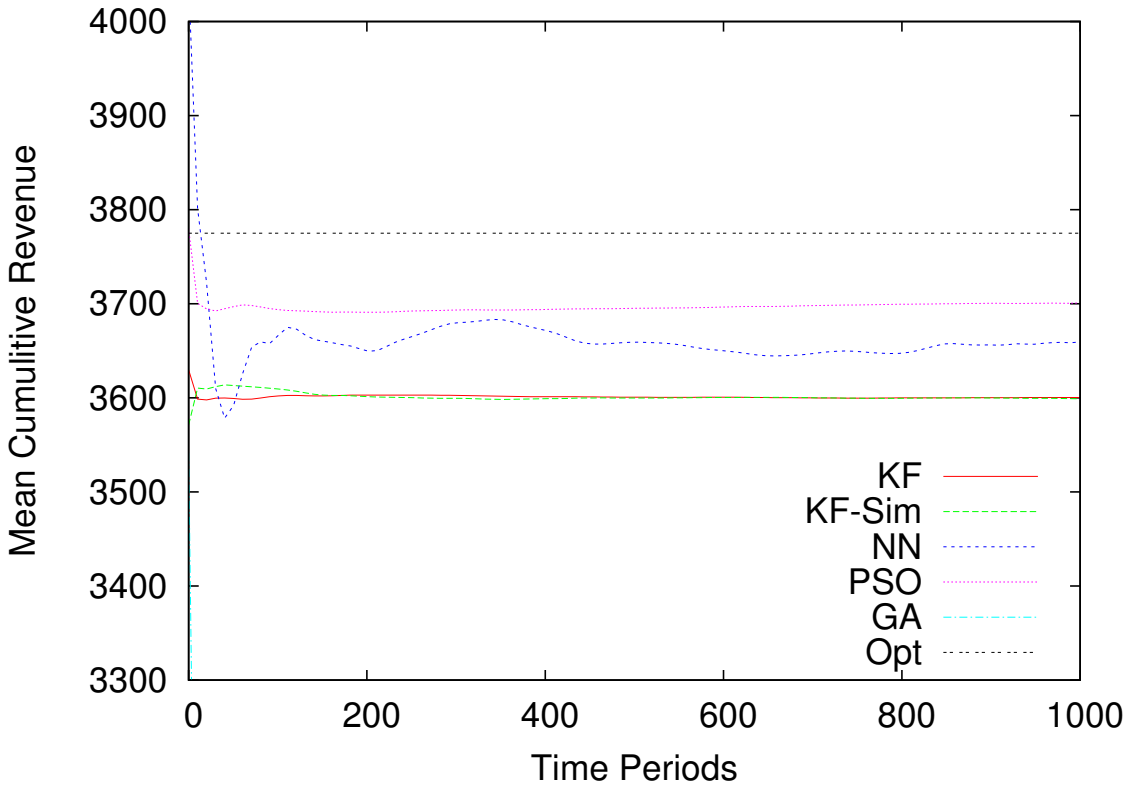


Figure 5.1: Linear demand model with constant parameters. The PSO algorithm performed the best here with the ANN following close behind. The two Kalman filter algorithms formed a baseline, for the most part using myopic pricing, which we really do not want to go below. The GA was too low to show up on the graph.

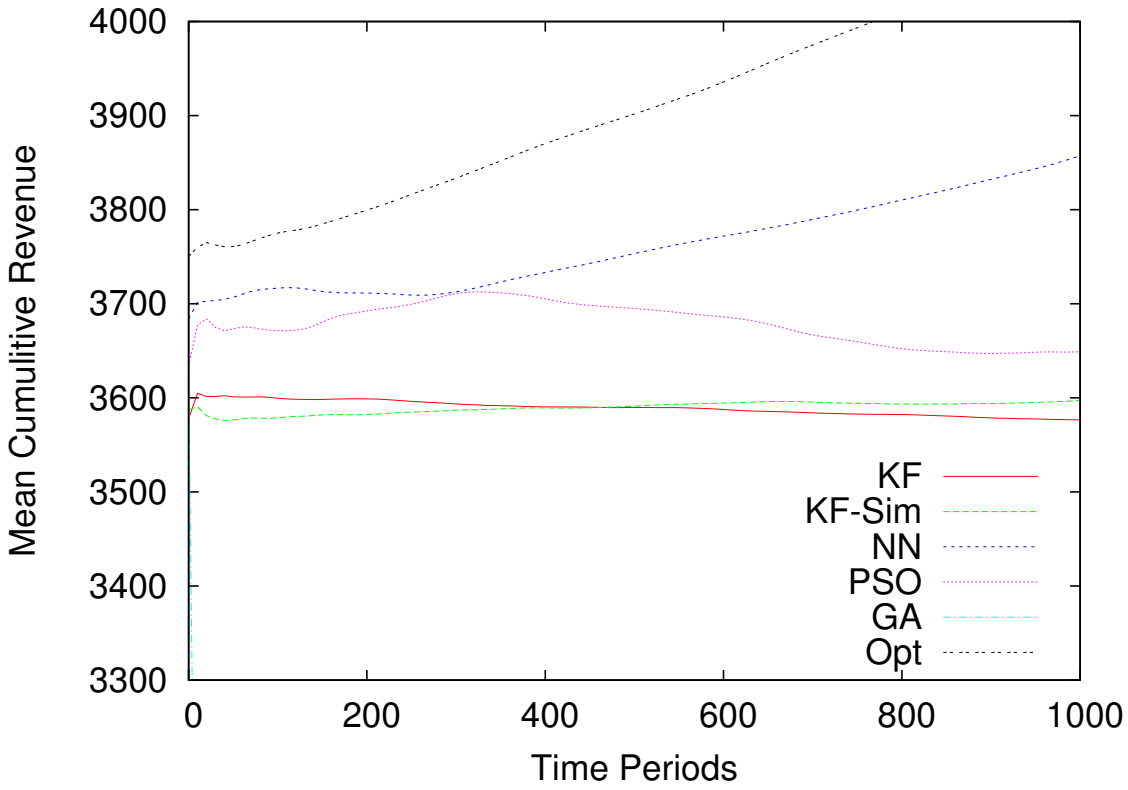


Figure 5.2: Linear demand model with random walk. The ANN performed significantly better then the other algorithms. The PSO seems to be wandering around a bit. The two Kalman filter algorithms did not do so well, again setting the lower limit of where we would hope our algorithms would be. The GA was below the scale on the graph.

