2010-03-15

# The Role of Algorithmic Decision Processes in Decision Automation: Three Case Studies

Blake Edward Durtschi
*Brigham Young University - Provo*

# The Role of Algorithmic Decision Processes in Decision Automation: Three Case Studies

Blake E. Durtschi

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

April 2010

ABSTRACT

The Role of Algorithmic Decision Processes in Decision Automation:

Three Case Studies

Blake E. Durtschi

Department of Computer Science

Master of Science

This thesis develops a new abstraction for solving problems in decision automation. Decision automation is the process of creating algorithms which use data to make decisions without the need for human intervention. In this abstraction, four key ideas/problems are highlighted which must be considered when solving any decision problem. These four problems are the decision problem, the learning problem, the model reduction problem, and the verification problem. One of the benefits of this abstraction is that a wide range of decision problems from many different areas can be broken down into these four “key” sub-problems. By focusing on these key sub-problems and the interactions between them, one can systematically arrive at a solution to the original problem. Three new learning platforms have been developed in the areas of portfolio optimization, business intelligence, and automated water management in order to demonstrate how this abstraction can be applied to three different types of problems. For the automated water management platform a full solution to the problem is developed using this abstraction. This yields an automated decision process which decides how much water to release from the Piute Reservoir into the Sevier River during an irrigation season. Another motivation for developing these learning platforms is that they can be used to introduce students of all disciplines to automated decision making.

# Contents

# List of Figures

## List of Tables

# Chapter 1

# Introduction

Decision automation is the process of creating algorithms to automate decisions. These decisions can range from simple logical decisions, “if X, then do Y”, to complex real valued decisions, “based on the current state, sensory data, and likely future expectations, turn left 8.7 degrees, and slow down to 23 mph.” The goal in decision automation is to create algorithms that can take data as input and make provably good decisions without the need for human intervention. Although this task may seem daunting, a rigorous science has been developed to address several key issues in the leap from data to decisions.

An algorithmic decision process, (or simply, decision process), is a systematic process for making a decision. It can be thought of as the final result of decision automation. The decision process is designed as an algorithm that can be executed on a computer, on another device, or manually by another human.

In this thesis, we develop a decision architecture which can be used in decision automation. This architecture focuses on the key issues that must be considered in order to create algorithmic decision processes. Because it focuses on the most important issues, this decision architecture may be used to provide a pedagogical model that can be taught to students studying decision automation. In addition, we have also created three learning platforms that demonstrate how this architecture can be applied in three different decision areas: portfolio optimization, business intelligence, and automated water management.

The rest of this chapter continues with a section introducing this decision architecture, followed by a section describing the motivation behind our three new learning platforms. Chapters

2 through 4 describe the three decision areas and the learning platform for each. In chapter 2 we describe in greater detail how the decision architecture can be applied to a non-traditional, non-engineering decision area using the example of portfolio optimization. We also introduce a portfolio optimization platform. Chapter 3 describes the business intelligence platform, the problem it is designed to help solve, and how the algorithmic decision process can be applied to help solve that problem. Chapter 4 describes the automated water management platform, going into significant detail on the implementation of the platform and its solution on the Sevier River System in central Utah. Finally, chapter 5 concludes this thesis.

## 1.1 A Decision Architecture

When teaching about decision processes, we focus on algorithms and the way they use and transform data. To scrutinize decisions through this lens, we have borrowed four of the "great ideas" from computer science to create a decision architecture. This architecture breaks a decision process into modules that can be independently formulated and solved as meaningful problems. This separation is useful in creating a pedagogical sequence where students can attack pieces of the problem before trying to master the entire process. Although our ultimate goal is to emphasize the interrelationship between these stages, questions about this interrelationship can be easily motivated by understanding the ways specific algorithms for one stage consume information and transform it to be used by another stage of the decision process.

This decision architecture is based on the simple observation that any discussion of the quality of a decision must involve a discussion of the consequences of various choices. We will call the mapping of choices to consequences a model. Without a model of the consequences of our actions, we cannot talk meaningfully about good (or bad) decisions.

With this model-focus, we characterize the modules of making a decision as decision making, learning, model reduction and verification. See figure 1.1. Decision making is the process of computing decisions based on a model of the consequences of available choices, along with an objective function that scores possible consequences to characterize which decisions are better than others. Learning is the use of historical observations (data) and a measure of quality to choose a model from a class that best explains the data. Model reduction is the process of selecting a simplified model class from which an adequate model can be chosen. Verification then designs the experiments that test the decision process in ways that generate new data from which a specified metric can score the performance of the design. This process and how it relates to making decisions will be described in greater detail using portfolio optimization as an example in the next chapter.

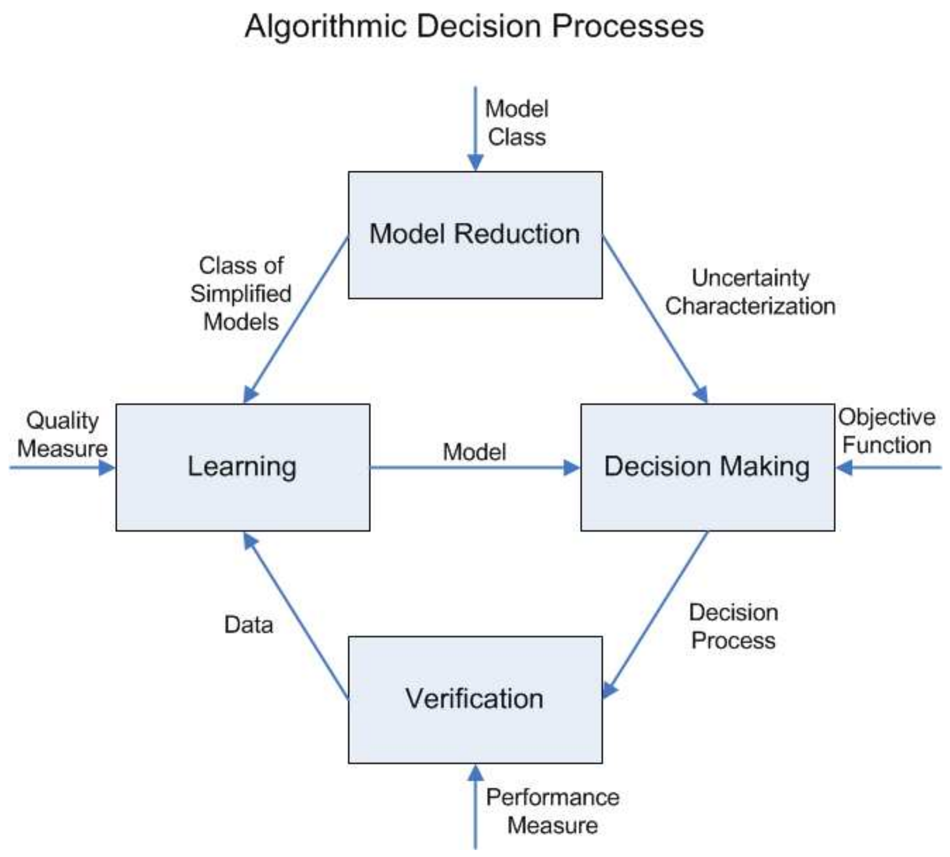


Figure 1.1: A decision architecture highlighting the central role of models in making decisions. Four major themes from computer science are highlighted as the primary stages of a decision process.

This decision architecture can be viewed as an abstraction of the field of decision automation. Many key ideas and topics in decision automation can be viewed as specific details, formulations, or ideas related to one or more of these four problems. Because of the generality of this architecture to the field of decision automation, it can be used in solving problems in many different decision areas that may seem very different. In this thesis we will show how this decision architecture can be used to develop algorithmic decision processes in order to solve three of these problems.

Learning platforms have been developed in the decision areas of portfolio optimization, business intelligence, and automated water management. Each platform focuses on a different part of the decision architecture. The portfolio optimization platform focuses on the verification portions of this architecture to try to verify various portfolio optimizing algorithms. The business intelligence platform focuses on the learning portion of this architecture because it involves students creating models for future sales of retail products. The automated water management platform focuses on the decision making part of this architecture in deciding how much water to release from a reservoir to downstream users.

Throughout this thesis we use the term architecture to refer back to this decision architecture which may be used to create algorithmic decision processes. The architecture includes the intereaction between these four key problems in decision automation. We will use the term learning platform, or platform, to refer to any software or hardware designed to be used by students in order to learn a specific idea. Thus the portfolio management problem is a problem to be solved by using our decision architecture, and the portfolio management platform is the software used in order to illustrate part of the decision architecture to students.

## 1.2 Laboratory Testbeds for Decision Automation

Learning platforms are frequently used in education in order to give students a place to experiment with concepts that can be more difficult to learn in a traditional discussion setting. Being able to “try it and see” allows students the opportunity to solve problems and then verify the quality of their solutions. Platforms also play a role in motivating the student, by making learning fun.

There has been much research devoted to designing and using learning platforms for education in algorithmic decision processes. Some of the most common learning platforms currently used come from the controls community and include the following: inverted pendula [30, 11], ball and beam systems [12], robotic arms [3], and other mechanical devices. A growing interest in multi-agent systems has likewise motivated team systems such as robot soccer [10] and other "bot" systems [1] that can execute various cooperation strategies to orchestrate efforts to accomplish a common goal. These systems can be powerful platforms for students to solidify their understanding of decision processes.

One drawback to common decision making platforms is that they require the students to have a mastery of concepts from physics, math, and engineering before they can explore the important problems in algorithmic decision processes. Because of this, students typically are not introduced to systems theory or decision algorithms before their junior or senior year. Another problem is that systems theory is usually only being taught to engineers while students from a wide range of technical areas could benefit from principles of systems theory.

Our view is that the central ideas from systems theory and algorithmic decision processes should be introduced to students much earlier in their education. This view is motivated by our observation that algorithm design aids any decision making process, thereby playing a foundational role in a broad range of applications and fields. Thus, students of all disciplines could benefit from an introduction or overview of algorithmic decision processes. Also, with earlier exposure, students will be able to decide sooner if they like the study of algorithmic decision processes and get a jump start on preparing for the rigors of the field.

To accomplish this, we suggest introducing various learning platforms in areas accessible to younger students. The goal is to simplify the context for discussing central issues in systems theory and algorithm design, providing jumping off points for students to appreciate and explore the depths of the field. Our point of view is that computer science is fundamentally about solving problems using computers. In the context of decision-making, this suggests making provably good decisions in the face of uncertainty and complexity constraints. The central problems associated with this algorithmic approach to decision-making are 1) the decision problem, 2) the learning problem, 3) the model reduction problem, and 4) the verification problem.

Another common trend in the teaching of decision processes is to use virtual platforms in order to simulate designs for decision algorithms. Koretsky et. al. at Oregon State University have been using a virtual laboratory of a simulated chemical vapor deposition process in their program and have shown it to be effective in teaching control theory. They report that students felt that it was the most effective learning medium used, even above physical laboratories [15].

All of our platforms described in this thesis contain both virtual and physical components. The portfolio optimization platform uses real stock data to create a virtual trading platform with the ability to add virtual dynamics. The business intelligence platform uses actual product sales data to drive a demand forecasting platform. The water management platform is built upon a real reservoir release gate and uses software to model a physical river and implement algorithms to control the gate. The virtual component makes each platform easily accessible to many students who can design their own algorithms in software, while the physical component of each platform makes the decision algorithm yield real world solutions that matter to real people. We believe that these real world applications tend to create a learning culture in which the student feels their work is important and motivates him/her to learn more in order to come up with better solutions.

While the primary use of these learning platforms is to introduce and demonstrate our decision architecture, we believe they can also serve to help students of all diciplines learn more about decision automation. During this thesis as we introduce and describe these platforms we will also emphasize how these platforms could be used by students in learning this decision architecture.

# Chapter 2

# Portfolio Optimization

The portfolio optimization platform is a stock trading platform where students can create trading algorithms which decide which stocks to buy and sell. Virtual cash is given to each algorithm or "automated agent", which then uses that cash to create a portfolio of assets. Live stock data is fed into the system which adjusts the values of each agent's portfolio accordingly. Our contribution, and the focus of this chapter is to explain the problem that is to be solved by students wishing to use this platform and how coming up with an algorithmic decision process for this problem leads students to ask, and answer, important questions in the field of computer science. While doing so, we will show that relatively simple solutions to this problem can be understood by younger students, while at the same time introduce them to complex and interesting topics currently being researched. This can motivate students early in the program to desire to continue in the field.

In the following sections we introduce and formulate the portfolio management problem. Then we discuss each piece of our decision framework and show how a student may formulate each problem as it applies to portfolio optimization. We start with the control or decision problem and formulate that. Then we formulate the learning problem. We discuss how uncertainty in the learning problem affects the approach that should be used for the control problem. Then we formulate the model reduction problem and discuss the verification problem introducing our learning platform as a verification technique for portfolio optimization

## 2.1 Portfolio Management

The question of portfolio management deals with choosing how to allocate money into different securities with the objective to maximize total wealth at some future time. We have chosen portfolio management as a learning platform because it is conceptually simple and because the objective is clearly parameterized in terms of equity returns. In this way, all questions of information and uncertainty can be posed in terms of what is known about the equity returns. This characterization allows us to reconsider the decision problem repeatedly as we peel back different layers of information and study the impact of uncertainty on our problem.

Moreover, since this problem is open loop, in that investment decisions do not affect future equity returns of the assets (for typical investors), key concepts from systems theory can be introduced without the complexity of feedback interactions. The hope is that students will be motivated to engage the rigors of the discipline necessary to master feedback control if they first appreciate some of the central problems arising from the interaction of uncertainty and complexity in decision problems. Next we describe the portfolio management problem.

Suppose an investor has a choice between holding his money in a risk free cash account with a fixed, positive rate of return, or purchasing any of $n-1$ securities having varying (positive and negative) rates of return. Any of these securities may be purchased at any time, and all that is known about them is their historical price over a finite period of time, $p_i(t)$, $i = 1, \ldots, n$. The goal is to make as much money as possible at time T, by purchasing shares in these securities with a fixed initial investment. The following definitions will help make this objective precise.

A portfolio is a distribution of wealth invested in these assets, characterized by $(s_1(t), \ldots, s_n(t))$, where $s_i(t)$ is the number of shares of security $i$ owned at time $t$. We denote the value of the shares of security $i$ owned at time $t$ by $x_i(t) = p_i(t)s_i(t)$. The value of a portfolio at any particular time is the sum of the value of the securities, $x_1(t) + \ldots + x_n(t)$. We let $x_1(t)$ correspond to the value of the risk free cash account. The total return of a security is the price change ratio of the security, given by $r_i(t) = \frac{p_i(t)}{p_i(t-1)}$. This quantity characterizes how the value of a fixed number of shares changes over time. The resulting dynamics of the value of a portfolio over time are given by

$$\begin{bmatrix} x_1\,(t+1) \\ \vdots \\ x_n\,(t+1) \end{bmatrix} = \begin{bmatrix} r_1(t+1) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & r_n(t+1) \end{bmatrix} \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix}. \tag{2.1}$$

Nevertheless, the investor does not have to keep a fixed number of shares in each security. Instead, he can change the distribution of wealth between the securities at each time step. This decision is represented by a set of numbers, $u_i$, $i = 1,\dots,n-1$ that indicates the dollar amount that the investor wishes to be moved from the cash account to the $i$th risky asset. A negative value of $u_i$ represents a dollar amount to be moved from the $i$th risky asset to the cash account. We will assume for ease of exposition that there are no transaction costs, although all of the ideas discussed here can be easily extended to include them. The portfolio dynamics incorporating this investor decision then become:

$$x(t+1) = R(t+1)x(t) + R(t+1)Bu(t) \tag{2.2}$$

where $x(t) \geq 0\ \ \forall t$, $R(t+1) = diag(r_1(t+1),\dots,r_n(t+1))$,

$$B \;=\; \begin{bmatrix} -1 & -1 & \cdots & -1 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

The dynamics in (2.2) describe how the value of a portfolio changes as a function of the investor's reallocation decisions, $u(k)$. In this expression, the return matrix, $R(t+1)$, represents the *future* impact of an external variable over which the investor has no control, while the input, $u(t)$, represents the change in the portfolio distribution over which the investor has complete control. Note that the positivity constraints on $x$ restrict admissible decisions $u$, allowing the purchase of securities only if you have the money to pay for them. This simple context then motivates a very basic decision problem.

## 2.2 The Decision Problem

The decision, or control problem is to decide which input best improves the performance of the system being controlled. In order to discuss making good decisions we must know the consequences of our decisions and we must have an objective function which ranks the consequences by what is most desirable. Optimal decision making therefore is simply computing the choice with the "best" consequence as defined by the objective function. With perfect information about the consequences of our decisions, the control problem becomes a search over possible choices to select the one that best supports the objective. For in-depth information on control see [8].

The control problem naturally surfaces in any discussion of portfolio optimization when we consider the decision to be made by the investor. At some initial time, $t = 0$, the investor has an initial amount of cash on hand and no money invested in other securities. Thus his initial portfolio is $x(0) = \begin{bmatrix} x_1(0) & 0 & \dots & 0 \end{bmatrix}^T$. The investor's objective is to allocate his money into different securities at each time step in order to maximize at some future time, $T$, the total value of the portfolio, $\|x(T)\|_1 = x_1(T) + \ldots + x_n(T)$. Stated formally,

$$\begin{array}{rrcl} \max\limits_{u_i(1,\ldots,T)} & \|x(T)\|_1 & & \\ \text{subject to} & x(t+1) & = & R(t+1)x(t) + R(t+1)Bu(t) \\ & x(0) & = & \begin{bmatrix} x_1(0) & 0 & \dots & 0 \end{bmatrix}' \\ & x(t) & \geq & 0 \quad \forall t. \end{array} \tag{2.3}$$

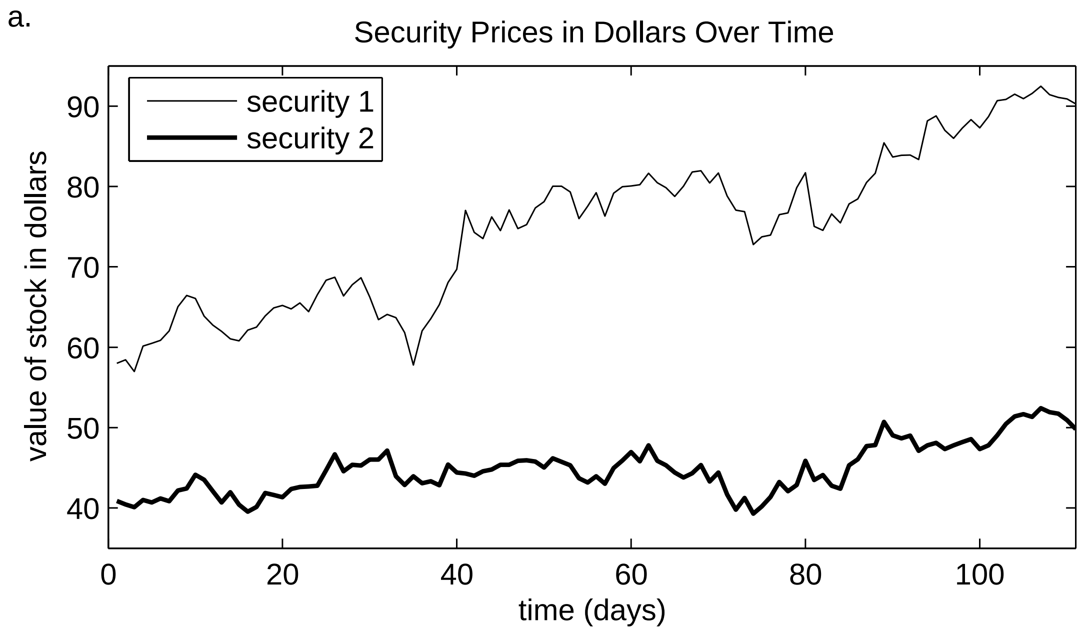
A student may easily discover that iteratively solving this problem for one time step will yield an optimal solution to the problem for multiple steps. This is because the decision for each step is independent of the distribution chosen in the previous time step. A suboptimal choice from one time step cannot lead to a better result in the second time step. This allows the student to reduce the problem to a sequence of one step problems. Note that this would not be true when considering the case where there are transaction costs.

