Order Number 9324949

Distributed economic systems with agents that learn

Perry, Stanley Foster, Ph.D.

Portland State University, 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DISTRIBUTED ECONOMIC SYSTEMS WITH AGENTS THAT LEARN

by

STANLEY FOSTER PERRY

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
SYSTEMS SCIENCE

Portland State University

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the dissertation of Stanley Foster Perry presented May 3, 1993.

George G. Lendaris, Chair

Kuan-Pin Lin

Thomas Potiowsky

Martin Zwick

Beatrice T. Oshika

APPROVED:

Beatrice T. Oshika, Director, Systems Science Ph. D. Program

Roy W. Koch, Vice Provost for Graduate Studies and Research

AN ABSTRACT OF THE DISSERTATION OF Stanley Foster Perry for the Doctor of Philosophy in Systems Science presented May 3, 1993.

Title: Distributed Economic Systems With Agents That Learn

George G. Lendaris, Chair

Kuan-Pin Lin

Thomas Potiowsky

Martin Zwick

Beatrice T. Oshika

Economic systems are distributed in the sense that economic agents make decisions without any central control. Prices, quantities, wealth, and market structure emerge from the interaction of agents acting in their own self interest. The concepts and language of systems science are used to define economic systems in a manner that captures and articulates the distributed nature of

economic systems. Further, the systems definition permits multiple views of the economic system, and in addition, allows the agents to "step outside" the system in order to study it.

Economic systems are defined in such a way that it is feasible to construct artificial economic systems, and in particular, ones that are composed of self-interested agents that operate according to principles that are prescribed by the researcher. An artificial economic system was actually constructed and tested in a computer environment. The model was verified with reference to several theoretical models such as static and adaptive expectations. The system constructed allows up to 1000 agents to interact without any central control.

A computer "blackboard system" is used as the architecture for providing common information to the agents in the artificial economic system. The blackboard design successfully allows complex agents to compete and trade in an artificial economic system created by the researcher. Prices, quantities, wealth, and market structure emerge naturally in the artificial economy that depend on the characteristics and prescribed strategies of the agents in the system. After a transition period, the trading frequently produces price and quantity time series that have the characteristics of a random walk, a condition that is well known in real world markets.

Three classes of producer agents were used in these artificial economic systems: optimizing agents that incorporate neural networks, satisficing agents that incorporate very simple rule-based approaches, and Stackelberg agents that have knowledge about the consumers in the system, but do not have knowledge about their competitor's strategies or intentions. Neural networks are used to model the behavior and strategies of economic agents that can be said to learn, i.e., those agents that develop general principles for adapting to changing market conditions that transfer across markets. The focus of this research was on the producers in the system. The consumption side of the economic system was represented by a set of simple consumers.

An important result emerging from this research is that at least one agent out of four in these experiments with accurate knowledge about market demand increases the wealth of the system as a whole. Markets containing a single Stackelberg or neural agent produced far more wealth than markets composed only of satisficing agents. However, the agents with knowledge do not necessarily capture the highest share of the wealth.

The success of individual agents depends on the agent's trading strategy, as expected, and in addition depends on the combination of agents in the system. Certain strategies appeared to be flexible while others were brittle, and were easily foiled by changing the agents in the market, or by changing the market conditions.

Earlier studies attempted to use neural networks to simulate an entire economic system, but were rejected because the organizing principles of the two systems are not analogous. Additionally, neural networks were successfully tested for solving various economics problems that were not related to the simulation of economic systems. Neural networks were found to effectively solve problems with missing and redundant data that are not directly solvable with well known methods such as least squares.

ACKNOWLEDGEMENTS

I would like to recognize the committee of advisors that participated in this research. Each member of the committee made useful comments and suggestions that are included in this work. I would especially like to thank the committee chairman, George G. Lendaris, for his guidance and direction.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

INTRODUCTION TO DISTRIBUTED ECONOMIC SYSTEMS

Economic systems are distributed in the sense that every agent in the system acts independently. Unfortunately, the traditional method of studying economic systems assumes that the agents are identical, and that they can be aggregated into representative groups. Thus, an economic system is usually modeled by making simplifying assumptions that yield an abstraction of an economic system that is not distributed. The model is fundamentally different from the real system at a macro level.

Economic systems comprise agents such as individuals, households, firms, etc. that act independently according to their own private motivation and individual ability. The traditional approach to modeling these systems assumes that all economic agents are rational and optimize some specific objective, such as maximizing profit. The model is fundamentally different from the system at a micro level.

Allocation of scarce resources through trade and competition among agents under time constraints are the primary organizing forces in economic systems. Prices, quantities and wealth emerge naturally in economic systems as a consequence of these primary organizing forces. Unfortunately, the traditional approach assumes that at least one of these key emergent variables is fixed, and ignores the primary economic forces of scarcity, trade and time. The traditional model is fundamentally different from the system in the manner of organization.