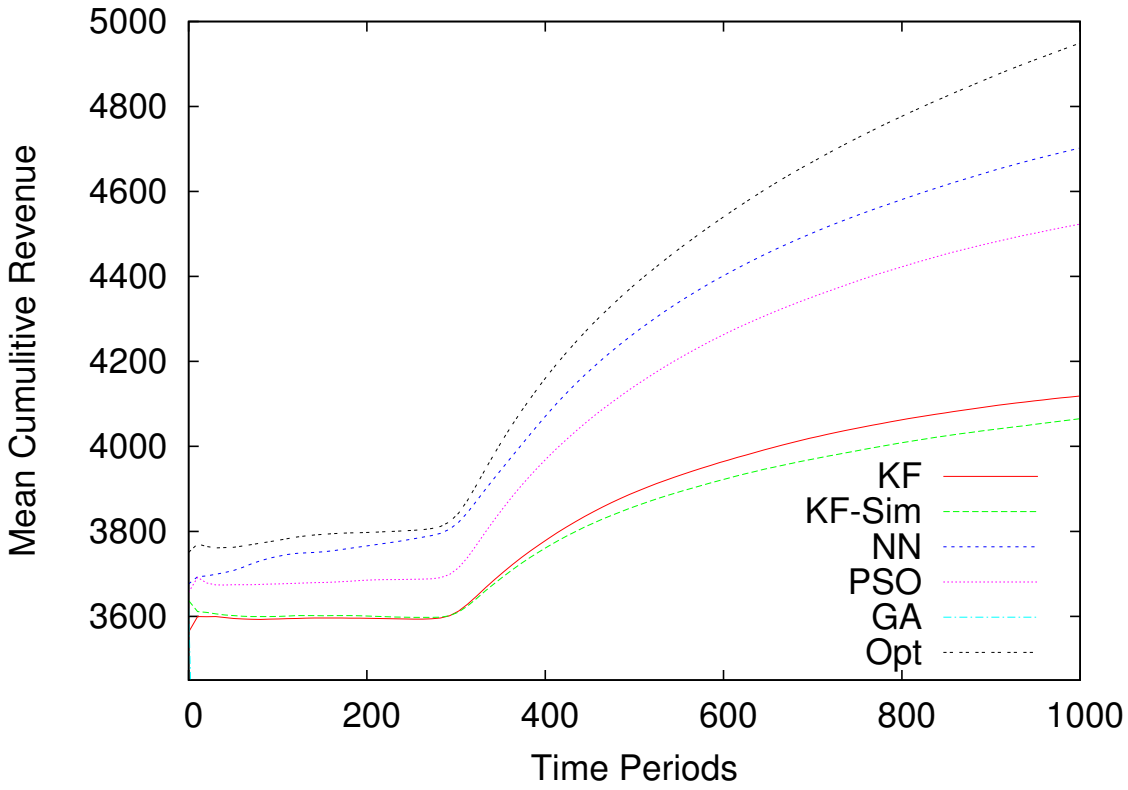


Figure 5.3: Linear demand model with random walk and large shift at $t = 300$. Once again the ANN performed the best but is followed closely this time by the PSO. The Kalman filter algorithms are still performing at the lower bound but outperforming the GA, which is again not shown on the graph.

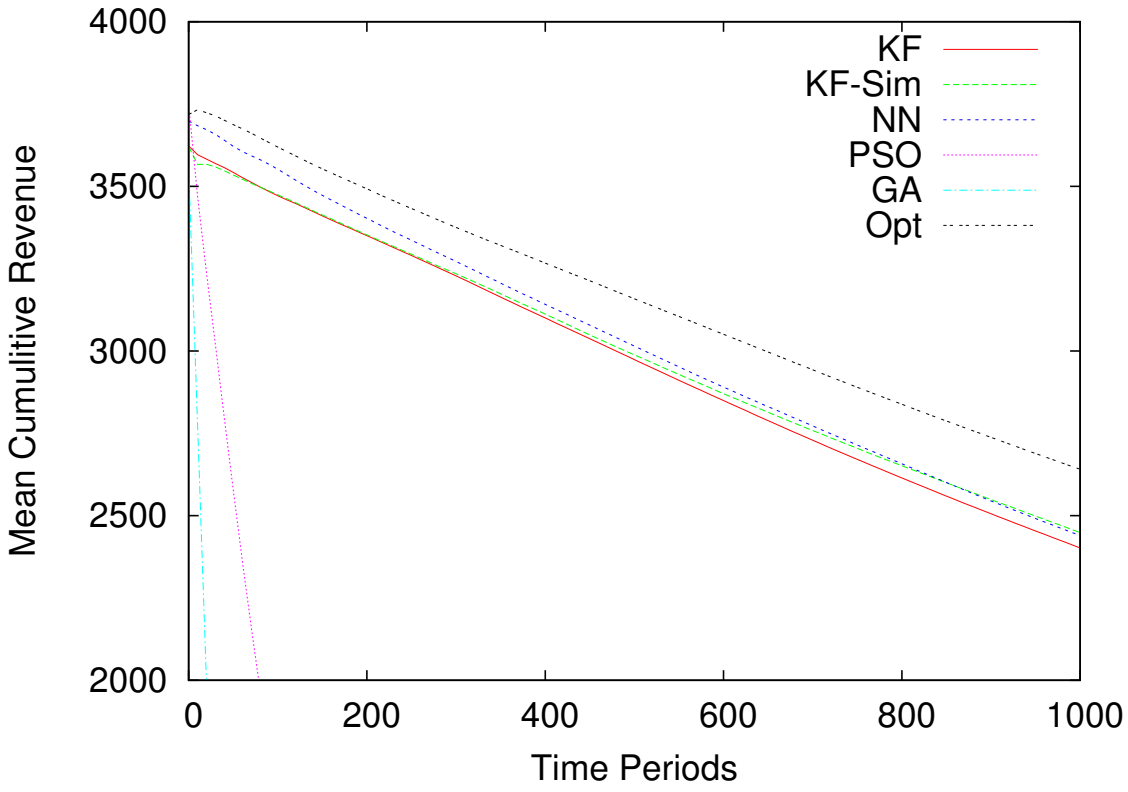


Figure 5.4: Linear demand model with random walk and overall trend down. The ANN was the best again with the Kalman filter algorithms not finishing too far behind. PSO does not perform well on downward trends as discussed in Section 3.7.