**Example 1** (Perfect Knowledge of Consequences) *Consider the problem where $r_i(t)$, $i = 1, \dots, n$ and $t \in [0, T]$ is given. This problem corresponds to the situation where an investor has perfect knowledge of the future returns. To maximize the value of the portfolio one needs to move all of the money to the security with the highest return at each time step.*

*Suppose we can invest in two different securities or keep our money in a cash account. We will start out with \$100 and let $x_1$ be the amount of money kept in the cash account and $x_2$ and $x_3$ be the amount of money invested in the risky securities. Figure 2.1a shows the value of the two securities over a 110 day period. Figure 2.1b shows the composition of the optimal portfolio over time as it switches all of the money between the three funds.*

Having perfect knowledge about the future resulted in a simplification of this otherwise complex decision problem. This simplification can lead students to consider important questions about decision making. For example, can you characterize the nature of optimal solutions to make their computation tractable? In other words does the optimal design have an analytic solution? How does computational complexity change when considering a sequential decision process where making choices that appear suboptimal now may result in a higher future payoff? For example, sacrificing a piece in chess to obtain a better board position.

Until now we have assumed perfect knowledge of the future. When we do not have perfect information about the consequences of our actions, we need to estimate a best guess of what the consequences may be in order to make decisions consistent with our objective. A model represents everything we understand about the mapping between choices and consequences. To determine a model for future consequences, students must solve the learning problem.



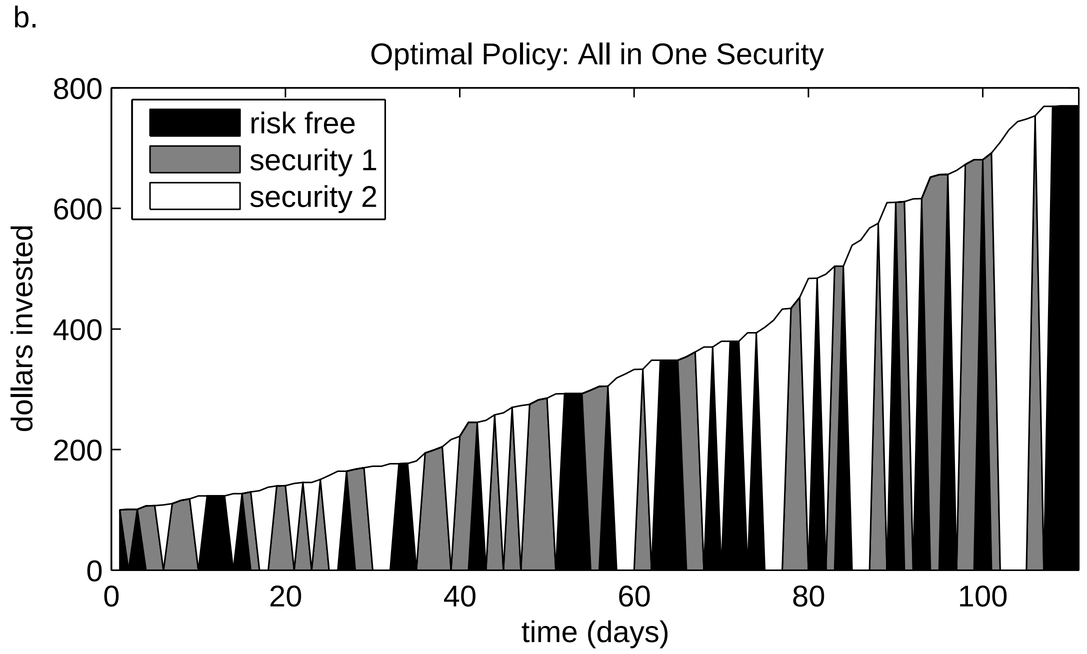


Figure 2.1: Prices of two different securities shown above. Assuming perfect knowledge of the future, the optimal policy switches all the money to the security with the highest return at any given time.

## 2.3 The Learning Problem

Formulating a learning, or system identification problem implicitly assumes that something about the mapping of decisions to consequences remains constant over time. Certainly if the relationship from choices to consequences is changing sporadically over time, there is no justification for using historical observations of the system to predict future behavior. This constancy between decisions and consequences is captured by the choice of model class. The knowledge we have about the true system's behavior is acquired by running experiments that collect input-output data. We use a quality metric to evaluate each model in the class based on this data.

Given a class of models, input-output data, and a quality metric, the system identification problem is to select the model from the class which best describes the observed input-output data according to the quality metric, see [18]

### 2.3.1 Sources of Uncertainty

Once a model is selected it becomes the basis for predicting consequences of various decisions. Inaccuracies in the predictions of this chosen model can come from insufficient data, the model class, or the quality metric. These sources of uncertainty are easily seen in portfolio management.

#### Uncertainty from Insufficient Data

**Example 2** (Uncertainty from Insufficient Data) *Suppose we have a risky security. We no longer assume that we have perfect knowledge about the future returns but that they are a stochastic process of independent identically distributed random variables. We select our model class to be the class of Gaussian distributions which are parameterized by their mean and variance. Our system identification algorithm is to choose the model whose mean and variance most closely match the sample mean and variance of the historical data.*

*Suppose the past returns from a risky security are truly generated from a Gaussian distribution with mean, 1.10, and variance, 0.10. By taking historical data as our sample we can compute*

Table 2.1: Estimates of the mean and variance of a distribution when taken from finite sample sizes are not always accurate.

| | Actual | Estimated | | | | |
|---|---|---|---|---|---|---|
| sample size | | 5 | 15 | 50 | 100 | 1000 |
| mean | 1.1 | 1.155 | .989 | 1.134 | 1.122 | 1.088 |
| variance | .1 | .040 | .148 | .143 | .109 | .102 |

*an estimated mean and variance. As shown in Table 2.1, the estimated mean and variance change depending on how large a sample we use.*

This example shows that with finite input-output data the learned model will be different from the true system, which introduces uncertainty into our predictions. As the available historical data increases, the sample mean and variance, and thus the selected model, converge to the same as the true system. This consistency is an indicator of a good system identification or learning algorithm.

**Uncertainty Intrinsic to the Model Precision**

Considering the previous example, suppose we had enough data that our system identification algorithm could select the correct model from the model class. From table 2.1 we would select a model with mean 1.1, and variance, 0.1. If we were to use this model, would our predictions of future returns necessarily be accurate? In this case, predictions of future returns are the mean of the Gaussian model chosen as most descriptive of the historical data. The next data point is not likely to be 1.1, but instead could be anything, say 1.195.

The model class chosen emphasizes a level of uncertainty in these predictions, characterized by the variance of the Gaussian distribution. We can expect our predictions to be accurate, on average, but within a range specified by the variance of our model. Thus, our particular choice of model class builds in an estimate of the level of precision of our predictions, and this precision defines uncertainty intrinsic to our particular model.

#### Uncertainty Due to Inaccurate Choice of Model Class

Another source of uncertainty is in the selection of the model class itself. While we may choose to represent returns as a Gaussian random variable, it may, in fact, be generated by a completely different process for which a Gaussian process is only an approximation.

#### Uncertainty Due to the Quality Metric

Suppose we have enough data such that our sample mean is 1.1, but we change our quality metric to choose the model which more heavily weighs the most recent week's data. The model selected will not be the same as the actual distribution mean, as our choice of metric reveals our assumptions about the predictive value of the model. Likewise, one may compute a "best fit" between model predictions and data using different norms, and the minimizer in each case may be an entirely different model from the chosen model class.

Frequently the choice of metric is used to guard against uncertainty in the choice of model class. This is accomplished, for example, by ranking each model in the class by the likelihood that it generated the observed data, and then choosing the model that maximizes this likelihood.

Consideration of these uncertainties in modeling leads students to a variety of important questions. How should one characterize uncertainty in a model, and how should this characterization change the control problem formulation? How does one design tests that check whether the underlying assumptions justifying a particular choice of model class, metric, etc. are still consistent with the observed data?

### 2.3.2 Uncertainty's Effect on the System Identification and Control Problems

Until now we have discussed the control problem and the system identification problem separately. Because of uncertainty, it is interesting to consider how these problems affect one another. Given a predictive model, when does it make sense to treat its predictions as the true future, leaving the formulation of the control problem unchanged? When does it make sense to modify the formulation of the control problem to account for the fact that our estimates of the consequences of various decisions are based on a model, not perfect knowledge? Likewise, should knowing that our choice of a model will be used as the basis for decision making alter the nature of the identification problem? If so, how?

One approach to resolving these issues suggests that both the system identification and control problems should be modified to account for their impact on each other. For example, some control-oriented system identification modifies system identification techniques to be compatible with robust control methodologies [5]. Likewise, robust control can be viewed naturally as identification-oriented control because it modifies classical control techniques to account for explicit uncertainty in the learned model. In the portfolio management example, these issues arise naturally as the objective function in the control problem is modified to account for uncertainty in the predictions based on the identified model. The degree to which the objective function is modified to account for this uncertainty can be scaled by a risk aversion parameter, facilitating an entire class of control problems depending on the degree one is willing to believe the predictions of the identified model.

**Example 3** (Control Problem Accounting for Model Uncertainty) *Now we assume that we have uncertain predictions of future returns and show how the decision problem is different from the decision problem with perfect knowledge of the future. We let $J$ be our one step decision problem without uncertainty and then consider how uncertainty in the returns affects the solution. $R(t+1)$ and $B$ are defined in equation (2.2).*

$$\begin{aligned} J = \max_{u_i(1,\ldots,T)} \quad & \|x(t+1)\|_1 \\ \textit{subject to} \quad x(t+1) &= R(t+1)x(t) + R(t+1)Bu(t) \\ x(t) &\geq 0 \quad \forall t \end{aligned} \tag{2.4}$$

*This optimization requires predictions for the future returns, $r_i(t+1), i = 1, \ldots, n$ (used in $R(t+1)$). Assuming that we solve an identification problem to find a Gaussian distribution that best explains the observed data, then we could use the mean for each security as the predicted* *return for that security. However, this would be equivalent to assuming that our identified model is perfectly accurate.*

*One way to formulate the control problem that accounts for our uncertainty in the predictions of our model is to say that each predicted return $\hat{r}_i$ is never more than $\varepsilon$ away from the actual return, $|\hat{r}_i - r_i| \leq \varepsilon_i$. This gives us a range of possible values for $J \in [J_{lo}, J_{hi}]$. Now we need to rethink our objective function based on the ranges of $J$ for each possible combination of $u_i$'s. We can think of protecting against the worst case by maximizing the minimum value of $J$. We could also maximize the maximum, the median, or some other value of $J$; we could minimize the difference $J_{hi} - J_{lo}$, etc. In each of these situations, considering the sense in which the resulting investment strategy is "best" encourages students to think deeply about the interaction between identification and control.*

*Another way to formulate the control problem to account for uncertainty in the identified model is to modify the objective to not only maximize predicted returns, but also to minimize the uncertainty intrinsic to these predictions. This is easily accomplished in the case where we model returns as Gaussian random processes. The error in our predicted returns, $|\hat{r}_i - r_i|$, is normally distributed, $N(0, \sigma_i)$. This causes $J$, defined in (2.4), to be normally distributed with mean, $\mu_J$, and standard deviation, $\sigma_J$, taken from the covariance of the individual returns. Our objective is then*

$$\max_{u_1(t),\ldots,u_{n-1}(t)} \mu_J - \lambda \sigma_J^2. \tag{2.5}$$

*where $\lambda$ is a user defined risk aversion parameter which can be adjusted to accomodate more or less risk. If lambda is low then the optimization will select the portfolio based on the best return regardless of the uncertainty. If $\lambda$ is high then portfolios which have high variance, or risk, will not be selected. This problem is the celebrated Markowitz model, which is commonly used in portfolio optimization [19] [21].*

**Example 4** (Mean Absolute Deviation Model) *Yet another approach may discount for uncertainty differently, by only considering the down-side risk of a particular investment. One way of doing this* *would be to let $\mu_J$ be the expected value of $J$ based on the expected returns $\hat{r}_i$, and a fixed portfolio specified by $u(t)$. Then for each previous time, $k = 1,\ldots,K$, we compute $y(k) = \mu_J - J(k)$, where $J(k)$ is how well our same portfolio would have performed at time $k$. $y(k)$ gives us a measure for how much our portfolio would have underperformed our expectation in the past. We then let our uncertainty be the average of the $y(k)$'s for all $k$. Once again using $\lambda$ as a risk aversion parameter gives us,*

$$\max_{u_1,\ldots,u_{n-1}} \mu_J - \lambda \frac{1}{K}\sum_{k=1}^{K} y(k), \tag{2.6}$$

*which essentially yields the Mean Absolute Deviation (MAD) portfolio optimization model [14].*

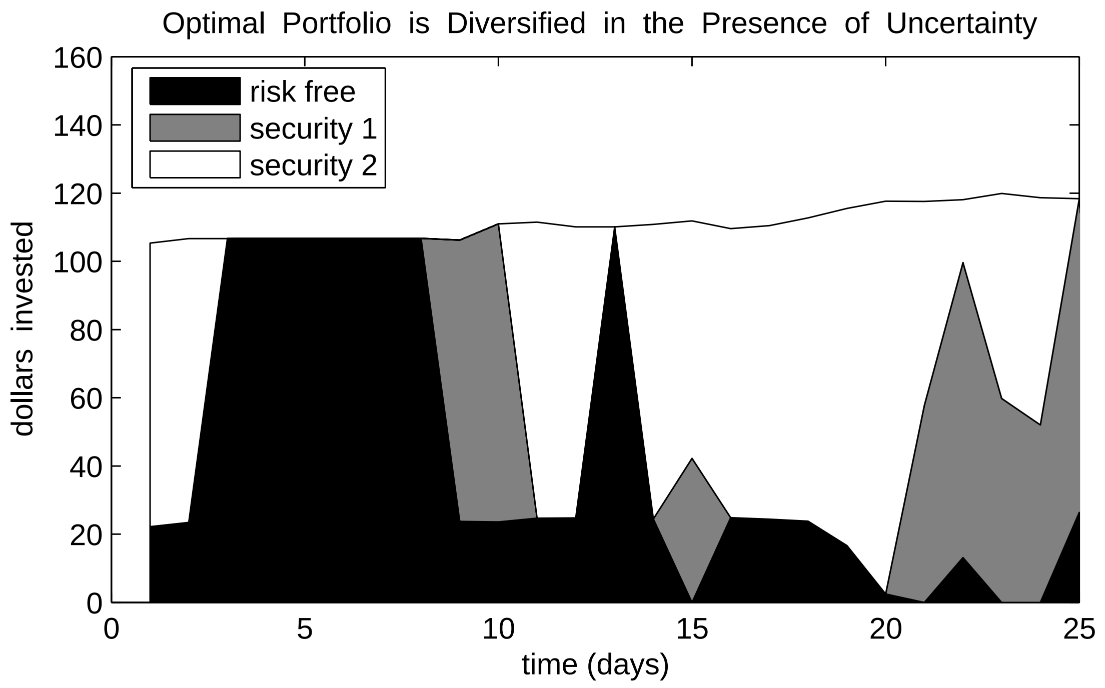


Figure 2.2: Using a system identification algorithm to predict future values with uncertainty, yields an optimal risk/reward portfolio that is diversified. These 25 days correspond to the last 25 days from Figure 2.1.

*Using the same three securities that we had in Example 1, but now with uncertainty, Figure 2.2 shows an optimal policy that allows a limited amount of risk. Unlike in Figure 2.1 when we had perfect knowledge of the future, this portfolio is* diversified, *splitting money between multiple* *securities to reduce risk. The return from Figure 2.2 is about 15% compared to about 50% in the optimal portfolio over the same time.*

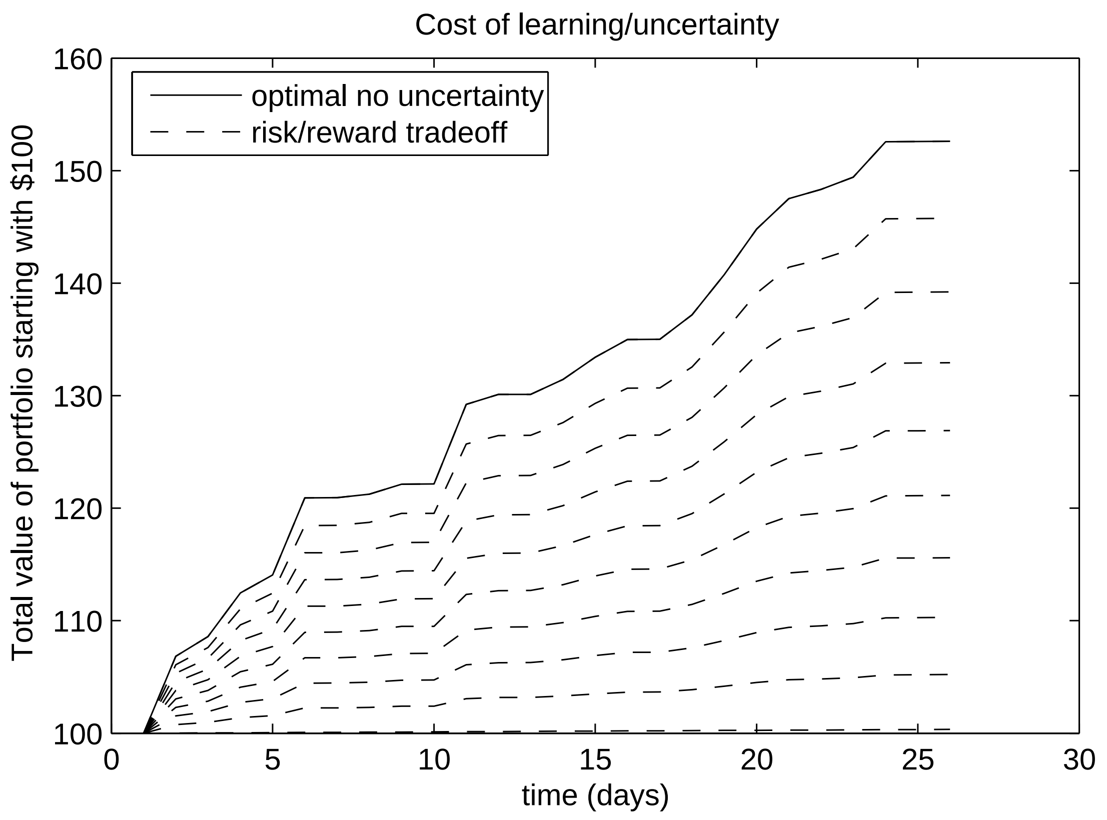


Figure 2.3: The cost of uncertainty compared to the optimal strategy. The dotted lines represent portfolios with different risk tolerances. Because of uncertainty in the future returns, portfolios with lower risk tolerance had to diversify more. Given that the predictions were correct, diversification lowers the optimal performance. In other words robustness incurs a performance cost.

It is also interesting to consider how uncertainty affects the possible solution. For example, suppose that we have an identification algorithm with uncertain predictions that happens to predict the returns exactly. Since we do not know ahead of time that they will be exact predictions, the risk/reward decision strategy will diversify the portfolio more or less depending on the uncertainty in the predictions. Figure 2.3 shows the loss in returns due to this diversification. By limiting the incurred risk of a portfolio, the maximum potential is lowered as well. Viewing diversification as a way to overcome uncertainty makes portfolio optimization an interesting platform for studying control.

## 2.4 The Model Reduction Problem

Until now we have assumed that the true system was contained in the model class chosen for system identification. There are however, many problems associated with this assumption. First, it is almost impossible to validate. At most one can only show that the selected model is not invalidated by the data from the true system. Second, models which can actually describe the true system are usually too complex to be efficiently identified or used in computing decisions. Third, complex models may require more data than is available in order to be identified. That is, they may be characterized by a large number of parameters, which demand a lot of data to estimate.