I contend that any system that includes distributed agents that act independently and are organized by scarcity and trade is an economic system. Prices, quantities and wealth will emerge as properties of such a system, and these properties depend on the interaction of the particular agents in the system. I develop a conceptual model of economic systems that is consistent with the vocabulary and concepts of Systems Science, and then construct and test an artificial economic system based on these ideas. Additionally, neural networks are developed and tested for simulating agents in these artificial economic systems, and for analysis of economic data.

Economic systems are defined in Chapter II, using a model based on two views of a system. The first view sees the system as a unit that contains subunits and has observable attributes relative to its environment. The second view focuses on the subunits of the system, and these are organized in a manner that manifests the observed attributes of the system. The organizing principle of economics is identified as distributing scarce resources by trading.

The observable attributes of any economic system emerge from the interaction of agents in the system as they respond to changes in their environment. In an unrestricted economy, agents make decisions based on their own private knowledge, beliefs and strategies, but without any central control. The decision making in such a system is defined as being distributed.

The primary goal of this research is to develop distributed models of economic systems that mimic selected attributes of genuine economic systems. This area of research is novel because distributed models have not yet been widely used for studying economic systems, even though the economy is an example of a distributed computer. In order to develop distributed models of economic systems it is necessary to better understand the concept of a system, and specifically the concept of an economic system. These concepts are developed in Chapter II.

Models of economic systems are discussed in Chapter III, including well known classical models, and the fundamentals of the distributed approach for modeling economic systems is reviewed. Of special importance to this discussion is the role of expectations, and the different paths individual economic agents could take to forming expectations. From my perspective, models of true economic systems must be flexible enough to include agents with different belief systems, goals, abilities and problem solving strategies. The ability to adapt, learn and gather knowledge are identified as important survival skills for economic agents. In particular, 3 classes of agents are discussed that are consistent with existing economic literature, and consistent with the concepts of agents in distributed economic systems: optimizing agents, satisficing agents and Stackelberg agents.

I assert that economic systems can be understood as a type of parallel distributed computer. That is, an economy is a computer made up of a large number of computing elements all operating at the same time, with little (if any) central control. A free market economy comprises a massive number of activities that proceed simultaneously (massively parallel), and has little or no central control. However, as mentioned by Simon [1981, 1982], both hierarchical and distributed aspects are present in any economy.

Assuming that each individual in an economic system acts according to utility maximization, then the economy does indeed act as a distributed computer, since there is no central processing unit that computes the prices or the quantities of goods in the economy at the macro level. However, some centralized control is apparent in business firms, governments and markets. On closer inspection, this central control can be seen as a feature that emerges in certain types of economic systems. The idea that an economy acts as a distributed computer is not entirely new. Herbert Simon appears to be the first writer to mention this interpretation of economic systems. However, the view he presents was strongly influenced by earlier writings, going back as far as Adam Smith, that emphasize the distributed nature of economic systems. The alternative to distributed economic systems are systems that exhibit central control or hierarchy. A common type of hierarchic economic system is a firm in which decisions are made at the executive level. Obviously, both hierarchic and distributed economic systems are observed in the world, and it appears that most economic systems embody some combination of control. A convincing model of an economic system would also allow differentiation of agents in the system, and include agents with varying complexity. A significant development of this research is a better understanding of the dual hierarchic / distributed nature of economic systems, from a computational perspective.

The typical researcher deals with imperfect information about the agents and cannot predict in advance what properties will emerge from the interaction of the agents. However, according to the systems paradigm, even if the researcher had perfect information about the agents, there are some properties that emerge from the interaction of the agents that cannot be predicted solely from the known attributes of the agents.

The traditional approach to answering questions in economics relies on aggregation to reduce problem complexity. Unfortunately, a collateral result of aggregating data into groups that operate in a supposedly identical manner is to mask the variability that is inherent in individuals, and to reduce the observed diversity of the world. An important result of the present research is the demonstration that individual roles are indeed important for building realistic models of economic systems. In particular, it is demonstrated that a single agent with knowledge about the system can dramatically impact the status of all agents in the market.

It has long been recognized that economic processes are adaptive rather than static (see [Murphy, 1965] for an example). Other authors have suggested a rational expectations hypothesis to explain the behavior of economic agents [Muth, 1961]. Simulation of economic systems have typically included adaptive or rational processes but did not model the economy as a collection of independent and distributed agents with free agency (see [Naylor, 1971]). An early paper by Aoki suggested modelling the interaction of economic agents as a stochastic estimator, which is very similar to a neural network approach [Aoki, 1979]. However, Aoki still modelled industries as collections of identical agents. Fourgeaud, Gourieroux and Pradel [1986], investigated learning and rational expectations models with agents that learned by using linear regression . Bray and Savin [1986] follow a similar approach using a cobweb model and endowed agents with a linear regression learning routine. The agents learned to converge to the theoretically expected equilibrium in a short time. In all those cases, all the agents were identical and had a simple problem to solve.