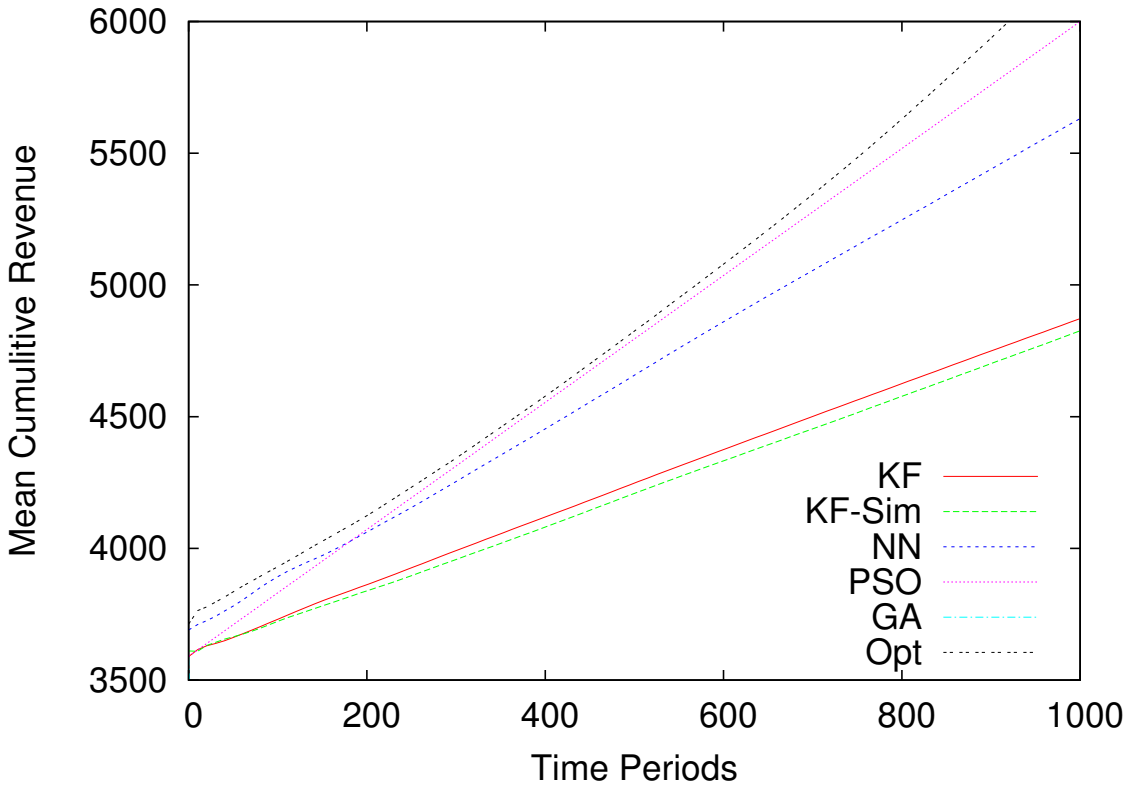


Figure 5.5: Linear demand model with random walk and overall trend up. PSO performed the best for this model, seeming to prefer the upward trends (see Section 3.7). The ANN was not too far behind. The Kalman filter algorithms set the lower bound for acceptable algorithms and the GA was once again too low to show.

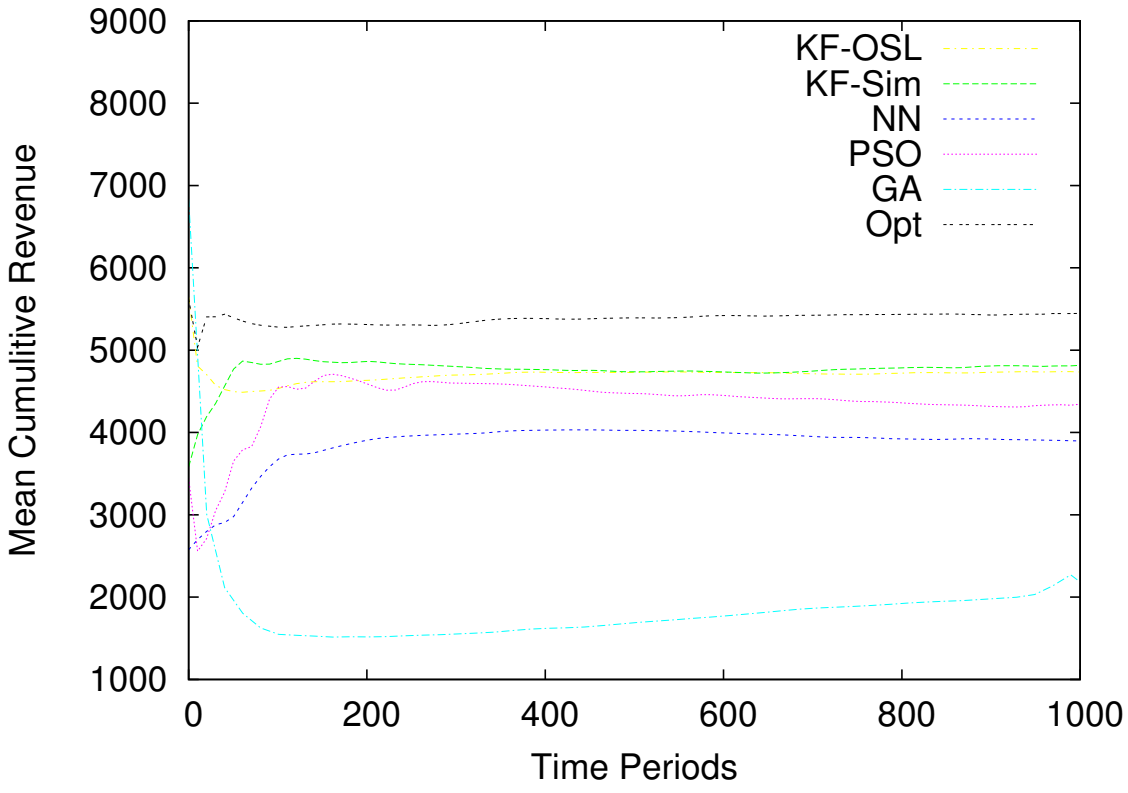


Figure 5.6: Log linear demand model with constant parameters. The two Kalman filter algorithms performed in similar fashion. The PSO finished shortly behind, followed by the NN. The GA was lower than what we would call a successful run.