The model reduction problem centers around reducing complexity while retaining as much accuracy as possible. The problem states that given a complex system, $G$, find a simpler system from a class of simple systems, $\hat{\mathbf{G}}$, such that the difference between them is minimized in some norm.

$$\inf_{\hat{G}\in\hat{\mathbf{G}}} \|G-\hat{G}\|_n \tag{2.7}$$

See [8] for more discussion on Model Reduction.

Another way to reduce complexity is to lower computational complexity by decreasing the size of the control and system identification problems. An example of this for portfolio management is to limit the number of stocks being considered. This will avoid introducing the uncertainty that comes from simplifying the model class, however, it may lead to decreased performance because some of the previous options will no longer be available for the decision algorithm. The next example shows many ways of how approximation may be done by students for portfolio optimization.

**Example 5** (Reducing Complexity in Portfolio Choice) *Continuing previous examples where we have a simple model for future returns, the Markowitz formulation (2.5) for the control problem must compute the covariance of each possible portfolio. This requires a quadratic program to solve the optimization, which is NP-hard. Thus, the solution can not be solved quickly for a large number of securities. The literature is rich with methods for approximating the Markowitz portfolio [25],*

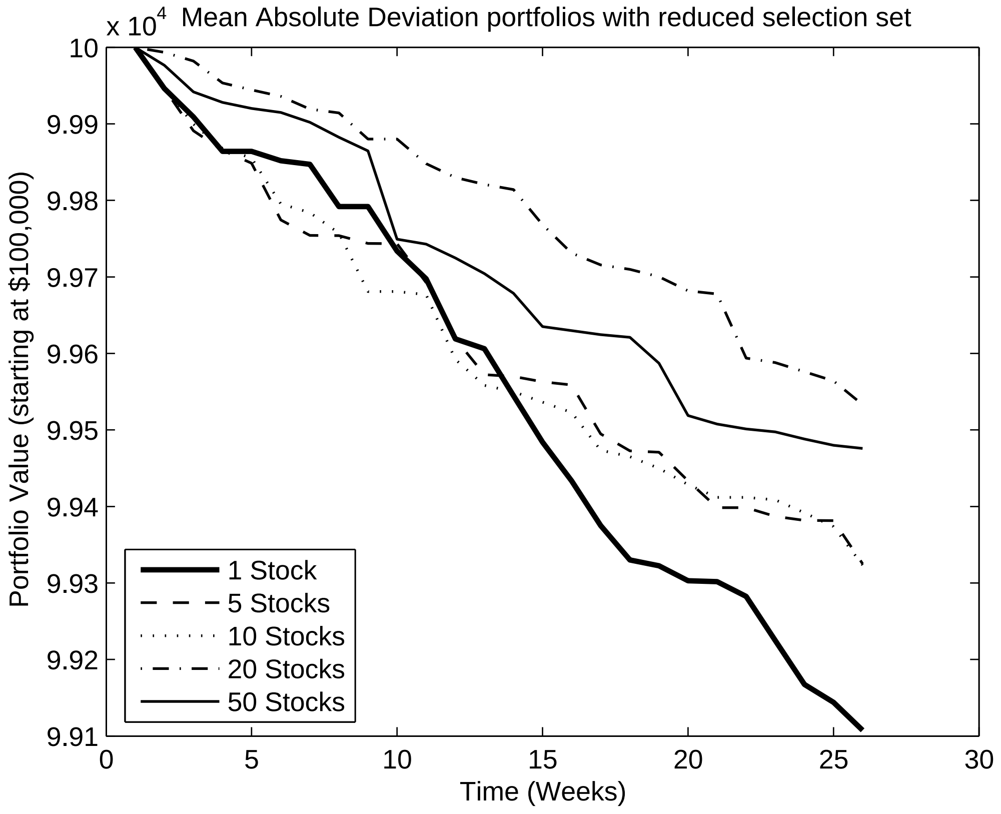


Figure 2.4: An experiment of the performance of the MAD model on progressively limited subsets of stocks. Students must answer the question, how does limiting the number of stocks affect the performance of the decision algorithm? The downward trend reflects the bear market that has been prevalent over the past year.

*[26]. One might also consider simplifying the problem by using an alternative formulation such as the MAD model which can be computed with a linear, instead of a quadratic program. Linear programs can be solved in polynomial time.*

*Another alternative for reducing complexity is to limit the number of securities available for selection. One might take the top n performers over the last period of time, or the lowest valued in the Dow Jones Industrial Average, etc. In our example, we limit the number of stocks under consideration from a progressively smaller subset of securities in the American Stock Exchange. Figure 2.4 shows how the value of a MAD portfolio is affected when choosing subsets of securities of different sizes. In particular, we see that it is non trivial to restrict stocks to a subset while maintaining the best possible performance for a problem of the specified size or complexity.*

The portfolio optimization problem introduces students to consider some important questions in approximation. For example, what approximations can be made that keep solutions close to non-approximated optimal solutions? How does limiting the number of securities affect the computational complexity as well as the performance of the decision algorithm?

## 2.5 The Verification Problem

Once the solution to a control problem has been formulated and has been approximated where necessary to ensure that a solution can actually be computed, some very important questions remain. How do we know that the approximated model is still precise? How do we know that the control solution works as desired? How do we convince non-technical people that our methods are sound? These questions motivate the final major problem in control that our platform introduces to students.

The verification stage monitors the performance of a complete solution, including the control, identification, and reduction steps. We call such a complete solution an *algorithmic decision process*. Verification determines whether we observe any new evidence that the assumptions justifying many of the choices made in the design of the algorithmic decision process have been violated, thereby motivating a redesign of the solution. Note that we can never prove that a particular design will always work, we simply look for evidence that it begins to fail [24].

Over time verification has become an increasingly important field of research. As controllers are implemented in software and hardware it becomes imperative to verify that these controllers will work as designed. In designing system-on-chip solutions, for example, 70% of the effort is spent on verification [27]. Because of this great effort required, verification should be considered early-on in the design of solutions to the control, system identification, and model reduction problems.

**Example 6** (Verifying a Solution for Portfolio Optimization) *In the portfolio optimization problem we want to use verification to determine whether our decision process is able to select portfolios* *that are better than the competition. One first attempt can be to run on past data to determine whether the algorithm performs above a specified benchmark.*

*In addition to verification with past data, running algorithms against others provides a useful means of verification. Many virtual fund management systems have a way to compare algorithms against each other in hopes of determining which is better. Brigham Young University's Tour de Finance (see figure 2.5) is a platform that allows for student-defined competition dynamics as a particular verification method [6].*

As students go through the control design process, the Tour de Finance platform gives them a fun competition which will also serve as a verification mechanism. Students can create leagues where algorithms can compete against each other. By seeing which algorithm performs better over time, they can determine which algorithm is better.

## 2.6 Platform Description

The platform itself consists of two parts. The first part is an online stock trading platform. The second part is a library of functions that an automated agent can use in order to interact with the platform. Students create automated algorithms which decide which stocks to purchase in order to maximize total portfolio value. The platform gives each agent $100,000 in virtual money and each agent decides which stocks to purchase. In order to complete a transaction the agent calls an API in the platform library which describes how many shares to buy or sell. The platform receives current stock prices from finance.yahoo.com and uses that data to adjust the portfolio value of each agent. The platform graphs each agent's portfolio value over time and students can see which agent is performing better.

There are several things that separate this platform from other virtual fund trading platforms. First it has an interface for automated agents. This encourages students to really try to quantify a superior trading algorithm based on data rather than just picking stocks manually. Second, the platform is designed to use competition dynamics in determining the value of a portfolio

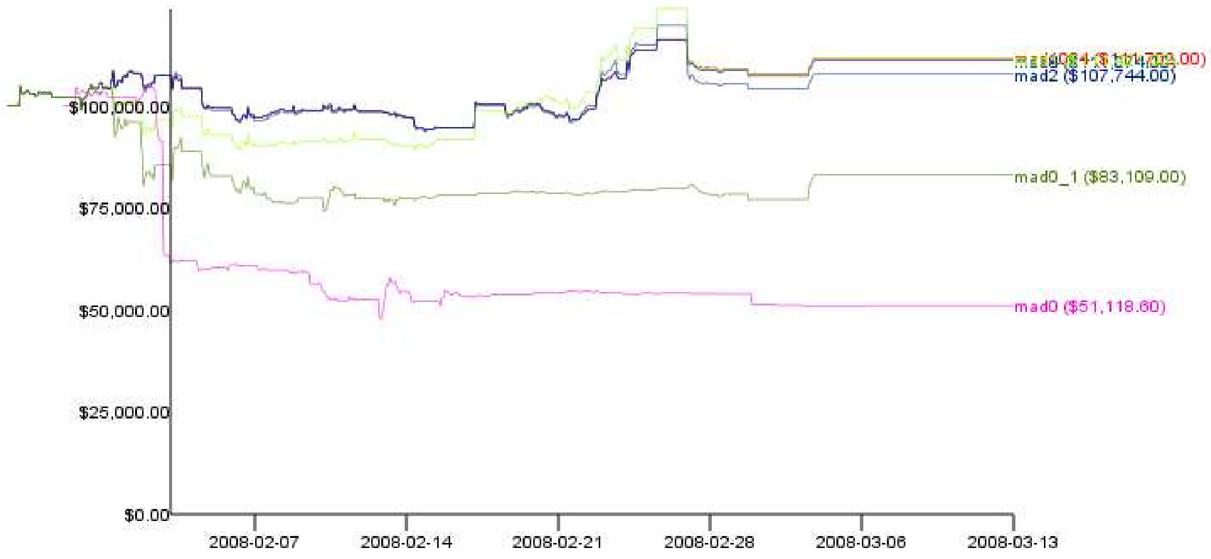


Figure 2.5: Screenshot from the Tour de Finance, a virtual trading platform for portfolio optimization. Shows several competing algorithms from the same MAD model for portfolio optimization but with differing risk aversion parameters, and hence, differing levels of diversification.

instead of just the value of the stocks. The platform will redistribute wealth according to the comparative performance between the agents. So if one agent is making money but it makes less than the other agents the platform will take some money from that agent and give it to the other, better-performing agents. This mimics the market share of a large trading firm which under-performs compared with other firms. The firm will lose investors who then invest their money in those other firms. See [6] for a better description on how this works as well as the motivation behind this design. Third, the platform allows for different groups to be created and each group can be set up with different dynamic and rules such as trading costs, etc. These groups can be created by students in order to study different dynamics, and investment strategies.

## 2.7 Conclusions

We have introduced portfolio management as a learning platform designed to teach students about the decision making process and introduce them to important controls problems earlier in their education. We walked through the example of portfolio management, showing how questions encountered by trying to select an optimal portfolio led to deep questions in the problems of control, system identification, model reduction, and verification. Moreover, classic results in finance such as the Markowitz model lead to exploring the interaction among these problems.

# Chapter 3

# Business Intelligence

Just like portfolio optimization, business intelligence is a great problem domain for studying algorithmic decision processes. Business leaders are in charge of making decisions every day. One important type of decision that retail business leaders must make is to set the price at which to sell each product. Retail managers must make a trade off between profit margins on each sale and quantity sold. As the price of an item increases the retail store makes more profit, but the number sold will be less. As the price of an item is lower, more items will be sold, but there will be less profit earned on each one.

In this chapter we describe how the decision architecture formulated at the beginning of this paper can be applied to this business intelligence problem. Unlike the previous chapter on portfolio optimization where we described in detail on how a student would step through the decision architecture to solve the problem, we will only go through the decision architecture enough to show that this problem can be approached by students in the same way. Then, we will describe the business intelligence platform that we have built in collaboration with the BYU Bookstore. This platform currently uses live bookstore sales data to address the learning stage of the process by developing models of product demand.

## 3.1 Applying the Decision Architecture to Business Intelligence

In this section we will describe how the decision framework developed in chapter 1 can be applied to the business intelligence problem where a manager must decide the selling price for each product in order to maximize profit. Just as in the portfolio optimization problem, the business intelligence problem can be broken down into four interconnected sub-problems.

### 3.1.1 Decision Problem

To describe the decision problem formally we will suppose that a retail store has a predetermined list of $n$ products for sale. Let vector $p$ describe the sale prices of these products where $p_i$, $i = 1,\ldots,n$ is the price of product $i$. We will let $q_i = d_i(p), i = 1,\ldots,n$ denote the quantity of product $i$ sold per unit time at given prices, $p$, where $d_i(p)$ is some function of prices (and is often called the demand function). The revenue generated from selling a product, $i$, is given by $r_i = p_i * q_i$, which is the sale price multiplied by the quantity sold.

We let $c_i$ be the marginal cost to the retail store for making a unit of product $i$ available. For simplicity we let this be the cost of purchasing the product from the supplier. It could also be expanded to include the cost of storing the product in inventory, shipping costs, or worker's salary associated with this product. We define expenditures, $e$, as the cost of an item multiplied by the quantity sold, $e_i = c_i * q_i$. We use the quantity sold and not the number of items purchased from the supplier because until the items are sold they are still maintained in inventory and so their value is still owned by the retailer.

The goal of the retail manager is to maximize profit, where profit is defined as revenue minus expenditures. Profit is given by, $P = (p-c)q$, and the decision is given in (3.1). The quantity sold is determined by $D(p)$, where $D$ is a mapping from $\mathbb{R}^{n+} \mapsto \mathbb{R}^{n+}$ and can be a combination of each $d_i(p)$.

$$\begin{aligned} \max_{p} \quad & (p-c)*q \\ \text{subject to} \quad & q = D(p) \\ & p(i) \geq 0 \quad i = 1,\ldots,n \end{aligned} \tag{3.1}$$

### 3.1.2 Learning Problem

In order to solve this decision problem we must answer how different choices of $p$ affect the quantity sold. In order to do this we must determine a model for how consumers decide to purchase a product. In retail the relationship between price and quantity sold is commonly called a demand function because often it is represented by a function. In reality this could be any relationship and include dynamics. Predicting this relationship is called demand forecasting.

The first step to modeling a demand function is to select a model class for demand. This model class can be as simple as a linear static function or include dynamic systems with memory of past prices. Modeling demand could also contain parts (functions, or dynamic systems) which are affected by other things beside price such as, the day of week, time of year, etc. These things can be taken in consideration in the selection of a model class.

Once the model class is chosen, historical data can be used to determine the parameters of a specific model in the class which best explains the historical data. These parameters will be coefficients to the demand function or the dynamic equations. In order for parameter estimation to be done well, it is important to use data which captures the important features you need. For example, to learn how demand changes as price changes, sales data is needed for different prices. Similarly, data only from summer months cannot be used to learn how demand changes based on the time of the year. The data must be informative over the desired range of prediction.

In the simplest case quantity can be unaffected by the price. We may choose a model that the quantity sold on a particular day will be the average of the daily sales over the past two weeks no matter how the price has changed. This is commonly known as a moving average demand forecasting model.

Demand could instead be a static function of the price. In this case the price will affect what the demand will be, and that affect will be determined by the function specified. This is the framework looked at by many economists and retail managers. This framework has a rich theory built around it, with standardly accepted functions and ways to come up with those functions.

Finally, demand could be a dynamic system with memory using price as an input. Not only does the price affect what the demand will be but also what the price has been in the past. This method for modeling demand is a current area of research and is appealing because it can capture a difference between increasing a products price to $x$, and decreasing the price to $x$. It also can explain consumer behavioral effects, such as waiting for things to go on sale because they have been seen on sale before. The business intelligence platform design can fully support each of these modeling paradigms.

### 3.1.3 Model Reduction Problem

Regardless of which model class is chosen for the demand function, it may be necessary to reduce the model class in order to make the computation feasible. For example, there may be so many different products for sale that it may be unreasonable to allow the demand of one product to be affected by all of the other products, because the number of parameters may be too many to esimate.

**Example 7** (Too Many Parameters in Demand Forecasting) *Suppose a store sells 10 different products, and the manager is trying to model how the sales of each product depend on the prices of all of the different products. Using a linear demand function there will be 10 parameters for how each product affects the sales of product 1. For product 2 there are 10 more parameters, and so on. There are 100 parameters in all which must be learned requiring a lot of data. Now imagine a more realistic scenario of a store with thousands of products. This would be millions of parameters.*

From this example we can see that even for simple models the number of parameters may be too large to estimate. Even if the computation was tractable the data that would be needed to estimate all those parameters would not be readily available. In situations such as this, algorithms can be developed which can select subsets of related products. One could use the data to determine which products were bought together often, or never at all, to determine which products complement each other or are substitutes for each other. These may be good candidates for deciding which product's price should affect another product's price. This could eliminate several variables and limit the number of parameters to be learned to a few parameters per product.

### 3.1.4 Verification Problem

The verification problem is to verify the algorithmic decision process created in the first three steps. To verify the solution to the decision problem of this business intelligence problem we need to verify whether the prices selected actually do maximize profit.

We can also formulate a verification problem on the model chosen in the learning problem. In this case the learning problem resulted in a demand function (or system) which relates the price selected with the quantity sold. The verification problem with respect to demand forecasting is to verify whether there is evidence in the data to suggest that the forecasts from the algorithms are incorrect. The standard way that this is done is to test the algorithm on some novel data set that was not used in learning algorithm parameters.

One of the main benefits of the business intelligence platform is that students have a mechanism for verifying whether the data suggests that their algorithm predicts well or not. By adding their algorithm to the platform they will be able to compare its predictions with the actual sales data to determine how well their algorithm performs. They will also be able to compare the performance of several algorithms to determine their relative performance.

## 3.2 Platform Details

The business intelligence platform is a platform which stores and displays BYU Bookstore sales data, and provides a mechanism for students to develop and study demand forecasting algorithms. Using this platform students are able to write algorithms that predict future sales of products and see how well their predictions work on real data. There are four major pieces which make up the platform. Figure 3.1 shows how these four pieces work together.

### 3.2.1 Sales Database

With our collaboration with the BYU Bookstore we have had access to their sales data for the past four years. This includes several hundred thousand products, and thousands of transactions per day. At regular intervals the Bookstore sends data files containing the data for every item that is scanned for sale, including price, item number, quantity sold, time, register, etc. Currently this occurs approximately once a week.

The data being received from the bookstore is parsed by automated scripts and is stored in a MYSQL database on our servers. Several preliminary calculations are made on this data including how many of each product were sold in a day, week, or month. Each of these preliminary calculations is stored in a separate table to speed data access. This data is then made available to be viewed on the platform or used by students to design demand forecasting algorithms.

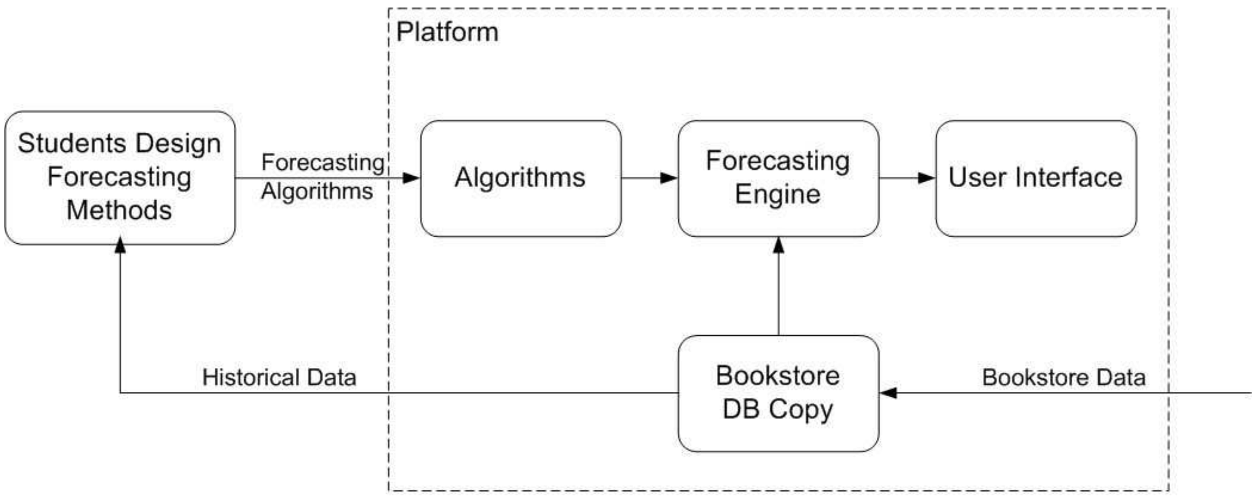


Figure 3.1: Architecture of the platform with inputs from the BYU bookstore in form of sales data and students inputting learning algorithms.