A related area, the field of experimental economics, has slowly emerged as a significant area of economic research (see [Smith, 1982], [Plott, 1982] for survey articles). The approach of experimental economics is to create micro markets with human and sometimes computer participants. One recent example of this approach was the Santa Fe Institute double auction market for computer agents [1990] in which I participated. This market was a distributed computational system based on economic agents that were designed to compete in a double auction. In a double auction market, a single unit is offered for sale, the sellers make offers to sell the unit, while the consumers make bids to buy the unit. The seller with the lowest offer and the consumer with the highest bid are free to trade with each other. The competitive system is similar to the system developed here except that the Santa Fe system is far more constrained because of the double auction framework. Additionally, the Santa Fe system contains a "monitor" that examines all bids and offers tendered by the agents, eliminating those that do not conform to the institution of the double auction. In the double auction framework, the agents and market are designed with a strong tendency toward equilibrium. In contrast, the market systems designed in my research are unconstrained by the double auction institution, and have no controlling monitor to eliminate behavior that is illogical from the researcher's perspective. In addition, the focus of my research is directed toward longer term learning, rather than the short term approach to equilibrium in a constrained market. As a result, pathological economic conditions such as underproduction and speculative bubbles are possible in my distributed economic markets, but do not develop in the Santa Fe Institute double auction type models.

As mentioned above, adaptive behavior has long been of interest to economists, and more recently, learning and artificial intelligence are entering the economics literature. For instance, Arthur suggests a bounded rational approach for designing artificial economic agents [Arthur, 1991]. Holland and Miller suggest the use of genetic algorithms for developing artificial adaptive agents to study economic phenomena [Holland and Miller, 1991]. Kiyotaki and Wright show that a good representing money emerges naturally in artificial markets where specialized agents meet randomly to trade goods [Kiyotaki and Wright, 1989]. Marimon, McGrattan and Sargent expand on the work of Kiyotaki and Wright by introducing more complex classifier agents for trading.

A research area that is in some ways similar to my approach to distributed economic systems is artificial life [Langton, 1991], [Levy, 1992]. The intent of researchers in the artificial life field is to study the emergent process that is called life. Automata interact and evolve in artificial ecosystems and display many of the qualities of life, and in particular, prosper or perish according to their fitness in their environment. While agents in economic systems share many of these qualities, such as independence, success according to fitness, and adaptability, the economic process is organized around scarcity and trade. This appears to me to be a distinctly different organizing principle from the life process, which typically includes scarcity, but not trade. While some well known authors such as Boulding [1981], have pointed out the isomorphisms between economics and ecology, I believe that the economic process is fundamentally different from the life process because of the economic organizing principle.

There is a second-order emergence in economic systems that depends on the prior (lower-order) emergence of an infrastructure complex enough to support the reasoning required for trade. In this research, I provide the lower order infrastructure in the form of computer programs that can reason about trade. I do not make the strong claim that these agents are alive, but I do claim that their interaction is the interaction of an economic system. In any case, it appears that the time series emerging from the processes of artificial life and artificial markets are similar and can be analyzed by similar methods.

The techniques of artificial intelligence become important in understanding how agents may develop knowledge, understanding and adaptability. Stackelberg agents are representative of agents with limited knowledge but without the ability to adapt and to learn. Satisficing agents are representative of the rule-based artificial intelligence strategies. Optimizing agents are representative of some computational neural network strategies.

Two types of distributed computer systems are considered in this research for use as models of distributed economic systems: neural networks and blackboard systems. Neural networks take their inspiration from the biological brain, where the neuron is the fundamental processing unit. Millions of neurons exist within the brain and operate together in a manner that manifests the attributes of the brain. These neurons operate in a way that is called massively parallel, where each neuron receives signals from, and sends signals to many other neurons, all at the same time. Each neuron is loosely connected to thousands of other neurons, but not all paths between neurons are of equal strength. Artificial neural networks (ANNs), or connectionist nets, are conceptually based on this aspect of brain structure.

In the neural element typically used for ANNs, the processing element has a simple nonlinear response function and may be connected in parallel to many other neural elements. The networks are said to learn in the sense that they modify their responses according to the stimuli they are exposed to. As is well known, the human brain naturally and easily solves problems that are quite difficult to solve with conventional computers. Such problems include pattern recognition, summarization of written material, and intuitive response and generalization.