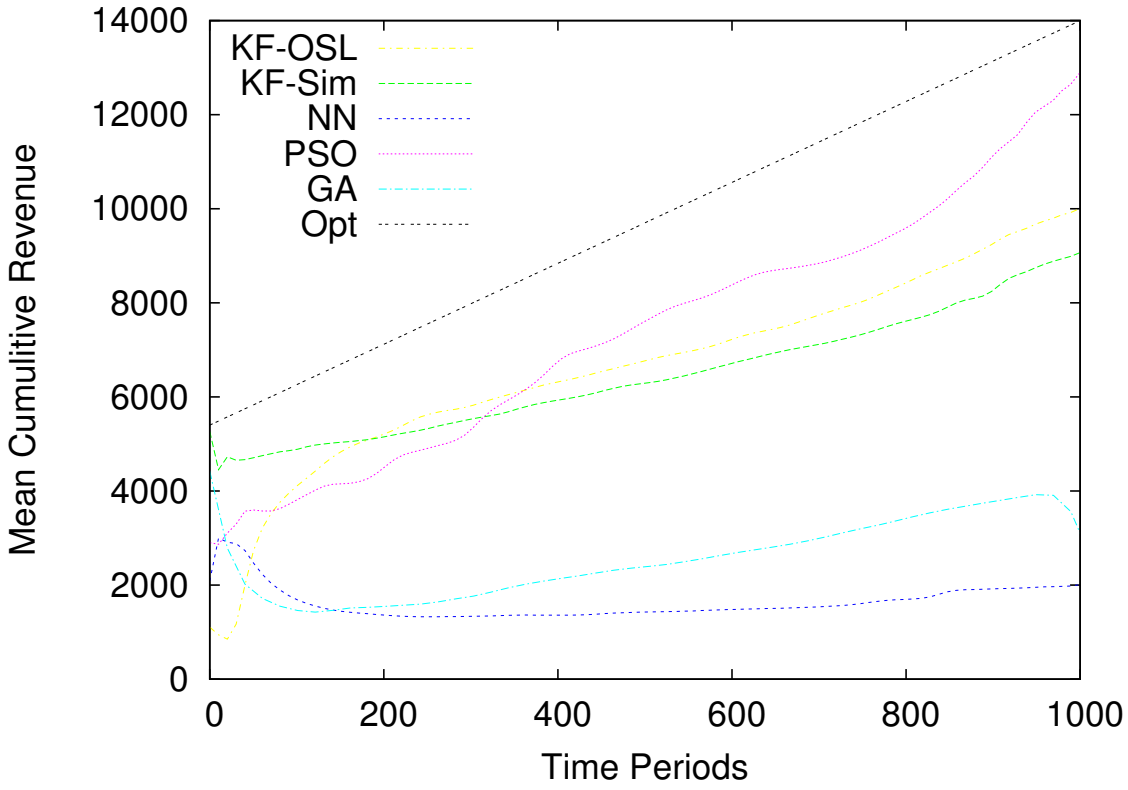


Figure 5.7: Log linear demand model with random walk. The differences between the KF and KF-OSL are interesting to look at in this environment. The KF-Sim started off better with the simulation handling the dynamic environment better. The KF-OSL, which uses random exploration, was able to catch and surpass the KF-Sim algorithm, which uses once exploit always exploit, as the parameters changed. The PSO gained the most revenue while the NN could not handle the dynamics here (the parameters changed more for this market model than in Chapter 4).

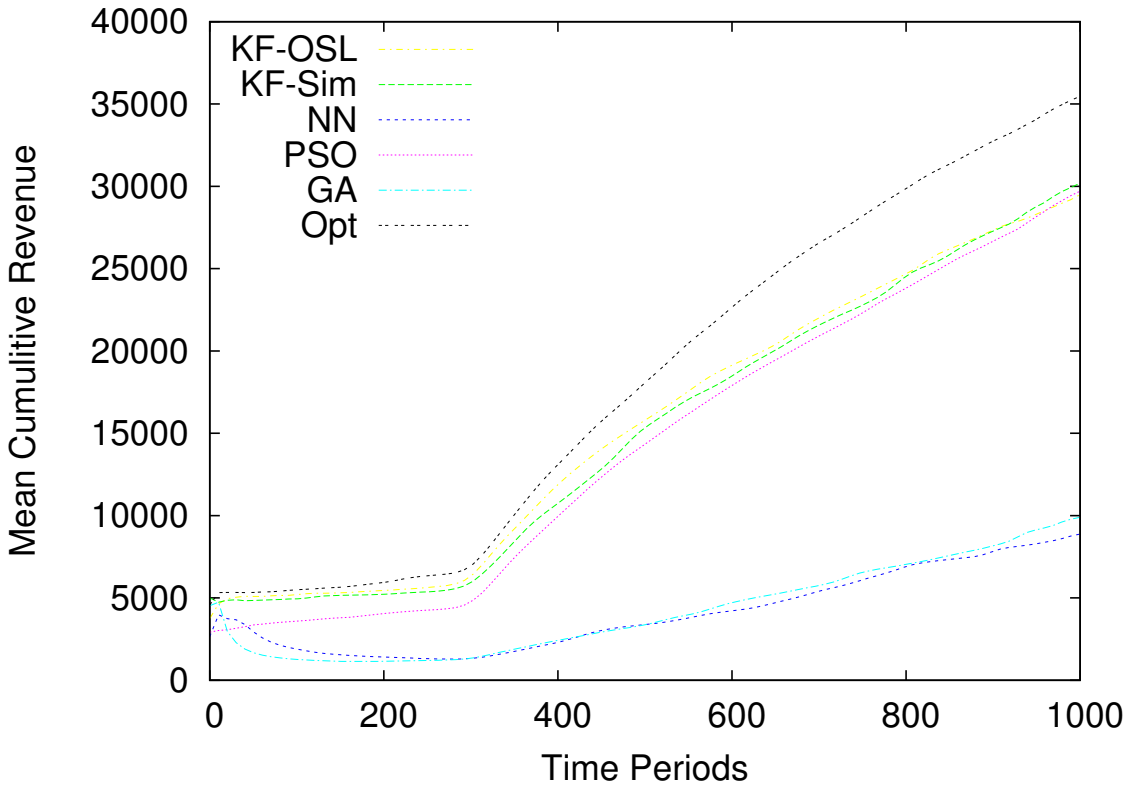


Figure 5.8: Log linear demand model with random walk and large shift at $t = 300$. The same behavior from the two KF algorithms are noted as in Figure 5.7. PSO was once again competitive with the two Kalman filter algorithms.