### 3.2.2 Algorithms

The platform consists of several common demand forecasting algorithms similar to those that can be found in any introductory course in demand forecasting. They include the moving average, exponential moving average, and Holt-Winter's algorithms. Students can learn about these demand forecasting algorithms and see what makes them effective or ineffective by comparing their performance on any of the bookstore products. After learning what has already been done, they can then design their own algorithms.

The algorithms that the students write must be written in Matlab and follow a standard Matlab API shown below in figure 3.2.

```
result = algorithmName(data, prices, processTime, forecastHorizon,
                       parameters)

data: a time series array of sales data for a product where each
 element of the array is the number of items sold for that
 time period.

prices: an array the same length as data with each element
 representing the price of the item during that time period.

processTime: the amount of data that the algorithm has in order to
 produce its first forecast.

forecastHorizon: how many periods of forecast ahead of the last
 data point are desired by the caller.

parameters: any other parameters needed by the  algorithm.
 (examples include weighting factors, etc. that cause the
 algorithm to return different results)

result: a time series array of length
 data-processTime+forecastHorizon where each element, $i$,
 contains the forecast for the data point in data[i+processTime].
 The algorithm should only use data from before that data point.
```

Figure 3.2: This is the API for a user defined demand forecasting function. Following this API ensures that the platform can call the function, passing the needed information, and get the results in a uniform manner.

Once a new algorithm is designed, students can then upload it into an algorithm database. The database keeps track of algorithm properties such as the minimum amount of data required to produce a forecast, the display name, the Matlab script name, optional parameters, and everything the platform needs to know in order to run the algorithm. Once an algorithm is loaded into the database it automatically becomes part of the platform. The uploaded algorithm is added to the user interface as a possible algorithm to be selected and is able to be run by the platform on bookstore data if selected.

### 3.2.3 Forecasting Engine

The forecasting engine executes demand forecasting algorithms at run time after the user decides which algorithms to run. This can take some time when multiple algorithms are selected, which can be a problem, but takes just a few seconds when only a couple of algorithms are chosen. The reasoning for executing the algorithms at run time is because of the large amounts of time and space that would be needed to run each algorithm on each product and store the results. We do not expect excessive amounts of traffic on all products so it does not make sense to make and store all of the pre-calculations. Another benefit to running the algorithms at run time is that the performance of an algorithm can be seen minutes after data is uploaded instead of the next day.

One thing that could affect the performance of the platform is if future algorithms are designed which take a long time to run. For these algorithms the run-time execution might take so much time that waiting is not desirable. The platform can be updated later to incorporate some pre-proccesing for certain algorithms, and/or products. These possibilities need to be analyzed based on the usage of the platform which will be better known at a later time.

### 3.2.4 User Interface

The graphical user interface (GUI) consists of a website written in PHP called the retail laboratory, www.idealabs.byu.edu/store/retailLab.php. It allows a student to select any product from the bookstore, and see daily sales of that product for the most recent month of data. Figure 3.3 shows the platform after a student has selected a product via the menu on the right. The graph on the top shows how many sales there were each day and the graph on the bottom shows what the price was for each corresponding day.

On the right side of the GUI is a menu of all of the algorithms existing in the platform. These algorithms are taken from the algorithms database for display in this menu. The user may

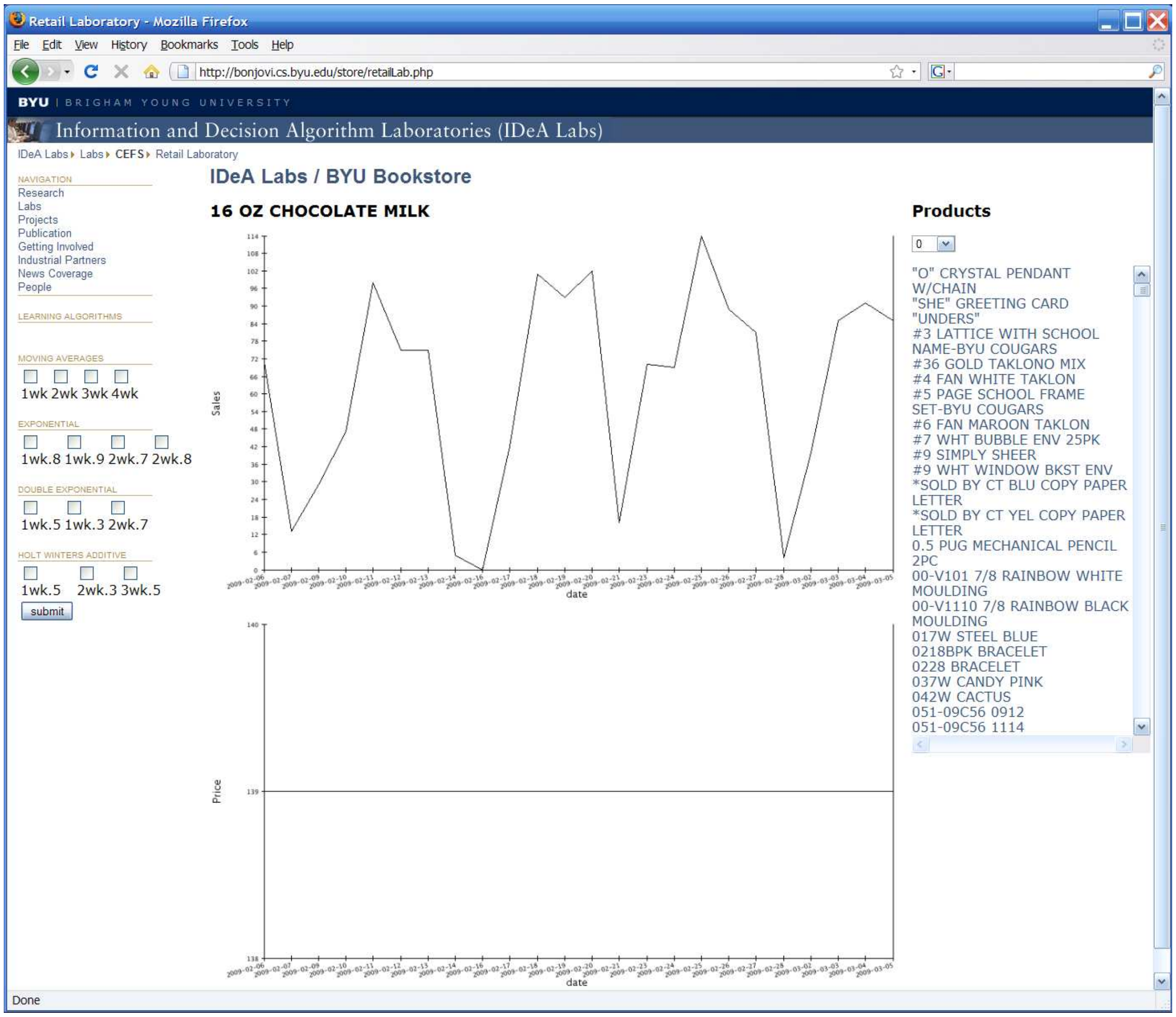
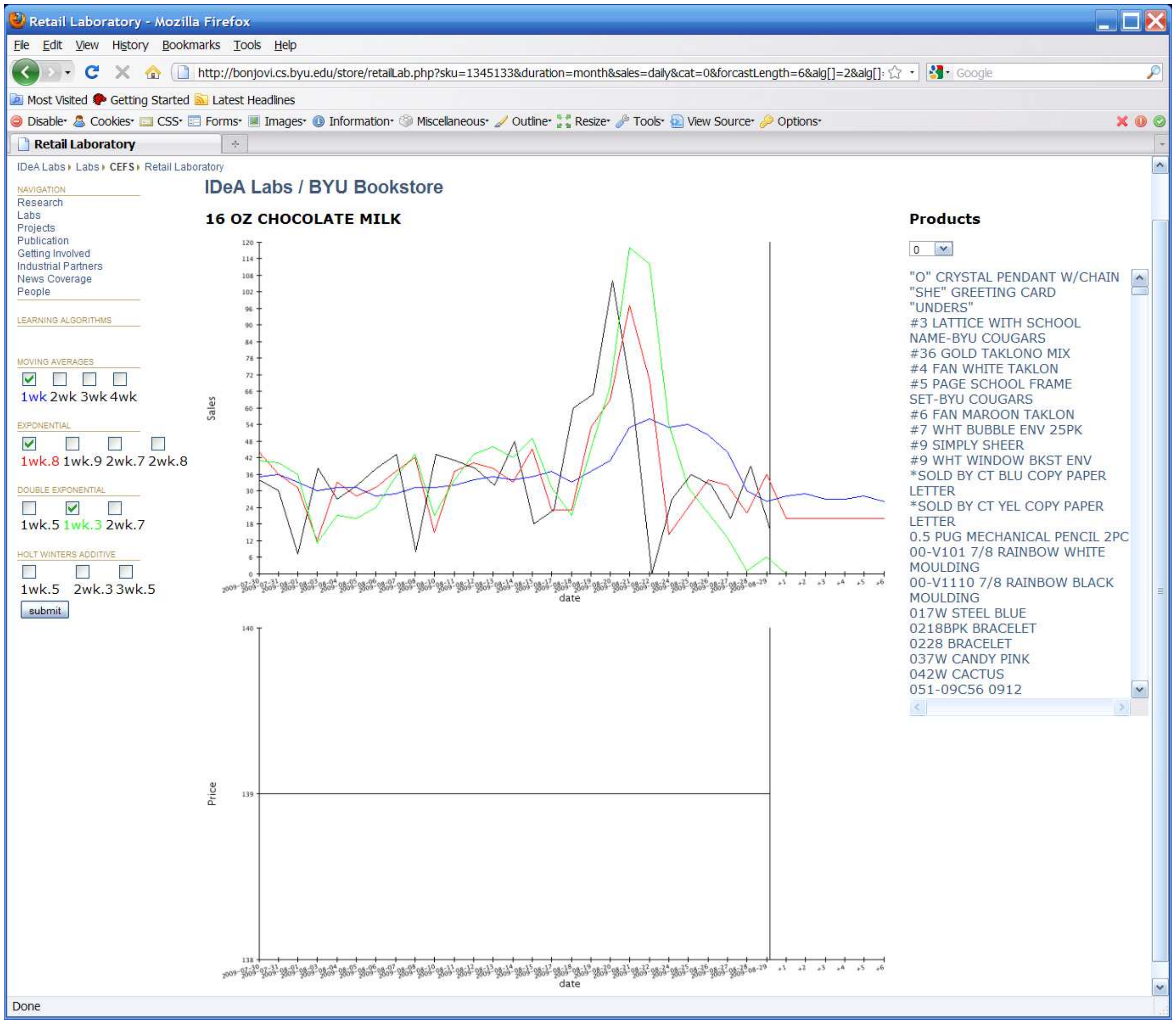


Figure 3.3: The user interface of the platform after a product has been selected. A product may be selected on the products menu to the right. First the user selects a department number, then a desired product from the list.

select any number of demand forecasting algorithms to be run on the current selected product and click a submit button. This causes the algorithms to be run on the appropriate data and a graph to be displayed for each algorithm. The algorithm graph shows for each day what the predicted number of sales would have been using that algorithm on only data from before that day. Figure 3.4 shows the platform after a student has selected several algorithms. Students can use this platform to compare different algorithms, including their own, and see what algorithms perform better.



http://bonjovi.cs.byu.edu/store/retailLab.php?sku=1345133&duration=month&sales=daily&cat=0&forcastLength=6&alg[]=2&alg[]:

Moving averages: [✓ 1wk] 2wk 3wk 4wk
Exponential: [✓ 1wk.8] 1wk.9 2wk.7 2wk.8
Double exponential: 1wk.5 [✓ 1wk.3] 2wk.7
Holt Winters additive: 1wk.5 2wk.3 3wk.5

"O" CRYSTAL PENDANT W/CHAIN
"SHE" GREETING CARD
"UNDERS"
#3 LATTICE WITH SCHOOL NAME-BYU COUGARS
#36 GOLD TAKLONO MIX
#4 FAN WHITE TAKLON
#5 PAGE SCHOOL FRAME SET-BYU COUGARS
#6 FAN MAROON TAKLON
#7 WHT BUBBLE ENV 25PK
#9 SIMPLY SHEER
#9 WHT WINDOW BKST ENV
*SOLD BY CT BLU COPY PAPER LETTER
*SOLD BY CT YEL COPY PAPER LETTER
0.5 PUG MECHANICAL PENCIL 2PC
00-V101 7/8 RAINBOW WHITE MOULDING
00-V1110 7/8 RAINBOW BLACK MOULDING
017W STEEL BLUE
0218BPK BRACELET
0228 BRACELET
037W CANDY PINK
042W CACTUS
051-09C56 0912

Figure 3.4: The user interface of the platform after several learning algorithms have been selected and run. Algorithms are selected from the Learning Algorithms menu on the left, by checking the appropriate boxes and clicking the submit button.

### 3.2.5 Based on a Real System

One of the reasons why this platform is appealing to use to teach decision automation to students is its tie to a real system. Rather than working with sample or made up data, students get to look at and work with actual BYU bookstore sales data. They get to see first hand many of the real world complexities that face retail managers. At the same time the platform gives them a virtual “sandbox” in which to play, where they can be creative and learn about common demand forecasting algorithms as well as design new ones.

Some of the interesting features found in the bookstore data are that there are large seasonality trends that are tied to the school calendar. Many items such as school supplies have higher sales at the beginning of the semester. Holiday seasonality is also seen in the apparel and gift categories. Many of the snack items experience a weekly cycle as well. These patterns can be analyzed and planned for as students design their algorithms.

As students come up with good forecasting algorithms and strategies, the BYU bookstore has also given the research group permission to run experiments and change prices on products in order to try out their ideas. This makes the business intelligence platform more than just a virtual test area. The physical bookstore can become part of the platform for teaching decision automation. Several promotions and experiments have been run so far, and the potential to run some more adds to the excitement of using this platform and learning decision automation.

## 3.3 Future Platform Development

As part of this thesis we have created the business intelligence platform. The business intelligence platform has been developed as a prototype to introduce students to business intelligence. It has been designed with revision and change in mind. As a prototype, there are a few extra features that are being planned for the future. One extra feature is to be able to view more than just one months sales data. All of the data retrieval, and graph drawing functions have been designed generally enough to allow retrieval and drawing of any number of previous days. There is an HTML get variable that is set for the amount of history to display. By changing this value more history can be viewed. What remains to be done is to add a control to the interface to allow a user to select how much data he or she wants to view.

Many of the products sold at the bookstore have low quantities of sales per day. For example paintings, art supplies, expensive clothing items, nativities, etc. may all have many days where there are no sales or just one sale. In these cases, as well as those with more sales, it may be desirable to view aggregated sales such as weekly sales of an item, or monthly sales of an item. These aggregates are already being calculated and stored inside the sales database, and there are functions which are designed to access these aggregate sales data if certain HTML parameters are set. Controls still need to be developed to allow the user to specify any aggregates he or she wants to view.

As of now the Matlab API that must be followed for user defined algorithms allows for optional algorithm specific parameters to be entered. This was defined to allow for different weighting parameters, seasonality parameters, or other parameters that apply to a specific demand forecasting algorithm. Currently there is no way designed for a user to enter these values manually. Instead they must be hard coded into the database. To run a specific algorithm with different parameters another algorithm must be created with different hard coded values. For example, for one forecasting algorithm, moving average, which takes a parameter for how long of a window to use, there will be four different algorithm check boxes for 1 week moving average, 2 week moving average, etc. For now this is not really a problem, however as more user algorithms are added to the system this can become very cluttered, and a way for user entered parameters may be a better solution.

A final feature that is to be added has to do with the way forecasts are generated. Currently the assumption that is made for future sales of a product is assuming the price is held constant to the most recent price into the future. An improvement would be to allow the user to be able to set the price for future predictions. That way the platform could be used to answer questions such as: “What would happen to future sales if the price is raised to X dollars.” As with many of the other features a control will need to be added to the user interface to allow the user to set a future price.

## 3.4 Conclusion

We have created a platform in business intelligence that focuses on teaching students to use the learning problem to select models for product demand. This platform gives students access to actual sales data from the BYU Bookstore in order to design and test demand forecasting algorithms. Although this is a very different field than portfolio management, we have shown that the same decision architecture used to solve problems in portfolio optimization can be used to solve business intelligence problems. The business intelligence platform is enhanced by the collaboration with the BYU Bookstore tying this learning platform to a real world problem with potential for students' solutions to be tried in the real world store.

# Chapter 4

# Automated Water Management

Water conservation in the western United States is very important. The area is primarily a desert, with little rainfall. In order to sustain large city populations as well as agriculture, reservoirs have been built to capture rainwater and snow runoff in the spring for use throughout the summer. Due to environmental regulations reservoir building has declined sharply while demands for water have continued to increase. Without new reservoirs being built new ways must be found to conserve the water currently being stored.

Automated water management is beginning to be used as an important water conservation technique. By controlling reservoir release more precisely, only water that is needed will be released from the reservoir, conserving the extra water that would normally go to waste downstream. Deciding how much water to release is made difficult because of large and variable delays in the river, uncontrolled inflows, and various environmental factors, such as evaporation, seepage, or rainfall.

The problem is how to decide how much water to release from the reservoir to be used downstream while accounting for external, uncontrollable factors and to minimize any excess release. The water management platform is designed to solve this problem and facilitate a study in controller design for future solutions to this problem. Just as in the other platforms, system identification, model reduction, and verification also must be considered in order to provide a good solution. This chapter focuses first on describing the system to be controlled, then goes on to describe the process of modeling and controller design to solve this problem. Lastly we show the implementation of the platform and show how it is designed to allow for many other controller designs. This chapter comprises a collaborative effort with the Bureau of Reclamation in managing the Piute Dam on the Sevier River.

## 4.1 Sevier River

The Sevier River, in central Utah, serves primarily as irrigation water with a small amount used for municipal water in the local towns. The Sevier River Basin is completely enclosed so that any water flowing down the river empties into the desert. The Piute Dam and reservoir are located on the upper portion of this river. Water released from the reservoir may be diverted into one of several canals. Any water not diverted into the canals continues down the river and is lost for any other use. This makes the Sevier River a great location to practice water conservation using automated control. Water can be conserved by designing a decision algorithm which releases just enough water from the reservoir so that every canal gets the water it needs, but no more.

Figure 4.1 shows the stretch of river below the Piute reservoir. The release from the dam determines how much water enters the system. There is also an uncontrolled, but measured inflow at Clear Creek, and eight diversion canals that take water out of the river. At the end of the river is the small Vermillion Dam. The goal is to have no water flowing past Vermillion Dam as it will be lost to our water users.

Each black circle in Figure 4.1 shows where there is station measuring flow. Each of the stations on an out-flowing canal also actuates a gate which regulates the flow to match a flow value set by the canal owner. There are also four measuring stations along the river, the reservoir release (not shown), just above Clear Creek, near Elsinore, and at Vermillion Dam. Flow data is collected at all of these stations every hour and stored into a central database. This remote monitoring and telemetry has been installed in the system since the summer of 2000.