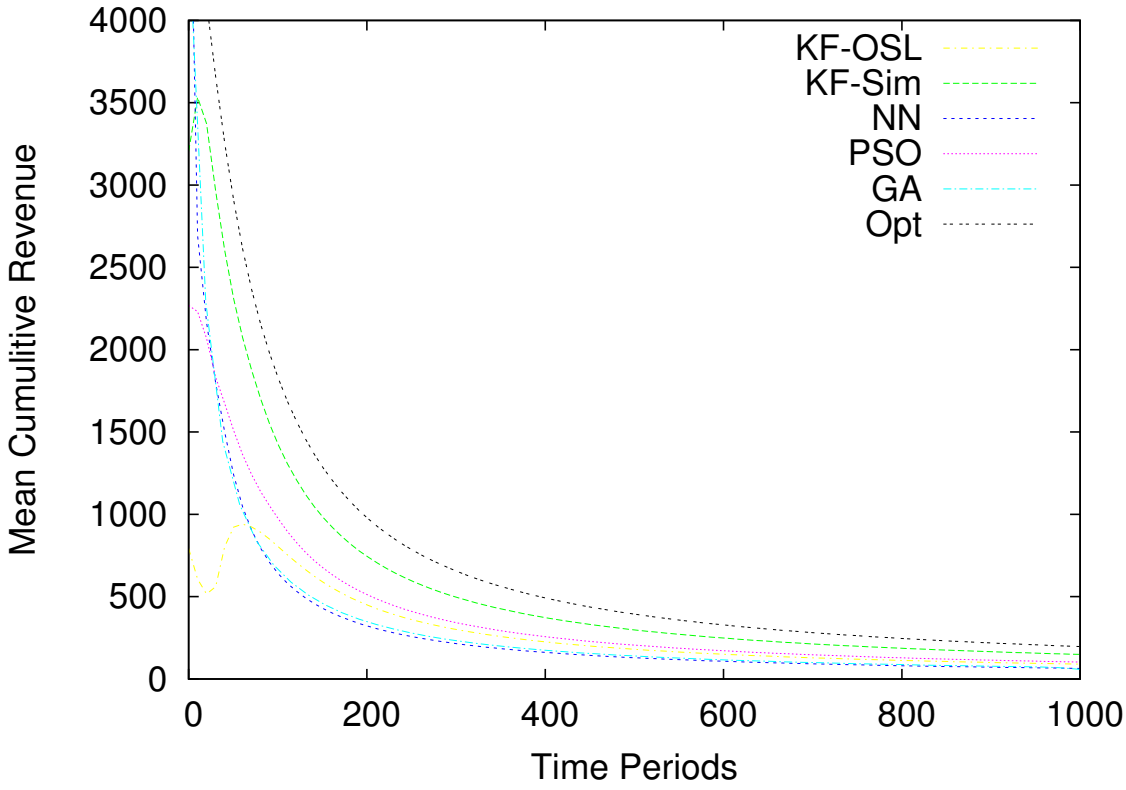


Figure 5.9: Log linear demand model with random walk and overall trend down.

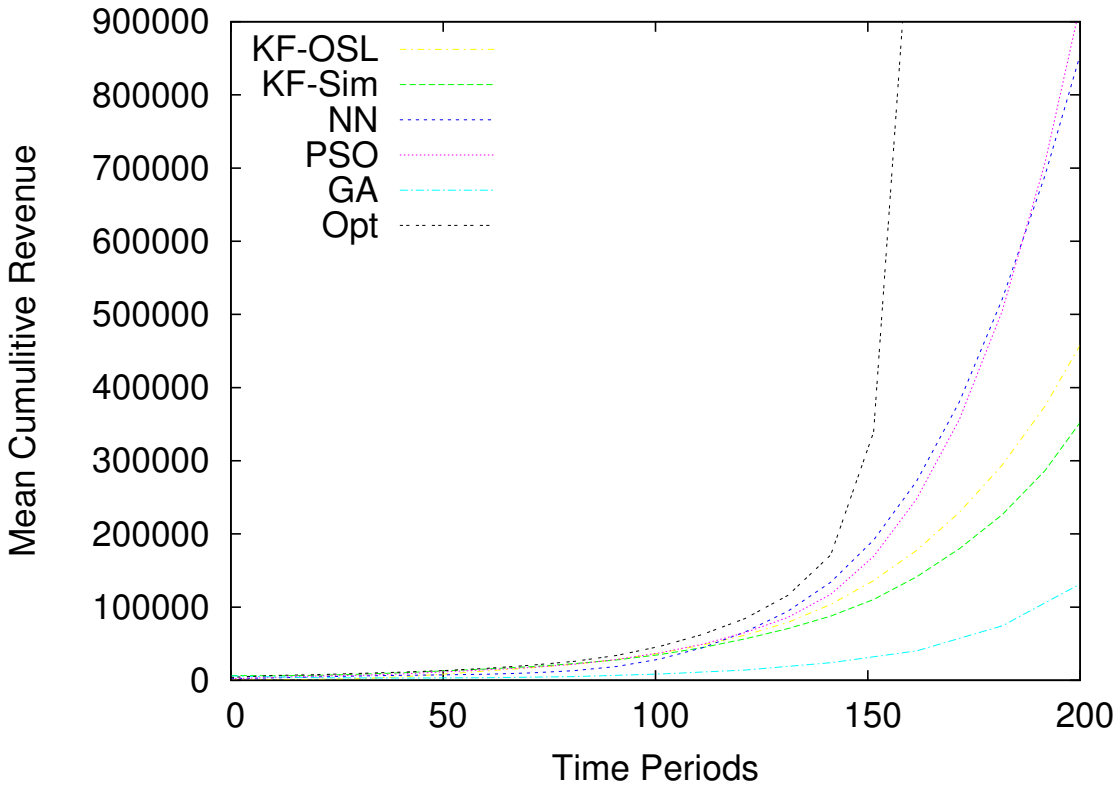


Figure 5.10: Log linear demand model with random walk and overall trend up. PSO handles upward trends very well, surprisingly the NN did as well as the PSO in the first 200 time steps. After that (not shown) the PSO outperformed the NN by a larger margin.