## 4.2 Other Work in Canal/River Automation

The majority of the previous work in canal/river automation has been focused on obtaining accurate measurements of the canal/river and manually controlling gates remotely. Few rivers or canal

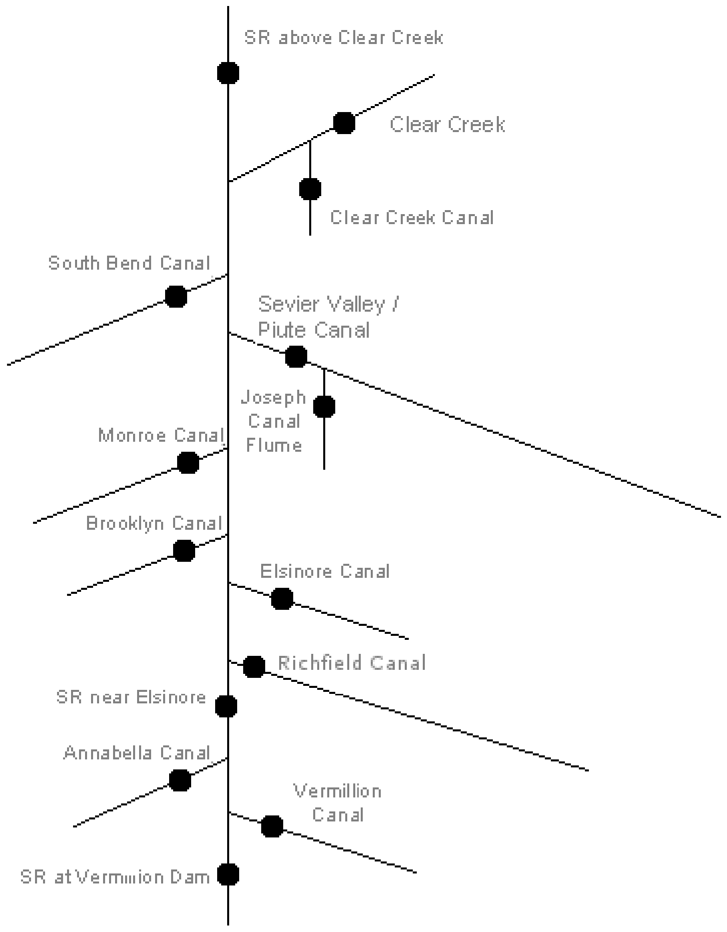


Figure 4.1: A stick figure representation of the central stretch of the Sevier river. The water flows from top to bottom. The water entering the system is determined by water released from the dam and the measured but uncontrolled inflow at Clear Creek. All outflows are measured and controlled. A small metal gate called Vermillion Dam represents the end of the system and any water flowing over the dam is lost downstream.

systems in the world have a computer determining how to control the gates releasing water from various reservoirs. This is mostly due to the fact that water is such an important resource, and many people are skeptical about having their livelihood controlled by a computer. There are however, to the best of our knowledge, only two groups that have had success in controlling rivers and canals: one in southern France, and one in Australia.

The group in southern France (Litrico, Malaterre, et. al.) has designed various algorithms, including robust control and proportional-integral (PI) algorithms, to control gates on different canal systems [17], [16]. They have also done some work with canal modeling using St. Venant equations for water flow [20]. The group in Australia, led by Erik Weyer, has modeled canals using a mass balance model and designed several different controllers including PI, and linear quadratic Gaussian control (LQG) to control canal gates [28], [29].

None of these approaches work very well with our system. First, the work has been done on canals that have a very small grade and accurate flow measurements. The flow measurements on our river are not very accurate and may be off by as much as 10%. Also our river has steep grade which causes the water to flow more quickly than the canals which changes the system dynamics. Second, the delay in our stretch of river is over a day and we receive data only once an hour. Previous work has been on canals with much less delay and data collection every second or minute. Third we have no control of the water after it has left the reservoir except for the water being diverted into the canals. Previous work has relied on control structures along the canal/river to control the flow. Because of these differences we will be using different models than these groups.

## 4.3 Modeling the river

### 4.3.1 Select a Model Class

The first step in modeling the Sevier River is to select a model class that captures the dynamics of the river. In [22] Maxwell compares several different models of the Sevier River. He concludes that a parameterized mass balance model most correctly describes the river during summer months when there are high flows. We select a parameterized mass balance model as the basis of our model class, however we add to that model terms for a third order dynamical system. This allows the model to also capture some of the smoothing effects of water flow that are absent using only the parameterized mass balance model. The rest of this section will describe the development of a new model for the Sevier River using this new model class.

We model the Sevier River as a multiple input single output (MISO) system with the flow past Vermillion Dam being the system output. We treat the two river inflows and the eight outflows to the canals as system inputs because each represents an external influence on the water in the river. Since the outflows of the river are measured, their exact value is known and the coefficients to these system inputs will be fixed at negative one. The reason negative one is used is because water is being taken out of the river. Because the influence of the inflows on the output is uncertain, we let the coefficients to the inflows be variables to be determined from our learning algorithm from the data. This allows the model to account for evaporation, seepage, or other inflows or outflows that may occur along the river before the water reaches the bottom of the system. Equation (4.1) shows the mathematical model of the system.

$$y(k) = a_1 y(k-1) + \ldots + a_3 y(k-3) + b_1 u_1(k-d_1) + b_2 u_2(k-d_2) - \sum_{i=3}^{10} u_i(k-d_i) \tag{4.1}$$

Our model (4.1) states that the output at time $k$, $y(k)$, is determined by the previous three outputs as well as the positive and negative flows added by the system inputs. We define inputs in the order that they affect the system, $u_1$ as the reservoir release, $u_2$ as the inflow at Clear Creek, and $u_3, \ldots, u_{10}$ as the other canals. We also define $d_i, (i = 1, \ldots, 10)$ to be the delay of the river from input $i$ to the end of the river.

### 4.3.2 Determining the Delay

To determine the different delays in the system we look at flows from two points along the river. We find where there is a large shift in flow at the upstream point and find how long it takes for there to be a corresponding shift in the flow at the downstream point. The reasoning for this method is that a possible experiment to determine the delay would be to let a large amount of water out at one time and measure the time it takes to reach the next part. Using data from 2007 we found several places in the data where there were large fluctuations in the flow and averaged the delay time to find different delays in the system. A sample of this technique is shown in figure 4.2. To find the delay of the canals we use the same technique, however we look for places where the upstream river flow is constant, the canal has a large shift, and the downstream shift is in opposite to the canal flow see figure 4.3.

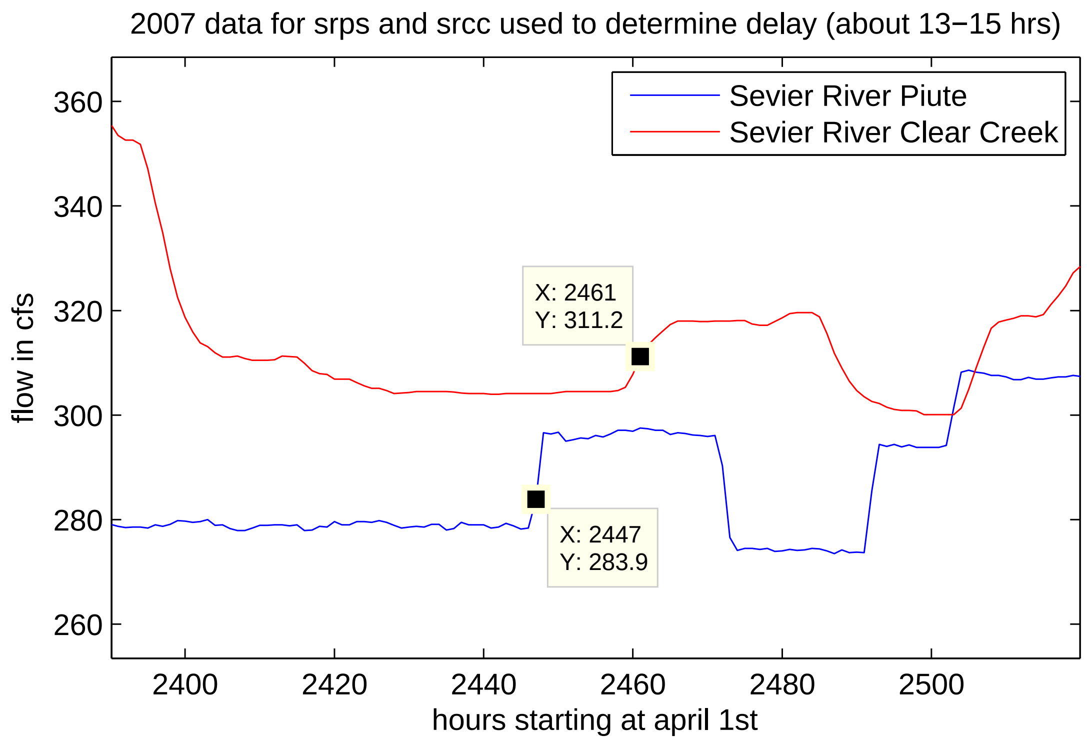


Figure 4.2: The delay for the stretch of river between the reservoir and Clear Creek is found by finding corresponding changes and measuring the delay.

It must be noted that the delays we find using this method are approximations or averages of the actual delay. As the amount of water flowing in the river increases the flow will be faster and

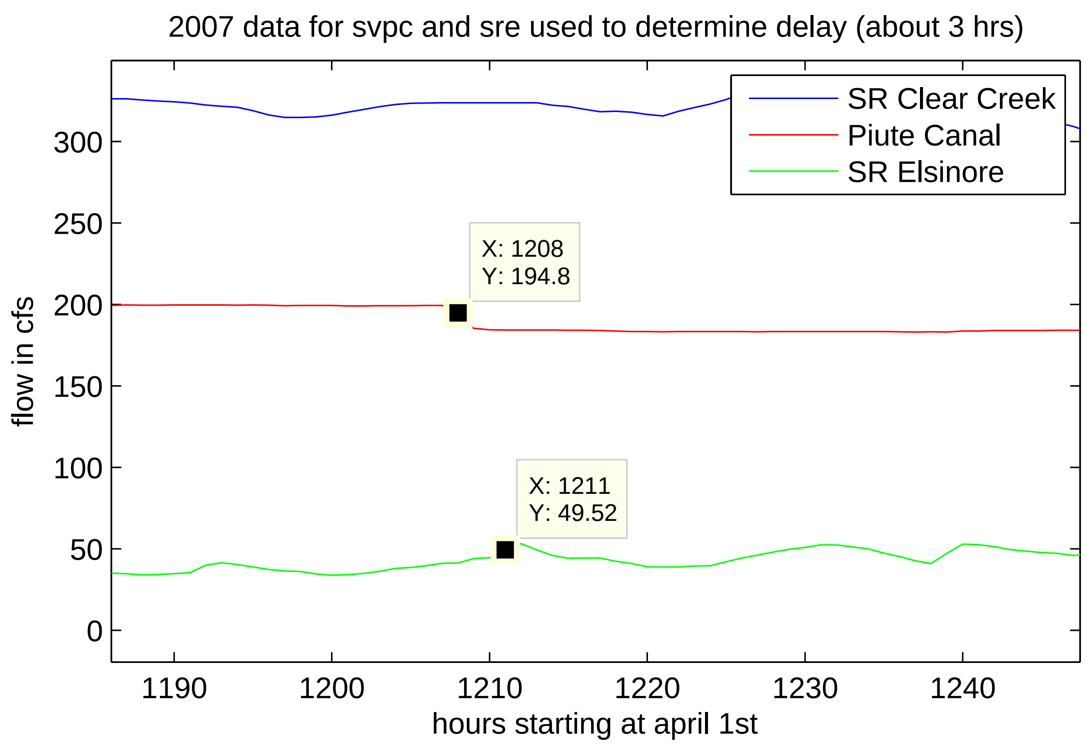


Figure 4.3: Finding the delay of a canal. The upstream river is constant, the canal reduces the flow it is taking out of the river and the downstream measurement reflects an increase in flow.

there will be less delay for the water to reach the bottom. The opposite is true if there is less water flowing. When we validate a control solution we will need to show that it is robust to differences in the delay of the river.

Using this technique we determine there is a delay of approximately 14 hours from the reservoir to the Clear Creek inflow. There is a delay of another 18 hours from Clear Creek to Vermillion Dam. This is a total of about 32 hours for the entire stretch of river. Table 4.1 shows the delays for each part of the system.

### 4.3.3 Selecting Data

Since water conservation is most important during the irrigation season we want our model to be effective in describing the river during spring and summer months. We will use river data during the months of April through September for six years (2000-2004,2006) to train our model, and we will use data from those same months for 2007 and 2008 to validate our model. We choose not to

Table 4.1: Symbols and delays associated with several stations along the Sevier River.

| Station Name | Symbol | Delay |
|---|---|---|
| Piute Reservoir | srps | 32 |
| Sevier River above Clear Creek | srcc | 18 |
| Clear Creek | ccd | 18 |
| South Bend Canal | sbch | 16 |
| Sevier Valley Piute Canal | svpc | 15 |
| Joseph Canal | jch | 15 |
| Monroe Canal | mch | 14 |
| Brooklyn Canal | bch | 13 |
| Elsinore Canal | ech | 12 |
| Richfield Canal | rch | 12 |
| Sevier River near Elsinore | sre | 12 |
| Anabella Canal | ach | 8 |
| Vermillion Canal | vch | 0 |
| Sevier River at Vermillion | srv | 0 |

use data from the year 2005 because that year was a flood year and the flows are unlike any other year. When that year is used for training it causes the model to predict too much water for the other years. It is possible to use 2005 data to learn a model which can be used in flood years, though we have not done that here.

### 4.3.4 Model Selection

To select a model from our specified model class we use an iterative maximum likelihood minimization method to select model parameters for our model. This method minimizes the error between the model's predicted output and the actual output from data. We start with an inital model and then adjust the model's parameters along a specific direction to decrease the error. For more information see Matlab's system identification toolbox documentation, specifically the commands pim and armax.

Using this method along with the specified data and model class, we find the following parameters for our model:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} -.7949 \\ -.4595 \\ -.1705 \\ .839 \\ 1.973 \end{bmatrix}. \tag{4.2}$$

These parameters seem reasonable as both of the inputs have positive coefficients. Also the larger coefficient from Clear Creek could be explained by it being an unregulated inflow. This inflow could represent all other unmeasured inflows along the river which would seem to be higher if the creek is higher and lower otherwise. The coefficient less than one from the reservoir could be explained by the effects of evaporation and seepage of the released water all down the river.

Using our model (4.1) with the parameters found in (4.2), we can use actual data from 2008 as the inputs, $u_1, \ldots, u_{10}$, and see what our model predicts the output to be. The closer the model output is to the actual river flow, the better our model represents the true river system. Figure 4.4 shows the model's predicted output compared with the actual output from 2008. Flows in the model output are negative in order to show where our model predicts a shortage of water in the system. This means that the last canal would not be getting as much as was ordered. This will be discussed further in a later section.

Because this model seems a bit too erratic we add a low pass filter to our model output to smooth out model flows at Vermillion dam. We select the filter parameter which minimizes the mean squared error of the model compared to the validation set. The filter parameter of .1 gives us a reduction in root mean squared error from 21.8 to 21.1 yielding the filter

$$\begin{aligned} x(k+1) &= .9048x(k) + .09516u(k) \\ y(k) &= x(k) \end{aligned}$$

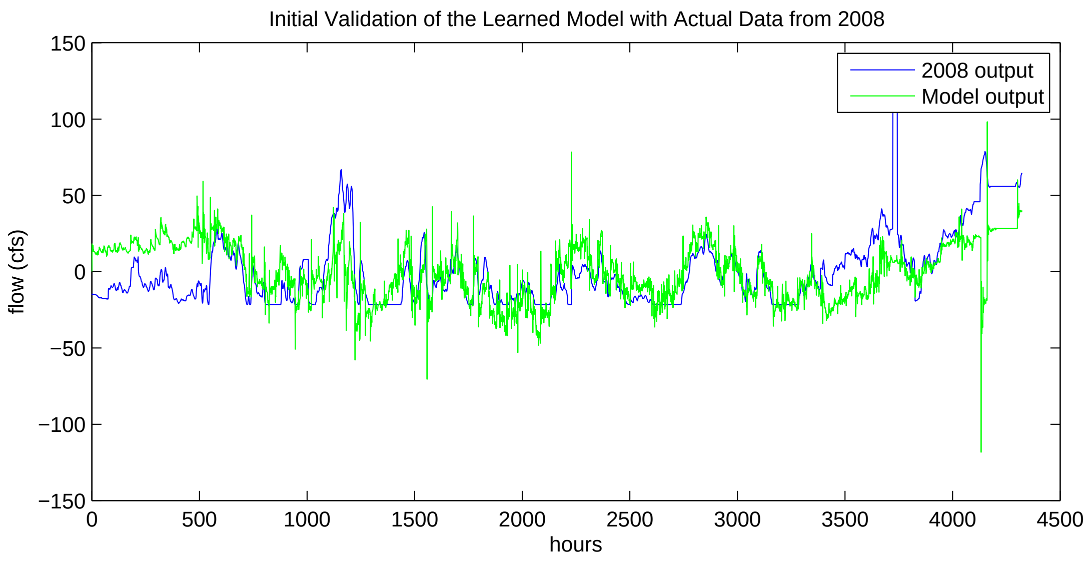


Figure 4.4: Estimated output of the original learned model (without low pass filter) with input from 2008 measurements compared to the actual river output in 2008. The low pass filter will be added to reduce the noisiness of the model.

and a final model of

$$\begin{aligned} x(k+1) &= Ax(k)+Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{4.3}$$

where

$$A = \begin{bmatrix} -0.7949 & -0.4595 & -0.1705 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0.0952 & 0 & 0 & 0.9048 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.839 & 1.973 & -1 & \cdots & -1 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

**Handling trends in the data**

In order for our learning algorithm to select model parameters with greater accuracy it was necessary for us to detrend the data by subtracting the yearly mean of each signal from itself. When running this model on-line during a watering season it is not possible to subtract the yearly mean, so we need some other way to detrend the data. We decide to use as a trend the weighted average between the current year to date flows and the average from all previous years except for 2005. The trend for a signal $s$ at time $k$ is

$$T(k) = \frac{\sum_{i=1}^{k-1} s_c(i) + D\mu_{all}}{D+k-1} \tag{4.4}$$

where $s_c(i)$ is the flow at time $i$ for the current year, $D = \max(3000-k, 0)$, and $\mu_{all}$ is the mean of the output for the previous years. After 3000 hours the trend becomes the current year's average, and before 3000 hours, previous years' average is included.

The reason we do not use the current year to date average for our trend is because water flow varies greatly during the season. During the beginning of the season the water flow is high and during the end of the year the flow is lower. Using the current average as the trend will cause the early part of the year to be shifted by larger trends than later in the year, thus adding a downward trend to the data which is undesirable.

### 4.3.5 Model Validation

Figure 4.5 shows the output predictions of our model for the validation set. The top graph is 2008 data and the bottom graph is 2007 data. Table 4.2 shows the absolute and root mean squared errors for both years and together.

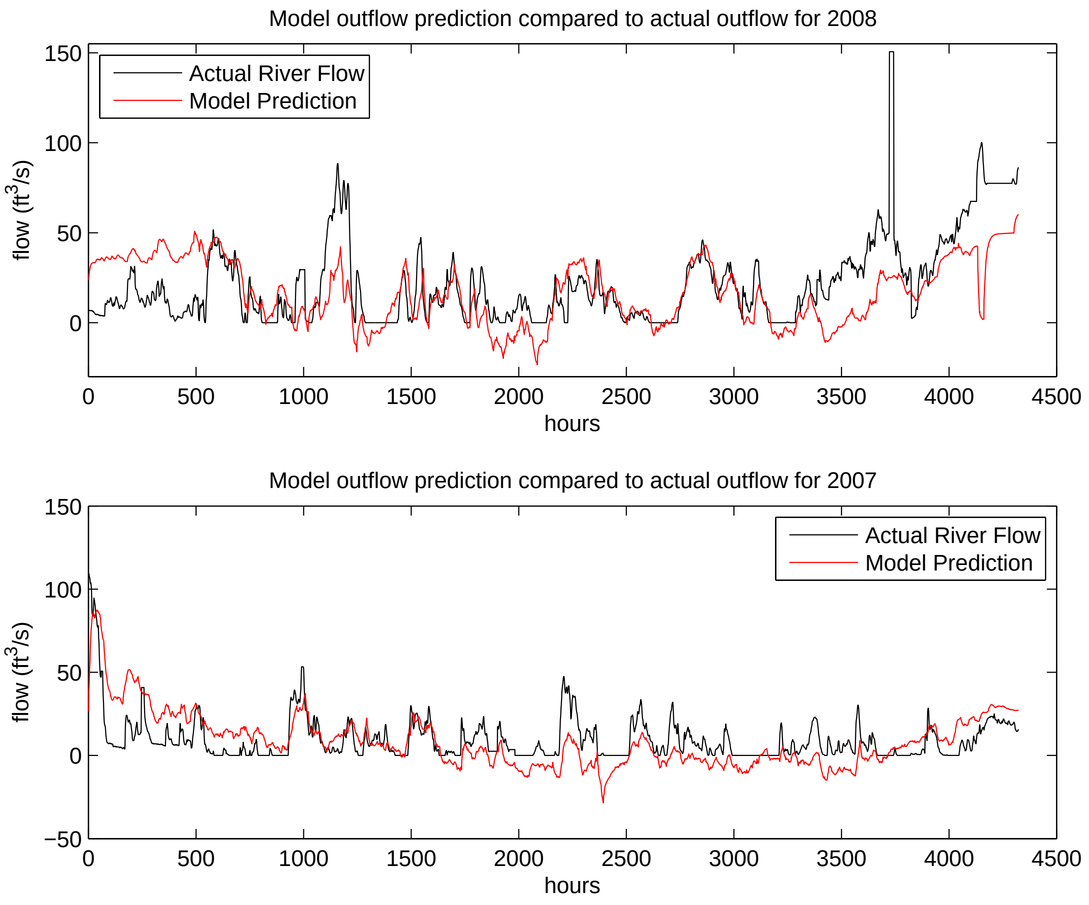


Figure 4.5: The model output compared with the actual output. Top is 2008 data bottom is 2007 data.

Table 4.2: Error of Model on Two Validation Sets.

| Year | Root Mean Squared Error | Absolute Error |
|---|---|---|
| 2008 | 21.67 | 15.30 |
| 2007 | 13.37 | 10.62 |
| Both | 17.80 | 12.96 |

The model definitely captures many of the trends and changes for both years, but also has periods where its predictions are off especially at the beginning and the end of the season. One limitation on performance is the reliability of the flow measuring devices. The measurements of flow are rated to be within 10% of actual flows. For the flows on this river that can easily be 5 to 10 cubic feet per second. Over several canals this can make a large variance. Also, when flows are at zero or close to zero the sensors can still read “false” flows due to sediment build up in the sensor.

One of the reasons for some of the error is that we allow the model to predict negative values. The river cannot have a negative flow, but a negative prediction is the same as predicting a shortage for the last canal. Thus a flow of -10 cfs means that the last canal has 10 less cfs than ordered. Because we do not keep track of how much water was ordered for each canal, it is difficult to determine how much shortage there was in the river when the flow was at zero. So, when the river is at zero and the model predicts some negative value, the negative prediction could be close to the actual state of the system.

There are a few possible explanations for the model’s poor performance at the beginning and end of the season. At the beginning of the season many of the canals are not being used and have a flow of zero but because of sediment the sensors may be recording one, this can cause unusually large errors. Also, some of the canals turn off before the end of the season which could contribute to large errors at that time as well. Typically more rainfall occurs at the beginning and end of the season, this would cause more water to be entering the system at those times. Another explanation for the error could be temperature which has a great impact on evaporation. During the beginning and end of the season the temperature is considerably lower than the middle of the season.

We feel that with the errors it is still a good model for peak season, especially given its relative simplicity. Even when there is some error the model still captures the trends and seems to be off by a constant factor. This leads us to believe that we will be able to control this river and that feedback control can improve system performance.

## 4.4 Controller Design

The objective of the controller for the Sevier River is to select the reservoir release, $u_1$, in order to cause the output to match some desired reference signal. On the Sevier River the goal is to minimize water waste while meeting all of the canal demands. Thus, the reference signal for the output of the system will generally be set at some small amount or zero, and stay constant the majority of the time. The controller will then have to decide how much water to let out so that water is delivered to the canal but the outflow at the end remains low.

### 4.4.1 Control Architecture

For our controller design we are going to use the two part control architecture shown in figure 4.6. The first part of the controller is a feed forward controller (F in the figure). It uses any a priori information that we have available in order to determine a close estimate of what the control input should be. For the Sevier River this information includes the reference signal as well as future water orders for the canals. The second part of the controller is the feed back controller (K in the figure). This portion of the controller uses error from the system output in order to ‘fine tune’ the estimates from the feed forward controller. The plant (P in the figure) is the system being controlled. In our case this is the river system.

This control architecture is very versatile. It divides the control into two very manageable pieces. One piece controls for the system the best we understand it, and the other piece controls for the things that were not designed for by the first piece. This architecture also lets us take advantage of the data we receive from the water orders for each canal and use that information in determining future water release.

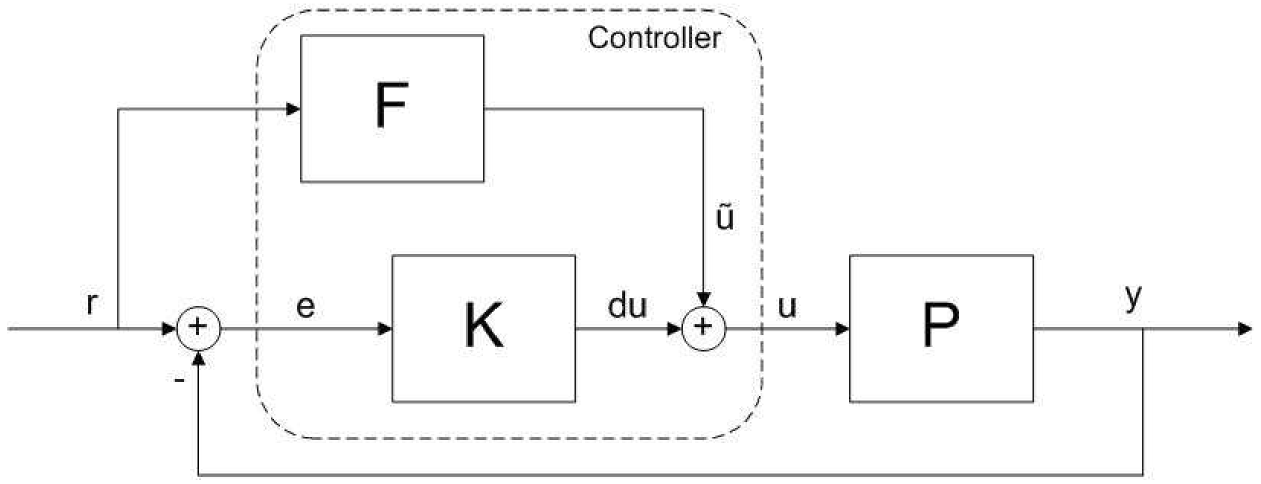


Figure 4.6: The control algorithm is split up into two pieces. The feed forward controller, $F$, uses all available information to get an approximate control choice, $\hat{u}$. The feedback controller, $K$, uses the error, $e$, to adjust the approximation by $du$.

This architecture also can be very useful when designing controllers for non-linear systems. Often when trying to analyze non-linear systems we make the assumption that the non-linear system behaves like a linear system in a small neighborhood, similar to the idea that if you get close enough to an edge of a circle it appears to be a line. This idea is further justified for analytic functions by looking at their Taylor Series and noting that “close enough” to the evaluation point, the linear terms in the series dominate all the others. By designing the feed forward controller to get close enough in controlling the non-linearities of the plant, a linear feedback controller can be sufficient to achieve acceptable performance.

### 4.4.2 Performance Criteria

We expect the reference command to change in a step like manner since the desired flows for the end of the river stay constant until a new flow is set. So the controller will be required to asymptotically track step changes in output. Due to the large delays in the system it will be difficult to have fast tracking, but we want to make sure that we eventually get to the right value. Canals also change in a step-like manner so we will require asymptotic tracking of step changes in the canals.

In addition to being able to follow changes in the river or canals we also want the controller to be robust to situations when the river behaves differently than our model predicts. While we expect our model to be descriptive of the behavior of the river, we know it can never be 100 percent correct. We want our control performance to be robust to incorrect parameters in our model. The performance must also be robust to differences between the delay we use in our model and the actual delay of the river.

Finally, we desire our controller to reject disturbances. This can include noise in the measurements, external disturbances such as rain or high temperatures, and sudden changes in canal flow without making appropriate orders first. In light of these disturbances the controller should maintain or return to reference flow.

### 4.4.3 Feed Forward Control Design

As stated before, our goal in designing the feed forward controller is to get the output to track the reference as closely as possible. In order to asymptotically track step changes in the output, the final value of the output needs to be equal to the reference command. We will choose $F$ to be the inverse of the final value of the plant. This suggests that we set $F = P(1)^{-1}$, where $P(1)$ is the discrete time transfer function of the plant evaluated at $z = 1$, which gives the final value of the plant. Now we show that this value for $F$ gives us asymptotic tracking.

With only the feed forward controller the open loop transfer function from $r$ to $y$ is

$$y = P(z)F(z)r \tag{4.5}$$

The final value theorem states that the final value of a time function $x$ is the same as the limit of its laplace transform evaluated at 1, $\lim_{t\to\infty} x(t) = \lim_{z\to 1}(z-1)X(z)$.

Applying this theorem yields

$$y = P(z)F(z)r \tag{4.6}$$

$$\lim_{t\to\infty} y(t) = \lim_{z\to 1}(z-1)P(1) * P(1)^{-1}r \tag{4.7}$$

$$\lim_{t\to\infty} y(t) = r \tag{4.8}$$

Thus for this selection of $F$ we have a controller which is both stable and meets our performance criterion of asymptotic tracking.

Since the model we developed in the previous section is the best idea we have for the plant, we use that model to determine $F = P(1)^{-1}$. The transfer function for our model (see (4.1)) is

$$P = \frac{.0952z^2}{(z-.9048)(z+.533)(z^2+.2619z+.3199)}. \tag{4.9}$$

Applying the final value theorem to the model yields a final value of .412. We then let $F = \frac{1}{.412} = 2.426$. Figure 4.7 shows the open loop system response to a step in the reference command using this $F$.

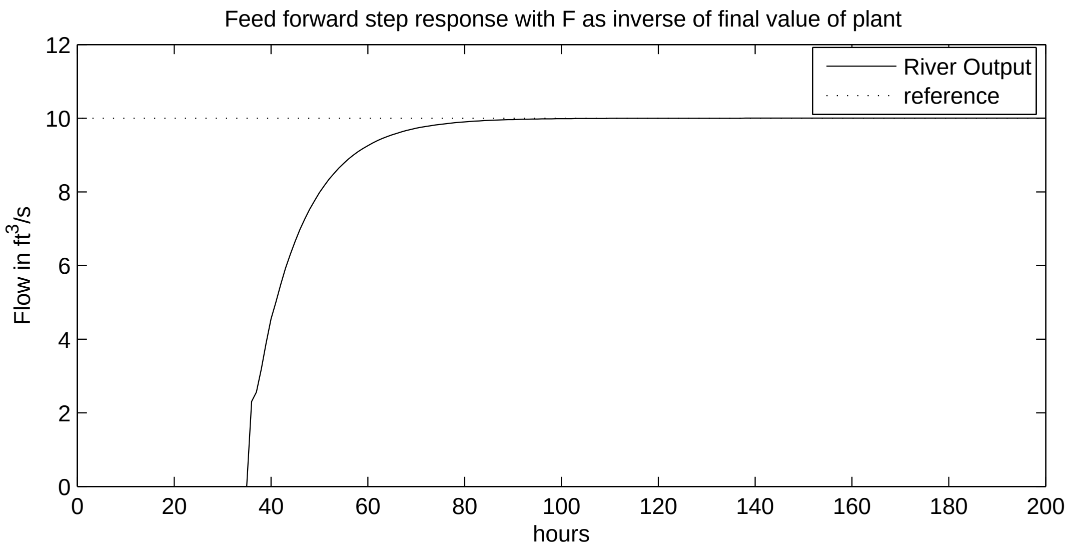


Figure 4.7: Open loop system with the feed forward controller. The plant is the model developed in the previous section and the feed forward controller has gain equal to 2.426. The system is stable and tracks steps in the reference command.

The feed forward controller also receives the water orders from the different canals. It uses these orders to predict what will happen to the canal inputs to the plant. From (4.3) notice that the inputs are each multiplied by a gain and added to the system. This controller takes each canal order and multiplies it by the inverse of the gain from the model and subtracts it from the control input. This has the effect of changing the reservoir release early, by the same amount that the input will affect the flow in the river later on. We will show how this works in section 4.5, controller validation.

As figure 4.7 shows, the feed forward controller performs very well when the plant is the same as the model we used in designing the controller. Unfortunately, but as expected, the feed forward controller is sensitive to errors between our model and the actual river. Figure 4.8 shows a possible response of our system for a case where the river is different from our model. When the river behaves differently from our model the final value of the system does not match the reference command. The system still maintains stability, however it no longer has asymptotic tracking. This motivates the need for the feedback controller.

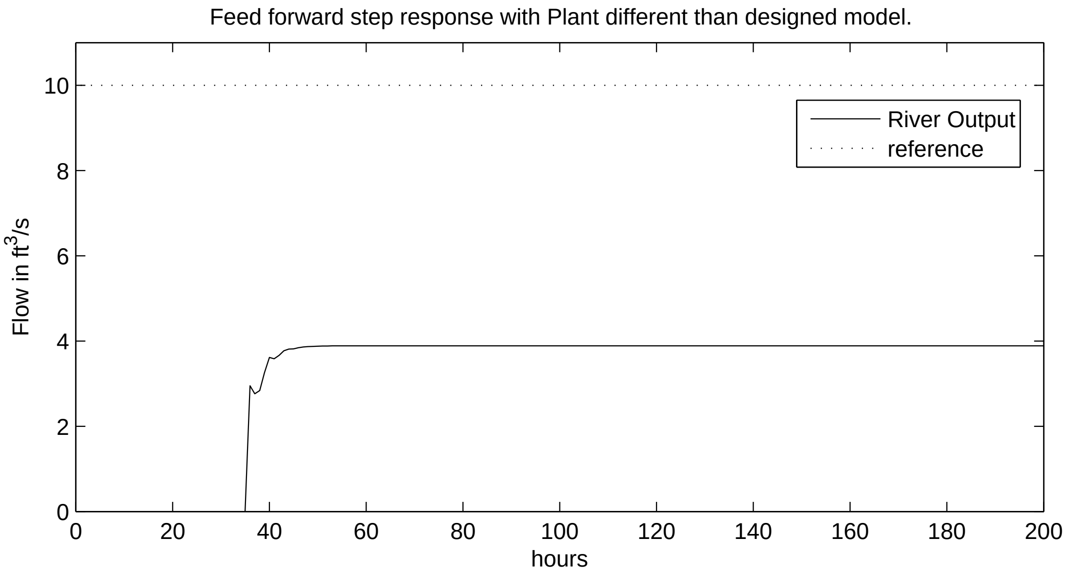


Figure 4.8: An example of the possible error in the open loop system when the river is different from the model we used to determine our feed forward controller. This shows the need for a feedback controller.

### 4.4.4 Feedback Controller Design

One of the important parts of our platform is the fact that many different controller designs could work for the feedback controller. Because our performance requirements require us to asymptotically track a step input we need a controller with an integrator. In this section we will design a proportional integral (PI) feedback controller, however other optimal controllers could also be designed including, LQG, $H_\infty$, $L_1$, etc.

The PI controller we have designed will be a controller of the form

$$K = k_p + \frac{k_i}{z-1} \tag{4.10}$$

where $k_p$ is the proportional constant or proportional gain, $k_i$ is the integral constant or the integral gain, and $z$ is the discrete laplace variable. These two gains must be carefully selected in order to achieve the desired control performance. If the gains are too high our controller may become unstable. If they are too low, the controller will not achieve the desired results.

We follow a standard process for selecting these two gains. First, we hold the integrator gain constant and adjust the proportional gain and check the system closed loop response. This will determine in large part how fast our controller response will be. The higher the gain is the faster the response will be, but also the more overshoot there will be. We continue adjusting the proportional gain until we achieve the most desirable response. After selecting a decent proportional gain, we keep it constant and adjust the integrator gain. This gain primarily affects how fast the system reaches steady state. However, as the gain increases too much, the system will also experience more overshoot.

While adjusting these gains we also have to pay attention to the gain and phase margins to determine how close the system is to loosing stability. Small margins signify that the system is stable but if there is very much uncertainty, the system with the true plant may not be stable. The gain margin is a measure of how much the system gain can increase before the system becomes unstable. The phase margin is a measure of how much shift in phase a system can take before it becomes unstable. The phase margin is also related to systems with delay. The more delay there is in a system the more the phase will shift, and so the more phase margin is needed. Since our system has large delays, we need to design our gains such that our phase margin is sufficiently large. We can increase the phase margin (and gain margin) by decreasing the gains.

The result of our gain tuning process results in $k_p = 1$ and $k_i = .05$. The bode plot in figure 4.9 shows that we have a gain margin of 2.69 and a phase margin of 139 degrees. These margins are sufficient for the closed loop system to be stable even with the large delays in our system. Figure 4.10 shows the step response of the closed loop system containing the plant with our feedback and feed forward controllers where the plant is as we have modeled it.

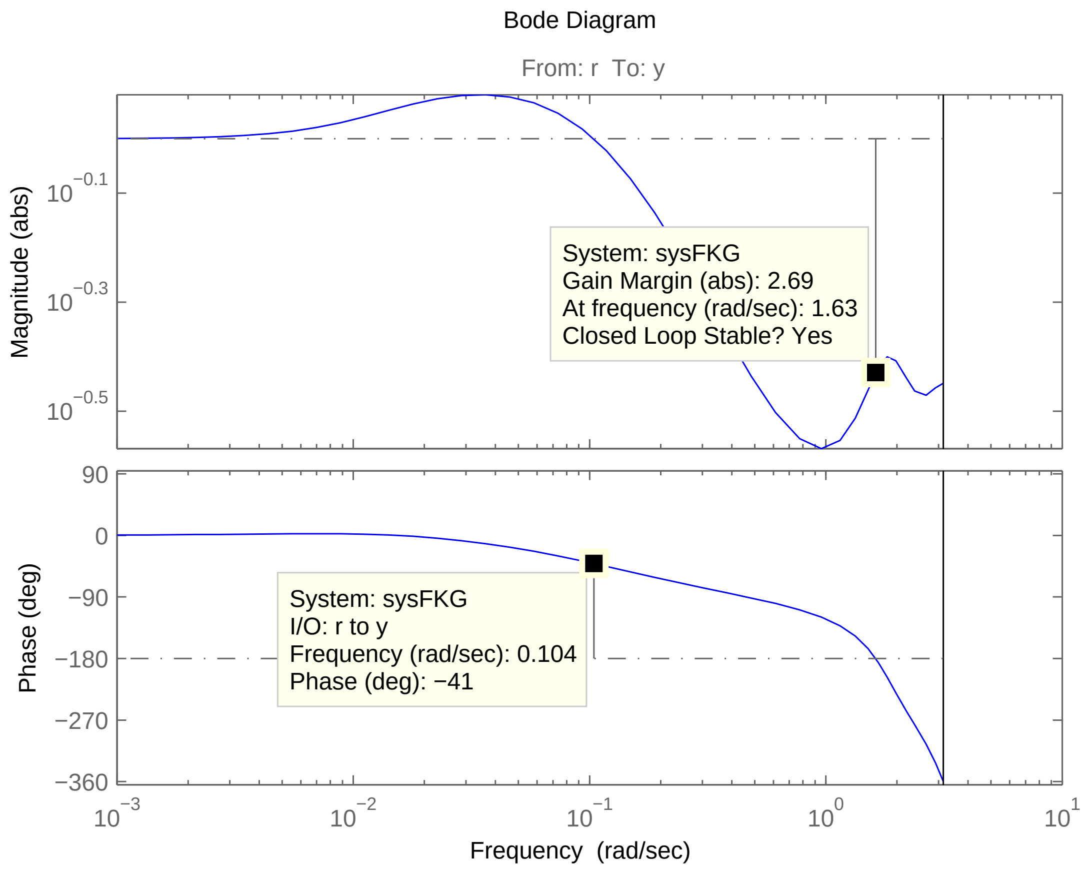


Figure 4.9: The closed loop bode plot of our system shows that we have a gain margin of 2.69 and a phase margin of 139 degrees. These are sufficient margins for robust closed loop stability.