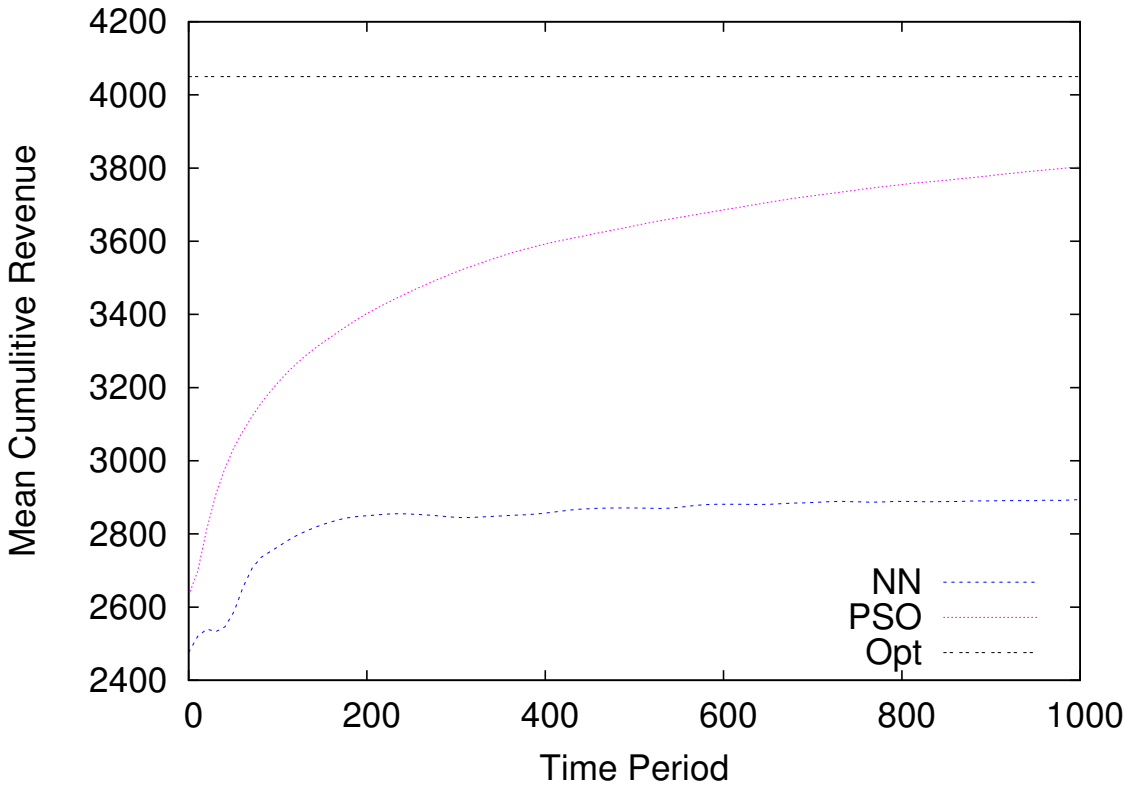


Figure 5.11: Multiple related products with log linear demand and constant parameters. The PSO algorithm did well to approach as closely as it did to the optimal in this more complicated environment.

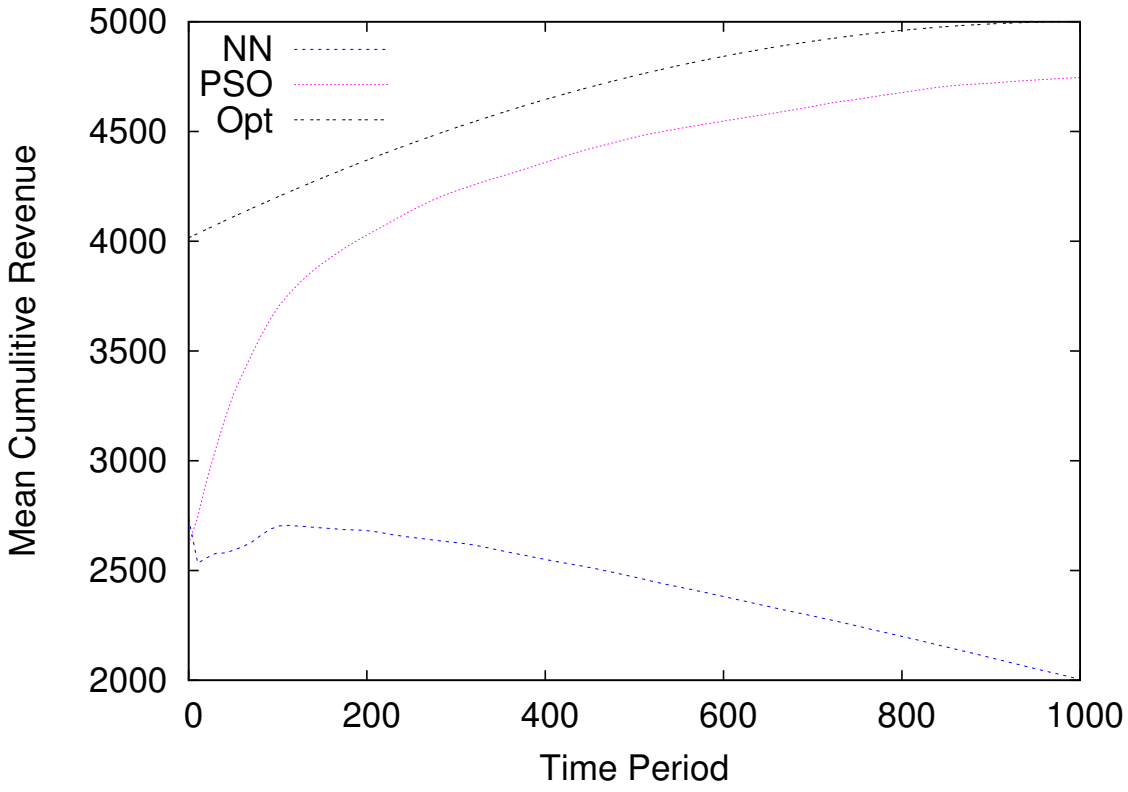


Figure 5.12: Multiple related products with log linear demand and parameters changing with time. The PSO was able to adapt to the changing environment while once again the dynamics caused the ANN's performance to degrade.