It is important to note that the delays we measured when obtaining our model were just estimates. Not only may they be incorrect, but they may change depending on how much water

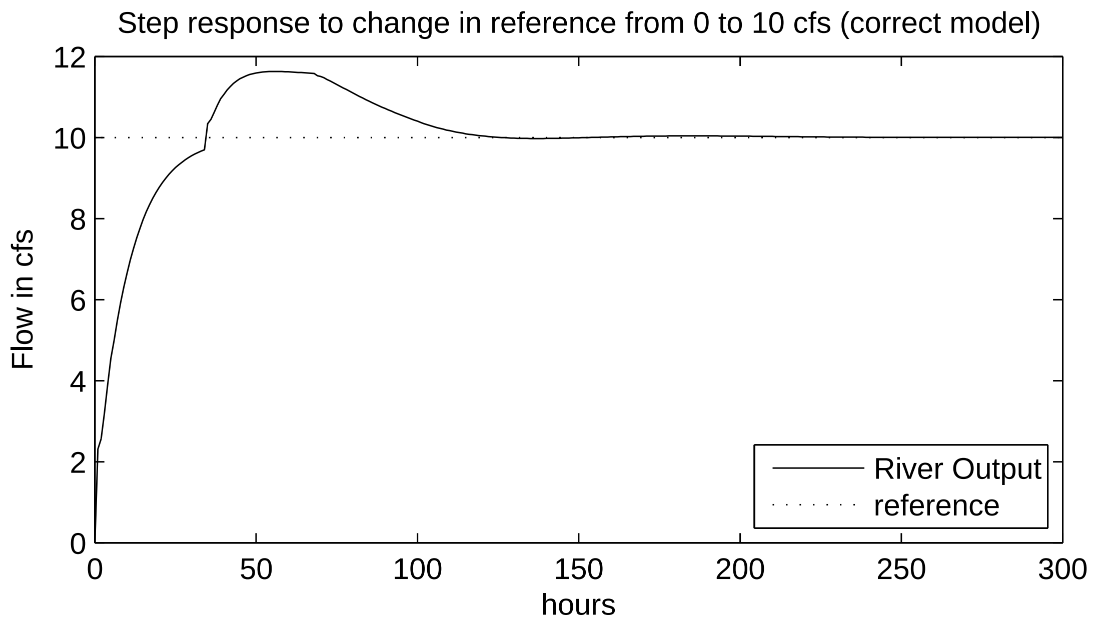


Figure 4.10: The step response of the closed loop system with both feed forward and feedback portions of the controller.

is flowing in the river. The more water there is the faster the water flows and the less delay there should be in our system. Conversely, the slower the water flows, the greater the delay will be. Because of this uncertainty, we need to be sure that the closed loop system is still stable if the delay in the river is different than we modeled. Figure 4.11 shows the step response of the closed loop system with eight hours more delay and eight hours less delay. Even with these extreme cases the closed loop system is still stable. With more delay there is poorer performance of the controller, however if there is less delay the controller performs better.

## 4.5 Controller Validation

In the preceding sections, we created a model for the Sevier River and designed a controller to control the reservoir release. We have shown that the designed controller is stable for differences in delay and has good nominal performance in tracking a step input. By nominal performance we mean performance on the system as modeled not taking into account inaccuracies in the model. In this section we are going to show that the controller remains stable even with inaccuracies in the model, called robust stability. We will also show the robust performance of the controller, which

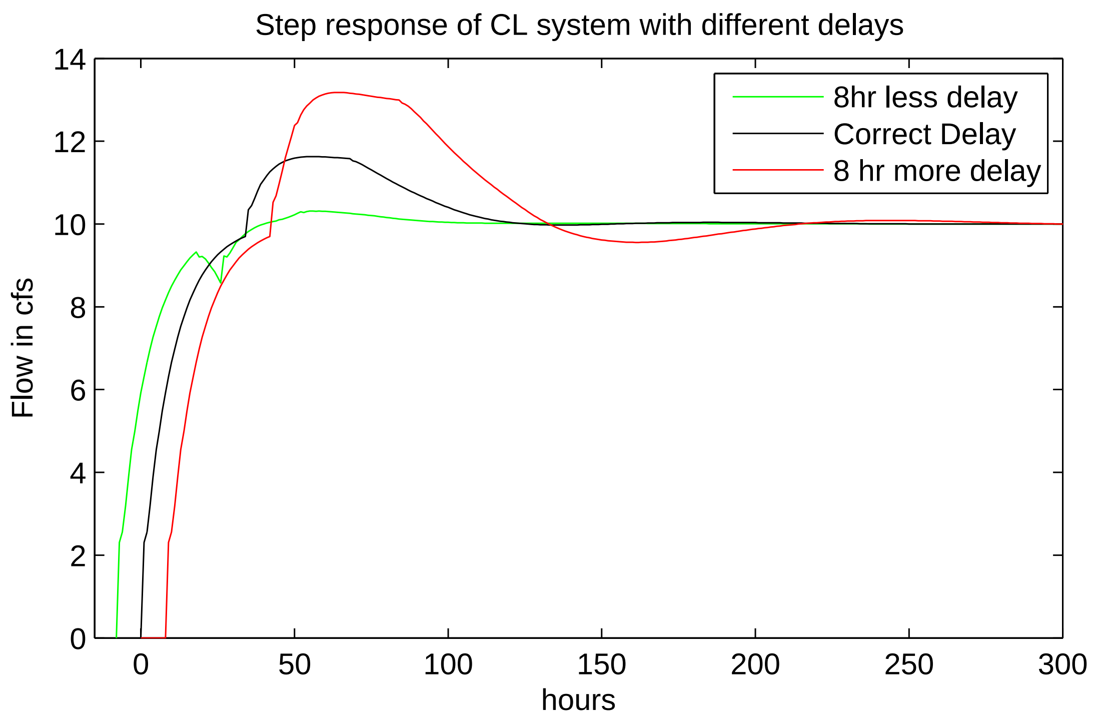


Figure 4.11: Closed loop sensitivity to delay uncertainty. If the river has more delay than modeled the performance is worse but the system is still stable. With less delay the performance is actually better.

is that the performance of the controller remains acceptable with disturbances and inaccuracies in the model.

### 4.5.1 Stability and Performance With Disturbances

#### Step Disturbance in Canal Input

The first type of disturbance to consider is if a canal suddenly takes more water than is ordered. This can happen because of an inaccuracy in the order or the canal measurements. It can also happen if the true impact of a canal input on the river flow is not -1, but some other value. This would mean that the parameter for the canal input is incorrect. Figure 4.12 shows the river output at steady state flowing at 10 cfs. At 100 hours a 10 cfs step input to a canal occurs. This canal disturbance has no order attached to it and so there is no way for the feed forward controller to compensate for the disturbance. There is a delay of 20 hours before the disturbance affects the river output. At that point the controller begins to make changes but the river flow continues to drop for 32 hours since it takes that long for the changes to reach the end of the river. From the time the changes start to affect the river output it takes approximately 35 hours to correct back to within 10% error of the reference.

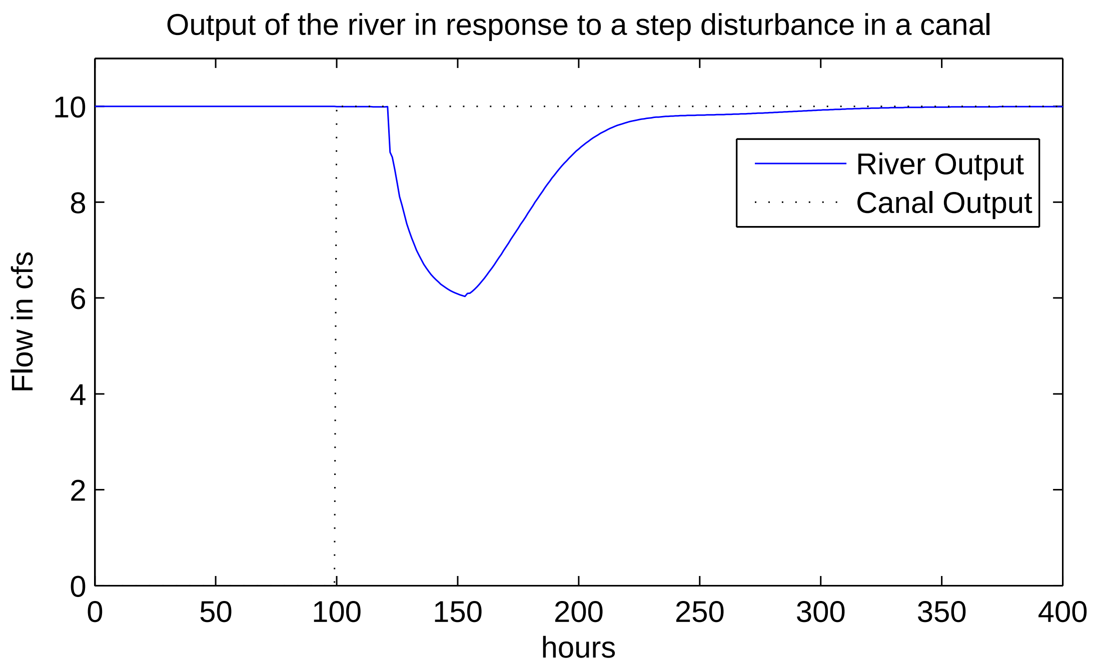


Figure 4.12: The river output in response to an unmodeled 10 cfs step input in one of the canals with delay 20. The disturbance is attenuated by 60% and eventually returns back to zero steady state error.

The performance of this controller at rejecting disturbances in inputs seems disappointing, over 100 hours to correct from a step disturbance. While this is slow there are several good results of this performance. First, stability is maintained in the controller. Second, the disturbance in the canal was 10 cfs, yet the error in the outflow was mitigated by 60%. Third, this is an extreme situation where the disturbance was held out indefinitely. Usually when canal operators make errors in the order it is corrected within a day, which would help the performance of this controller. Finally, this response does not use the feed forward part of the controller because we assume for a worst-case scenario that the canal operator may not change the order. Once the canal input was changed from the order, the order could have been automatically updated, which is how the system operates under normal conditions unless a manual override is made by the canal operator. This would still have caused some error but the feed forward part of the controller would have kicked in 20 hours before the error was noticed at the end of the river by the feedback part of the controller. This would have dramatically reduced the final disturbance.

### Incorrect Canal Delay

The next disturbance to validate is if the delay of a canal on the actual system is different than we have previously modeled. To perform this validation the same step disturbance of 10 cfs is going to be given to a canal with supposed delay 20 hours. This time there will be an associated order 20 hours ahead and the actual delay for the river will be varied. Figure 4.13 shows the river output for the canal delay being off by two and four hours.

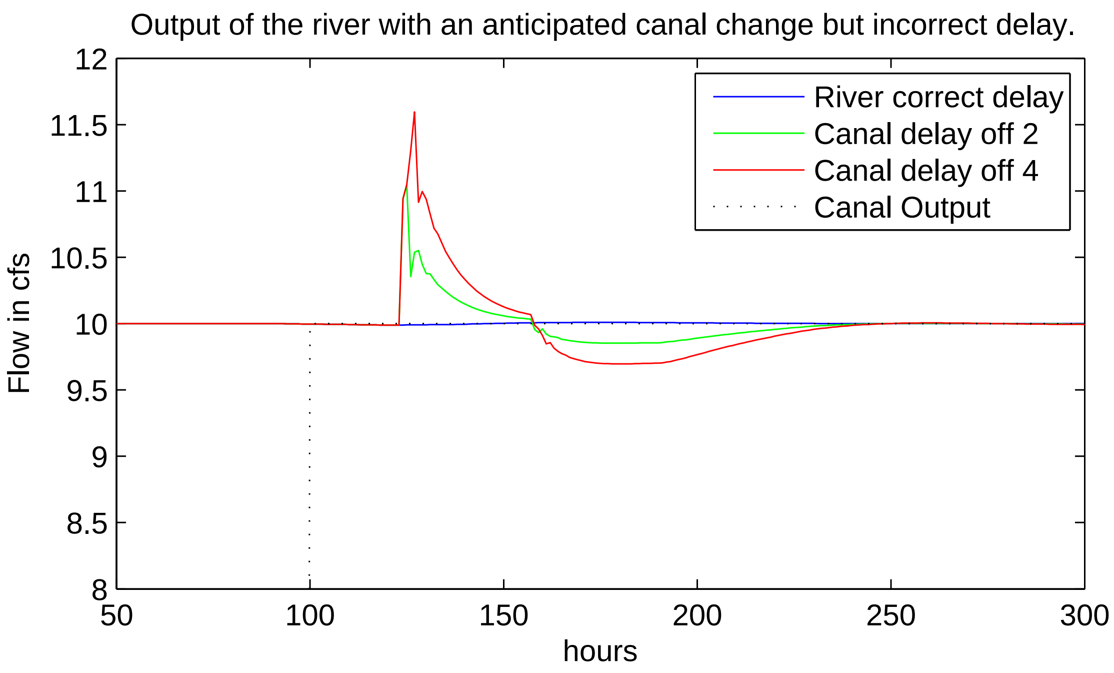


Figure 4.13: Shows the closed loop sensitivity to delay uncertainty. If the river has more delay than modeled, the performance is worse but the system is still stable. With less delay the performance is actually better.

The controller still maintains stability with this disturbance and the errors are only outside of the acceptable error of 10% (as stated above, measurements are only accurate to about 10%) for a few hours with the 4 hour discrepancy, and not at all with the 2 hour discrepancy.

### Sensitivity to Input and Output Noise

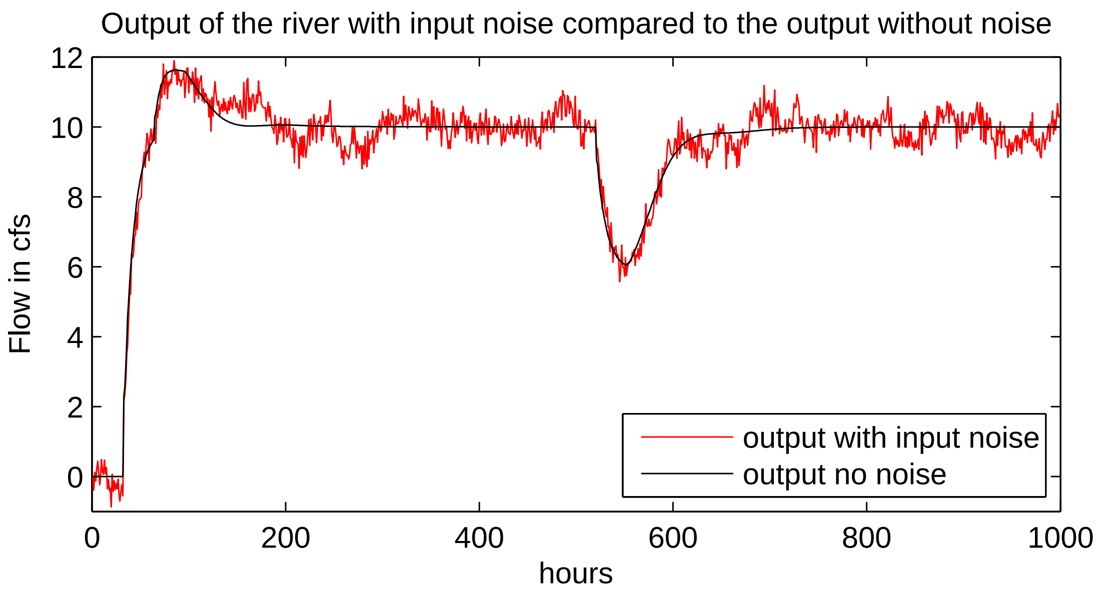
High frequency white noise is not too damaging to the system. In fact we have assumed that many of our measurements are going to have noise on them. Figure 4.14 shows the output of the system when noise is added to the input and the output. Input noise is added to the reservoir release and is shown in the top graph. Output noise is added to the output measurement and is shown on the bottom graph. The noise is added to a simulation with a step input at time zero and a step disturbance on a canal at time 500.

Because of the large delays in the system and the slow response of the controller, it is unable to correct for noise in the system. It takes 32 hours for changes in the input to be seen at the output so reducing the noise which can be changing every hour, is not possible. We can see from figure 4.14 that the closed loop system maintains stability with the noise. General performance does not seem to be slowed down much due to the noise. The rise time and the disturbance rejection do not take any longer.

### 4.5.2 Simulation on 2008 Data

As a final method of validating the controller it will be tested on the true river data for the summer of 2008. The desire is to validate whether the controller remains stable and meets desirable control objectives with noisy data. Another goal is to say what the controller would have done if it had been running in 2008 and how much water it would have saved. Since there is not any order data for 2008, we use the actual amount of water that was flowing out of the canal as its order. Under normal canal operation this should be within our measurement error of what the actual order would have been. We then give that order to the feed forward controller with enough advance that it can make any changes it needs. Figure 4.15 shows how well the controller performs when we enter the actual outflow as the reference command to track.

The controller is able to track the actual reference command very well, better than the model alone predicted the output to be, see figure 4.5 top. Next the controller was run with a reference command of one cfs, and again with a reference command of five cfs. Figure 4.16 shows



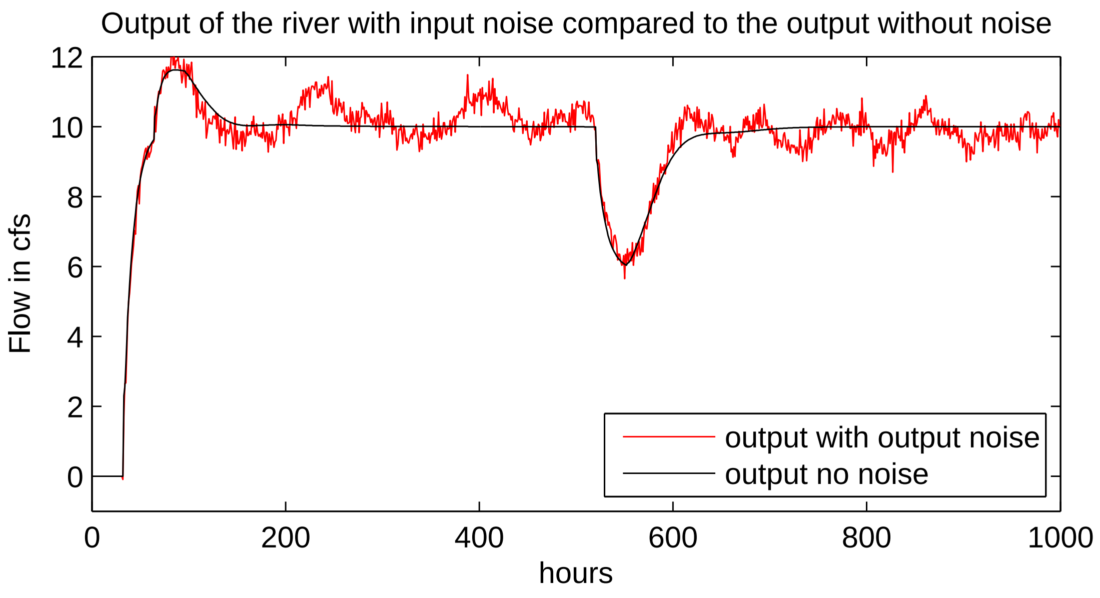


Figure 4.14: Top: Output of the system when white noise is added to the input, compared with the output of the system with no noise. Bottom: Output of the system when white noise is added to the output, also compared to the system with no noise.