# Chapter 6

## Conclusion

This work is dedicated to applying machine learning algorithms to dynamic pricing. Each of the papers contained herein have drawn their own conclusions that are appealing in and of themselves. Section 6.1 will summarize those conclusions and comment briefly on the performance of Genetic Algorithms. Section 6.2 will devote itself to discussing conclusions from observing the work as a whole. Some possible future directions for research in this area will be presented in Section 6.3.

## 6.1 Summary of Results for Specific Algorithms

In order to have an intelligent discussion on the overall effectiveness of machine learning in dynamic pricing, the relevant points from the conclusions in the individual chapters of this work will be presented here. For more details on specific algorithms, their variations, and how they all performed in the modeled market place, please refer to the specific chapters dedicated to those algorithms.

**Markov Decision Process:** Using a Kalman filter to track the hidden parameters of demand is moderately successful in the dynamic pricing framework. Adding exploration, either through the one-step look-ahead function or by simulating what may happen in the future and finding the value of exploratory prices, allows the Kalman filter to do better. After exploitation begins, some random exploration can help the Kalman filter track a dynamic market better than a myopic pricing strategy.

**Artificial Neural Networks:** The neural net does well for dynamic pricing as long as the model does not change too much with time. This is shown by comparing the results from Figure 4.9 and Figure 5.7. In the more dynamic system, the neural net attained less revenue even though the earning potential was higher. The large amount of data required to train the neural network signifies that even if it accurately models the function it sees from the data, that function will largely be obsolete at the current time period. The neural network performed well in the linear model and not so well with the log-linear model. This may mean that the neural net would benefit from a different setup (change the number of hidden nodes, size of training data, etc.) depending on the market, making it not as flexible to any demand situation as we had originally hoped.

**Particle Swarm Optimization:** The greedy nature of particle swarms is good from an exploitation perspective, but in noisy situations overly optimistic observations will cause the swarm to converge incorrectly. The changes to PSO to allow it to handle noisy dynamic systems performed very well. Different variations performed best in each situation. PSO proved quite robust in changing to the more complex multiple product case. In fact, the adaptations (detection/response) that worked well in the single product environment translated to the multiple product environment.

**Genetic Algorithms:** Genetic algorithms as presented here suffer from the same problems that PSO has with noisy dynamic systems, however no alterations were made and the results reflect this.

## 6.2 Overall Conclusions

A dynamic noisy system, though interesting, makes a difficult problem for machine learning. Old data is quickly outdated. This is most problematic for ANNs, but even PSO and GAs are making decisions based on data that is no longer valid. This can be minimized by

keeping population sizes small and taking steps to ensure that some exploration continues after convergence. The Kalman filter has the advantage in dynamic situations because it uses the old data to track and predict where the parameters will be for the next price step. This advantage is based on how accurately the chosen model represents the true demand. When the model did a poor job of representing the market, the performance of the Kalman filter suffered.

A case can be made that all four main algorithms have an internal representation of the price/revenue relationship. However, two of the algorithms (MDP and GA) have that relationship defined in part by the user. In setting up the Kalman filter matrices some belief about how the tracked parameters correspond to the observations is necessary. For GAs, the function making the pricing decision has to know how the parameters $\alpha$ and $\beta$ that are estimated relate to which price should be set. As was shown with the Kalman filter, this can be an advantage when the model is correct, but a disadvantage if the representation is wrong.

This leads us to a conclusion that is fairly obvious; the more (correct) prior knowledge that an algorithm can incorporate before hand the better that algorithm will perform. This is true even in the case of PSO. Looking at the results in Section 3.7 shows that in every situation a different detection/response combination performed the best . Notice that in most cases the top three or four algorithms are variations of the same detection and response methods. This implies that the best detection and response combinations is dependent on the type of dynamics in the model, however this is not entirely negative for PSO. The algorithm used to generate the results in chapter 5 ($\text{Fixed}_{10}$-$\text{PBDPSO}_{99}$) outperformed everything in Figures 5.7 and 5.4, and was still very competitive in all others. Therefore, even though PSO may benefit from some information about the demand in choosing which detection and response methods to use, the detection and response method combination presented here will perform competitively with the Kalman filter in the majority of situations even without that knowledge.

Some of the algorithms presented here are naturally exploring and others were made to explore. In the case of the neural network, this forced exploration did not help. However, it did help for the Kalman Filter. The results show that the natural exploration of particle swarms helped in the log-linear case, except when the demand was held constant. When demand is not changing with time it is fairly obvious that exploration is going to hurt revenue since no extra knowledge is needed beyond the initial exploration.

## 6.3 Future Research

For the immediate future there are two directions that may give some improvements. The first would be to try a combined algorithm. A particle swarm could do the initial exploration and then this data could be used to train the neural network. If the accuracy of the ANN reaches a certain point, the ANN could then be used for the exploitation phase, until the system moves enough that exploration is again necessary. As part of this new, combined algorithm the detection and response methods that were developed for the PSO could be applied to the ANN in order to discover when exploration is necessary and how to handle this new exploration phase. The second direction would be to improve PSO in the downward trend. This would have the benefit of improving PSO in any changing parameter situation as long as the performance in the upward trend could be maintained.

The results from this work are exciting and may have wide application. There are interesting problems that may benefit from the contributions presented here. One example is in evolving control algorithms for rovers that operate in a dynamic noisy environment (since they are based on sensors and their fitness function is based on what all other rovers in their group are doing). A current solution uses Multi Layer Perceptrons that evolve with time [Tumer and Agogino 2005]. This research may benefit from either the PSO adaptation presented here to search the problem space, or by altering their current GA to use a similar detection/response algorithm. Other current research attempts to find the impact the strategies of large strategic traders have in a dynamic noisy market [Vayanos 2001]. This

research would possibly benefit from the simulation algorithm developed for the neural network.