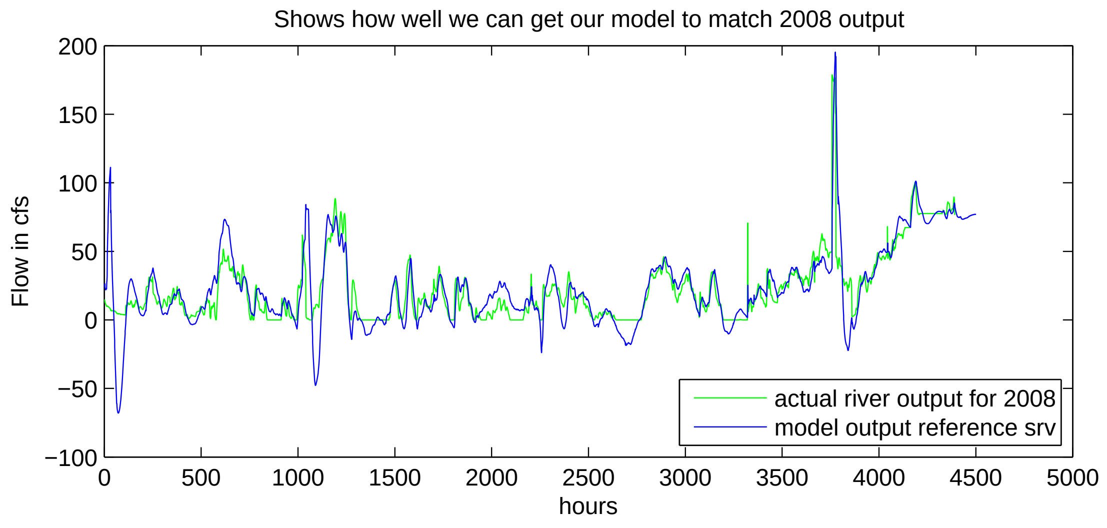


Figure 4.15: Controller output when run on 2008 data with actual river output as the reference command to track. The controller is able to match the output very well. Orders were taken from the actual canal inputs, which mimics how under normal operation canal inputs are made to match the water orders.

the output of the controller with each reference signal, red and blue respectively, as well as the actual river output in green.

From figure 4.16 it is easy to see that the controller would have conserved water compared to the actual flow in 2008. The controller also shows the value of the water going negative. This represents not enough water being let out of the reservoir and a shortage to the last canal. This is not desirable. In fact, it may be more desirable to waste a little more water in order to not have a water shortage. One way to do this would be to increase the reference command until the controlled output goes below zero only some of the time. Table 4.3 shows the different amounts of savings and water shorted by running our controller with varying reference points.

## 4.6 Platform Implementation Details

In this section we describe how the automated water management platform is implemented. Up until now the focus has been on creating a model of the river and designing a controller for the reservoir gate. This is the work that must be done by a student in order to use this platform. Once

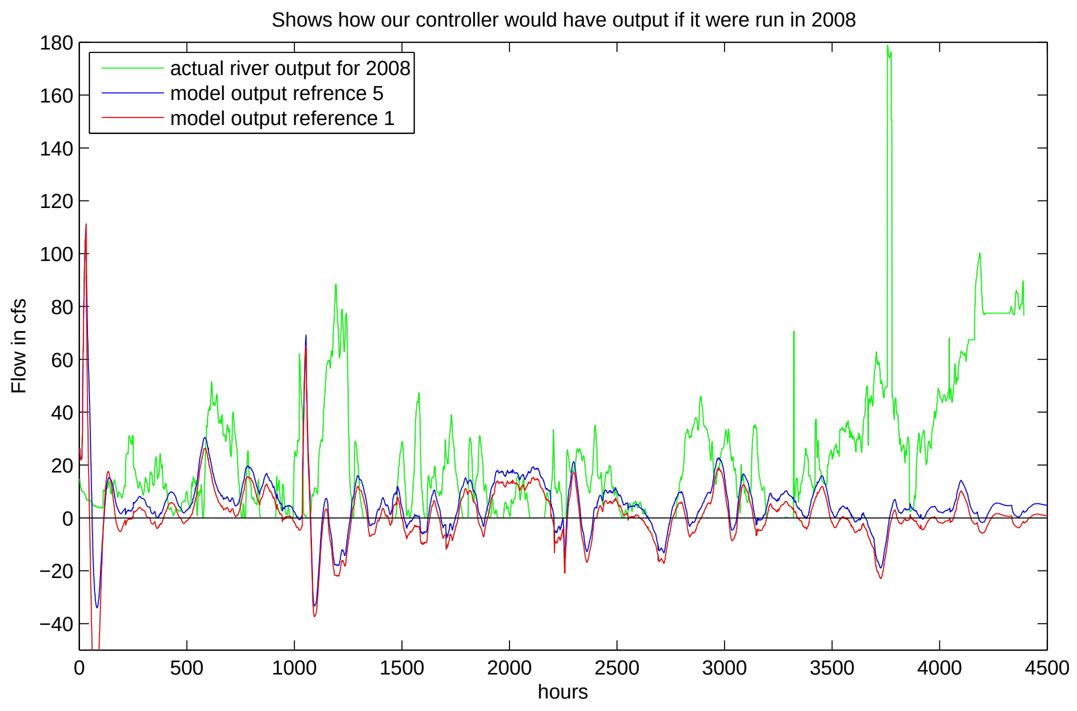


Figure 4.16: Actual river output (green) compared with system output with the controller actuating the reservoir release. The controller was given a reference signal of 1 cfs (red) and 5 cfs (blue) to try and maintain. The controller saves water but also may cause a canal to not get enough water because it goes below zero several times.

Table 4.3: Shows the amount of water that could have been saved and the amount of water shorted to the last canal, if we would have run our controller in 2008. All values are in Acre Feet. As the controller tries to hold the river release closer to zero we save more water but we also tend to not let enough water out some of the time.

| Reference Command | Water Saved | Water shorted (Less than zero) |
|---|---|---|
| r = 1 | 6514 AF | 650 AF |
| r = 5 | 5508 AF | 312 AF |
| r = 10 | 3985 AF | 140 AF |
| r = 15 | 2375 AF | 55 AF |
| r = 20 | 716 AF | 20 AF |

the controller is created it can be inserted into the platform which will implement the controller on the reservoir.

### 4.6.1 Platform Architecture

The architecture of the platform contains three pieces which work together to control the reservoir, and also the gates on the canals. The first part is the on line water order form. This form allows canal owners to go on line and enter the future orders for their canals. These orders can then be used by the controller as explained above to make a feed forward estimate of the needed control input. Also when the time for the water order comes up the platform can actually move the gate to the appropriate place just as it does for the reservoir.

The second part is the implementation of the controller itself. The controller is implemented in Linux so that it has access to the database of water orders and current river flows. It has been implemented as described in the previous section using the PERL language and it uses Perl's DBI to access the database. Once it calculates how much water to let out of the reservoir it writes the value to a file to be used by the third part of the platform. It also writes to the file the most current order for each canal so that those gates can also be set.

The third part of the platform is a program that constantly runs on a windows server. The windows machine has a shared folder with the Linux machine and it waits until the controller file appears there. Once it sees the file it opens it and then uses the API of a third party software called Loggernet to communicate via radio to each station site to set the flow for each gate. Each gate including the reservoir gate has a local controller which automatically adjusts the gate up and down to match the desired flow set by the platform. Loggernet only runs on windows which is why this third part of the platform is on windows.

One major benefit of this three part architecture is modularity. It allows us to completely replace the controller with a new design or implement it in a new language without changing the rest of the platform. As more students use this platform they can just focus on the controller design and not have to worry about communicating with the stations.

Each student who creates a controller can just set up their controller to run on the Linux machine via a cron job every hour. There they have access to all past data and future water orders. Their algorithm needs only to output a text file with the amount of water to let out of the reservoir (in cubic feet per second) and the platform will handle everything else. Of course before any student can insert their control algorithm there must be sufficient testing to ensure that it will work on the real river, just as we have shown above with the controller design in this thesis.

### 4.6.2 Communication to the stations

Loggernet software handles the communication between the server and each of the stations on the river. This communication travels via ethernet to a data hut near Richfield, then from there travels by radio to the different stations. These radio communication channels present the single most challenging part of making the platform functional. The radio channels can be very noisy and are not very reliable. It can take a long time to connect or transmit data and the signal can be dropped entirely. Also the entire system of stations is on just a couple radio frequencies. If someone is using the frequency to change a station's flow or read its values then the platform will not be able to connect to any other station on the same frequency. This can become more serious when someone forgets to log off and keeps the line tied up for hours.

While communication errors are still present, much care has been taken in the platform design to mitigate communications problems and increase the reliability of the system. The platform is designed to only call out to the stations once per hour to make needed changes. This reduces the traffic on the radio frequencies during the rest of the time. The platform also is designed to make all the needed changes on the canals so there should not be a need for other people to use the communication channels as often.

Before it makes any communications with the stations, the platform checks to determine whether there has been a significant change in the order for the canal or reservoir release. A significant change is .25 cfs for a canal and 2 cfs for the reservoir. If there has not been a significant change, it will not call to make a change. The only communication taking place is for those stations where there is a significant change.

When the platform actually does need to make a change it waits a long time for the communication to be established. If communication fails it will quit all communications and wait for 30 seconds and then try again. After three separate tries it will give up until the next hour.

### 4.6.3 Security and Reliability

Because water is the livelihood of those who are served by the reservoir, great care must be taken to make the system secure and reliable. If the system fails the failure must be recognized in time to minimize the damage. If a farmer looses water for a few days his crop for the year could be ruined. This danger causes many of the water users to be skeptical to allow the platform to control the reservoir and the gates.

The on line water order entry form uses SSL encryption and a password system to ensure the identity of those using the system. Those who use the system must have their account manually verified and be given access to each canal they are to be able to control. Users are not allowed to see any canal that does not belong the them.

If at any time the platform has trouble making a change on a canal or the reservoir then it will send an email notification that there is a problem. The email will then be forwarded via short message service (SMS) to the administrator's cell phone. That way someone knows immediately and can evaluate and fix the problem. Possible causes of these errors could be communication error, a software error, a file read or syntax error, or even an alarm if the flow is outside of some prespecified range.

As a guaranteed fail-safe, every station is equipped with two flags that must be turned on for the platform to control the gate. If at any time the platform must be discontinued for an emergency reason one can simply change either of the flags. The first flag, AUTOMODL, must be set to 1 otherwise the platform will not attempt to change the set flow. The second flag, AUTOGATE, must be set to 1 otherwise the local control will not try to move the gate to the set flow. Having these fail-safe flags means that in the worst case scenario one can just turn those off and move the gate manually.

## 4.7 Conclusion

In this chapter we have designed a learning platform for water management. This platform includes all of the applications and software necessary to make a reservoir gate control algorithm move the corresponding physical gate. This platform is designed in a way that any control algorithm in any language may be used, provided that it can access the database. This allows students to design control algorithms and try them on the actual river, provided that their solution has been sufficiently tested and verified.

We have also used the decision architecture introduced in chapter one, to design a control algorithm to be used on the Sevier River. We modeled the river using a parameterized mass balance model, and verified the model with actual river data. We designed a two part controller with both feed forward and PI feedback controllers. We showed that this controller design is robust to modeling errors, noise, and unmodeled disturbances, and that the controller design is likely to have saved a significant amount of water during the 2008 irrigation season using 2008 data.

# Chapter 5

# Conclusion

At first glance portfolio optimization, business intelligence, and automated water management might seem like completely separate problems in three different fields. However, all of these problems have several things in common. They are all problems that can lend themselves to study in decision autmation. Each problem relies on making intelligent decisions from data. In doing so their solutions handle information in similar ways and yield algorithmic decision processes. Finally, each problem is simple to understand but difficult to solve. Good solutions to these problems are actively being researched and developed.

## 5.1 Decision Architecture Applies to Each Problem

We have introduced a decision architecture which leverages these similarities and breaks each of these problems into four smaller problems that work together to produce an algorithmic decision process. These problems are the decision problem, the learning problem, the model reduction problem, and the verification problem.

The decision problem focuses on making the best decision based on a model of consequenses for each decision and an objective function which rates possible consequenses. In portfolio management the decision is to determine which securities should be purchased in order to create a portfolio which will have the most value after a certain amount of time. In the business intelligence problem the decision is to set the price at which to sell each product in order to maximize profit. In the automated water management the decision is to release enough water from the reservoir to meet water demands without letting out any excess water.

In the learning problem a model of possible consequenses is chosen from a model class using historical data and a measure of quality. These models may contain varying amounts of uncertainty, which may affect how the decision problem is solved. In portfolio management models are learned in order to predict future values of securities. Models in business intelligence predict future sales of each product. In automated water management models are used to predict the effects of released water on downstream flow.

Each problem area may be complex enough that models used for learning and decision making may not be able to be computed. The model reduction problem finds simplified models which are still accurate and lend themselves to be computationally tractable. In portfolio management the number of stocks being considered must be reduced for some models. In business intelligence model parameters must be reduced by limiting the number of products whose prices influence another product's price. In automated water management the physical system is so complex that a simplified model must be designed to capture the most important dynamics of the river.

The verification problem involves designing experiments that test the decision process in order to show the quality of the solution. Each solution needs to be verified to determine whether there is evidence that the solution does not perform as desired. For portfolio management this can be done by comparing two different decision processes in an online trading platform such as the Tour de Finance. For the business intelligence problem predicted quantities sold of a product can be compared with its actual quantities sold. For automated water management past water data can be used to validate a water management algorithm to an extent as shown in chapter four. The algorithm can also be tried on the real river to determine how well it works, which is planned for the future irrigation season.

## 5.2 Three New Learning Platforms Were Created

In addition to formulating a decision architecture we have created learning platforms in three problem areas: portfolio management, business intelligence, and automated water management. These learning platforms are good platforms to teach decision automation to students for several reasons.

All of the platforms are non-traditional applications for students to study decision automation. The problems addressed by each platform are easy to understand by students of all diciplines. Portfolio optimization and business intelligence do not require extensive mathematical and engineering prerequisites, so they can be taught early in the educational process. This allows students an opportunity to decide if they want to pursue education in the field.

For the portfolio management platform we created an in depth tutorial showing how a student could use the platform to learn about algorithmic decision processes. We showed examples of how students were naturally led to consider the four problems of decision, learning, model reduction, and verification, and consider the interactions between them to address important issues currently being researched in the field. We also showed how students could be introduced to some of the important results in portfolio optimization by using the portfolio optimization platform.

For business intelligence we developed a demand forecasting platform that uses live BYU Bookstore sales data to aid students in developing demand forecasting algorithms and validate them on real products. Students are able to create their demand forecasting algorithms in matlab and then upload them to the database which automatically adds them to the website. A web interface allows any user to select any product and see the actual sales along with the predicted sales of any algorithm.

For automated water management we created an algorithmic decision process using our decision architecture. We modeled the river, designed a decision algorithm, or controller, and validated the results carefully to make sure that it would work as designed. Because of the model selected, model reduction was not necessary. We then implemented this controller design in a modular way so that the infrastructure for controlling the gate can serve as a platform for other controller designs. We designed into the platform several safety features that protect the river system against software error to help ensure that the correct amount of water is delivered.

Future work can proceed in several different directions. Since each of these problem areas are areas of current research, work can be done to develop improved solutions to each problem. This could include developing higher fidelity models and/or improved decision solutions for portfolio optimization, business intelligence, and automated water management. Work can also be expanded on each of the four problems of the decision architecture. Better algorithms are needed for learning, decision making, model reduction, and verification. Finally, future work can be directed at the interaction between these different problems, as discussed in section 2.3.2. It is important to find ways to improve how these problems work together, to yield better algorithmic decision processes.

## References

[1] Salman Ahmed and Mohd N. Karsiti. A testbed for control schemes using multi agent non-holonomic robots. In *Electro/Information Technology, International Conference on*, pages 459–464, Chicago, IL, USA, May 2007.

[2] Karl J. Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2 edition, 2008.

[3] A. Bindal, S. Mann, B. N. Ahmed, and L. A. Raimundo. An undergraduate system-on-chip (soc) course for computer engineering students. *IEEE Transactions on Education*, 48:279–289, May 2005.

[4] Campbell Scientific, Inc. *Loggernet Software Development Kit Programmer's Reference*, 2.2 edition, 2006.

[5] Jie Chen and Guoxiang Gu. *Control-oriented system identification: An H-inf approach*. Wiley, 2000.

[6] Casey Dougal, David Merriman, Jeffery Humpherys, and Sean Warnick. Competition dynamics in a virtual fund management system: The tour de finance. In *American Control Conference*, pages 1817–1822, July 2007.

[7] John Doyle, Bruce Francis, and Allen Tannenbaum. *Feedback Control Theory*. Macmillan Publishing Co., 1990.

[8] Geir E. Dullerud and Fernando Paganini. *A Course in Robust Control Theory: A Convex Approach*. Springer, 2000.

[9] Arthur J. Helmicki, Clas A. Jacobson, and Carl N. Nett. Control oriented system identification: A worst case/deterministic approach in h-inf. *IEEE Transactions on Automatic Control*, 36(10):1163–1176, October 1991.

[10] R. Hill and A. van den Hengel. Experiences with simulated robot soccer as a teaching tool. In *Information Technology and Applications, International Converence on, ICITA*, volume 1, pages 387–390, July 2005.

[11] G. Hovland. Evaluation of an online inverted pendulum control experiment. *IEEE Transactions on Education*, 51(1):114–122, February 2008.

[12] Saul Jimenez and Wen Yu. Stable synchronization control for two ball and bearn systerns. In *Electrical and Electronics Engineering, International Conference on, ICEEE*, pages 290–293, Mexico City, Mexico, September 2007.

[13] Tohru Katayama. *Subspace Methods for System Identification*. Springer, 2005.

[14] Konno, Hiroshi and Yamazaki, Hiroaki. Mean-absolute deviation portfolio optimization model and its applications to tokyo stock market. *Management Science*, 37(5):519–531, May 1991.

[15] M. D. Koretsky, D. Amatore, C. Barnes, and S. Kimura. Enhancement of student learning in experimental design using a virtual laboratory. *IEEE Transactions on Education*, 51(1):76–85, February 2008.

[16] X. Litrico, V. Fromion, J.-P. Baume, and M. Rijo. Modelling and pi control of an irrigation canal. In *Proceedings of European Control Conference ECC03*, Cambridge, UK, 2003.

[17] Xavier Litrico. Robust imc flow control of simo dam-river open-channel systems. *IEEE Transactions on Control Systems Technology*, 10(3):432–437, 2002.

[18] Lennart Ljung. *System Identification Theory for the User*. PTR Prentice-Hall Inc., 1987.

[19] David G. Luenberger. *Investment Science*. Oxford University Press, 1998.

[20] P.-O. Malaterre and J.-P. Baume. Modelling and regulation of irrigation canals: existing applications and ongoing researches. In *Proceedings of IEEE Conference on System, Man, and Cybernetics*, pages 3850–3855, San Diego, CA, USA, 1998.

[21] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):77–91, March 1952.

[22] M. Maxwell and S. Warnick. Modelling and identification of the sevier river system. In *Proceedings of American Control Conference ACC06*, 2006.

[23] Su Ki Ooi and Erik Weyer. Closed loop identification of an irrigation channel. In *Proceedings of IEEE Conference on Decision and Control*, pages 4338–4343, Orlando, Florida, USA, 2001.

[24] Kameshwar Poolla, Pramod Khargonekar, Ashok Tikku, James Krause, and Krishan Nagpal. A time-domain approach to model validation. *IEEE Transactions on Automatic Control*, 39:951–959, May 1994.

[25] W.F. Sharpe. A linear programming approximation for the general portfolio analysis problem. *Journal of Financial and Quantitative Analysis*, 6:1263–1275, December 1971.

[26] B.K. Stone. A linear programming formulation of the general portfolio selection problem. *The Journal of Financial and Quantitative Analysis*, 8:621–636, September 1973.

[27] Chen Wenwei, Zhang Jinyi, Li Jiao, Ren Xiaojun, and Liu Jiwei. Study on a mixed verification strategy for IP-based soc design. In *High Density Microsystem Design and Packaging and Component Failure Analysis, Conference on*, pages 1–4, Shanghai, June 2005.

[28] Erik Weyer. System identification of an open water channel. *Control Engineering Practice*, 9:1289–1299, 2001.

[29] Erik Weyer. Lq control of an irrigation channel. In *Proceedings of IEEE Conference on Decision and Control*, pages 750–755, Maui, Hawaii, USA, 2003.

[30] M. Yamakita, T. Hoshino, and K. Furuta. Control practice using pendulum. In *American Control Conference*, volume 1, pages 490–494, San Diego, CA, June 1999.

[31] Kemin Zhou and John C. Doyle. *Essentials of Robust Control*. Prentice-Hall Inc., 1998.