# Bibliography

R. Bhar and S. Hamori. *Hidden Markov Models: Applications to Financial Economics*. Kluwer Academic Publishers, 2004.

T. M. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 19–26, San Francisco, CA, USA, 2002.

A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In *International Conference on Artificial Intelligence (ICAI 2000)*, pages 429–434, Las Vegas, Nevada, USA, 2000.

A. Carlisle and G. Dozier. Tracking changing extrema with adaptive particle swarm optimizer. In *5th Biannual World Automation Congress*, pages 265–270, Orlando, Florida, USA, 2002.

A. X. Carvalho and M. L. Puterman. Dynamic pricing and learning over short time horizons. Working Paper, University of British Columbia, Vancouver, BC, Canada, 2003.

A. X. Carvalho and M. L. Puterman. Learning and pricing in an internet environment with binomial demands. *Journal of Revenue and Pricing Management*, 3(4):320–336, 2005.

M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evolutionary Computation*, 6(1):58–73, 2002.

J. D. Cowan and D. H. Sharp. Neural nets and artificial intelligence. *Daedalus*, 117: 85–121, 1988.

M. Dequnes. Genetic. Genetic Algorithm package for python, 2003. URL `http://home.gna.org/oomadness/en/genetic/`.

R. C. Eberhart and Y. Shi. Tracking and optimizing dynamic systems with particle swarms. In *IEEE Congress on Evolutionary Computation (CEC 2001)*, pages 94–97, Seoul, Korea, May 2001.

A. C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Press Syndicate of the University of Cambridge, 1989.

J. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: University of Michigan Press, 1975.

X. Hu and R. C. Eberhart. Adaptive particle swarm optimisation: Detection and response to dynamic systems. In *IEEE Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1666–1670, Honolulu, Hawaii, USA, 2002.

V. Jayaraman and T. Baker. The internet as an enabler for dynamic pricing of goods. *IEEE Transactions on Engineering Management*, 50(4), November 2003.

I. Kaastra and M. Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236, 1996.

K. Kalyanam. Pricing decisions under demand uncertainty: A Bayesian mixture model approach. *Marketing Science*, 15(3):207–221, 1996.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks (ICNN 1995)*, volume 4, pages 1942–1948, Perth, Australia, 1995.

D. Kong. One dynamic pricing strategy in agent economy using neural network based on online learning. In *Proceedings of the Web Intelligence, IEEE/WIC/ACM International Conference on (WI'04)*, pages 98–102, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2100-2.

T. Krink, B. Filipic, G. B. Fogel, and R. Thomsen. Noisy optimization problems - a particular challenge for differential evolution? In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC 2004)*, pages 332–339, Portland, Oregon, USA, 20-23 June 2004. IEEE Press. ISBN 0-7803-8515-2.

X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 105–116, 2004.

X. Li and K. H. Dam. Comparing particle swarms for tracking extrema in dynamic environments. In *IEEE Congress on Evolutionary Computation (CEC 2003)*, volume 3, pages 1772–1779, Newport Beach, California, USA, 2003.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–137, 1943.

P. B. Mullen, C. K. Monson, and K. D. Seppi. Particle swarm optimization in dynamic pricing. In *Proceedings of the IEEE Congress on Evolutionary Computation (CECC)*, pages 4375–4382, Piscataway, NJ, July 2006. IEEE Press.

P. B. Mullen and K. D. Seppi. Dynamic pricing with artificial neural netowrks. Tech Report, Brigham Young University, Department of Computer Science, Provo, UT, USA, 2006.

P. B. Mullen, K. D. Seppi, and S. C. Warnick. Dynamic pricing on commercial websites: A computationally intensive approach. In *Proceedings of the 8th Joint Conference on Information Sciences (JCIS)*, pages 1001–1004, Salt Lake City, UT, July 2005.

S. Nissen. Fast artificial neural network library (FANN). Neural Network library for C++, 2006. URL `http://leenissen.dk/fann/`.

G. Pant, P. Srinivasan, and F. Menczer. Exploration versus Exploitation in Topic Driven Crawlers. In *Proceedings of the Second International Workshop on Web Dynamics*, Honolulu, Hawaii, USA, May 2002.

D. Parrott and X. Li. A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC 2004)*, pages 98–103, Portland, Oregon, 20-23 June 2004. IEEE Press. ISBN 0-7803-8515-2.

K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization for imprecise problems. In *5th international workshop on mathematical methods in scattering theory and biomedical technology*, Corfu, Greece, 2001a.

K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimizer in noisy and continuously changing environments. In *IASTED International Conference on Artificial Intelligence and Soft Computing*, pages 289–294, Cancun, Mexico, 2001b.

J. Pugh, Y. Zhang, and A. Martinoli. Particle swarm optimization for unsupervised robotic learning. In *IEEE Swarm Intelligence Symposium*, pages 92–99, 2005.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach - Second Edition*. Prentice Hall, 2003.

J. Shapcott. Index tracking: Genetic algorithms for investment portfolio selection. Report EPCC-SS92-24, Edinburgh Parallel Computing Centre, The University of Edinburgh, Edinburgh, U.K., 1992.

K. Tumer and A. Agogino. Evolving multi rover systems in dynamic and noisy environments. NASA's Ames Research Center's Intelligent Systems Division Publication, 2005. URL `http://ic.arc.nasa.gov/publications/1084.pdf`.

D. Vayanos. Strategic trading in a dynamic noisy market. *Journal of Finance*, 56(1):131–171, 02 2001. available at http://ideas.repec.org/a/bla/jfinan/v56y2001i1p131-171.html.

M. S. Voss and X. Feng. A new methodology for emergent system identification using particle swarm optimization (PSO) and the group method data handling (GMDH). In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 1227–1232, New York, New York, USA, July 2002.

M. K. Warmuth, J. Liao, G. Rätsch, M. Mathieson, S. Putta, and C. Lemmen. Active learning with support vector machines in the drug discovery process. *Journal of Chemical Information and Computer Sciences*, 43(2):667–673, 2003.

Y. Zhang, W. Xu, and J. P. Callan. Exploration and exploitation in adaptive filtering based on bayesian active learning. In *International Conference on Machine Learning (ICML 2003)*, pages 896–903, Washington, DC, USA, 2003.