Since the neural network is a distributed computer, I entertained the idea of modeling economic systems as a neural network. (This is not a model from the economics or neural network literature). However, instead of using a neuron as the basic processing (decision making) element in the network, an economic system would be conceptualized as a massively parallel network of human beings who are making economic decisions. The nodes of an economic system could represent individuals, each receiving and sending many signals to other units. The units may specialize so that not all units receive and send identical signals.

I rejected my neural model of economic systems because neural networks do not allow for the basic operating principles on which an economic system is organized. In particular, the neural network approach to distributed computing does not allow processes resembling trade. Additionally, complex hierarchic structures similar to economic entities are difficult to incorporate in the neural network schema. Neural networks fail to capture the structural and organizational nature of distributed economic systems, but are used successfully to model the behavior of agents that make up the system.

Neural networks are discussed in general in Chapter IV of this dissertation. Neural networks have many potential applications as computational models in economics, as well as for modeling the logic and adaptive behavior of economic agents. The use of neural nets to solve several statistical problems in economics is discussed in Chapter VIII.

A "blackboard" structure [Englemore and Morgan, 1988] is an alternative model of distributed computing that is more in keeping with the independence of agents in economic systems. This model was developed in the field of speech recognition, where many computational processes interact. The blackboard model that is developed and implemented for modelling distributed economic systems in this research is described in Chapter V. Briefly, the model is composed of a shared area of computer memory that is called the blackboard. In this model, economic agents operate as independent programs which have equal access to all shared data. The agents have roles as producers and consumers in the economic market, and are free to use any strategy to make decisions. An important feature of the blackboard market is the way time is handled so that both price and quantity can emerge from interactions in the market.

The blackboard model is an alternative conceptual model of distributed computing that is consistent with the distributed and independent nature of economic systems. Blackboard models are discussed and put forth as an example of a distributed economic computing in Chapter V. Using the blackboard model as the system-level view of an economy, agents are independent and can interact according to the economic forces of scarcity and trade. An artificial economic system was constructed from independent computer programs that use satisficing, optimizing and Stackelberg strategies for trading scarce resources under time constraints. The optimizing strategy was implemented in the form of a neural agent that learned and adapted to changing market conditions.

Chapter VI details the preliminary experiments and verification of the blackboard market system. The blackboard model was tested and verified with several well known economics problems. Agents employing static and adaptive expectations were tested, as well as more complex agents using a Stackelberg strategy. Two approaches were taken for the market verification, the first used a cobweb framework to test the market infrastructure and general operation of inputs and outputs. The second approach used Stackelberg agents with different numbers of producers and consumers to find an appropriate level of complexity to work with, and to verify that the theoretically expected results emerge. Without exception, these markets came to the theoretically expected equilibrium.

The distributed market experiments are reported in Chapter VII. The emerging prices, quantities and wealth from experimental markets composed of selected agents were examined, and the impact of different classes of agents are determined. In general, the researcher is free to construct markets composed of any number of agents that follow selected strategies, and are subject to given constraints.

One of the strengths of these markets is that the researcher can directly test the applicability of different strategies and assumptions about the market. In these experiments, the focus is on agents that use different assumptions and knowledge to interact in the market. Specifically, agents use one of three classes of behavior to make production and price decisions:

1. Satisficing agents use any approach that is good enough to satisfy their private criteria.

2. Optimizing neural agents learn about the market, and attempt to maximize profits by adjusting their output and prices to profit from the market conditions.

3. Stackelberg agents have knowledge of the average demand curve, but do not know the intentions of other producers in the market.

These agents interact with each other, and with consumers, in both homogeneous and heterogeneous markets (e.g. composed of a single agent or many agents of different types).

Chapter VIII discusses the analysis of economic data with neural networks. Several computational economics problems are solved with neural networks, and compared with more traditional methods when possible. In general, the successful application of neural methods depends on the problem complexity, and the availability of data. Neural methods are recommended when problems are complex and the data is messy. Neural methods are not recommended when the problem is simple, or when the internal structure of the solution and relationships among the variables must be recovered.

While others have used neural nets for solving economic problems, no approach up to this time (known to the author) has been rooted in the idea of systems theory nor related to the actual activities that take place in economic systems. The application of neural networks in economics generally are in predicting bond ratings, bank failures and other classification problems, [Dutta & Shekhar, 1988, 1989], [Surkan & Singleton, 1990], [Odom & Sarda, 1990], trading stock, currency and commodities, [Kimoto et. al., 1990], [Kamijo and Taniya, 1990], [Bergson & Wunsch, 1991], [Weigand et.al. 1991], and other forecasting applications [Hoptroff and Hall, 1991].

Some of the published neural network research in the economics literature claims advantages for neural models that appear to be overstated or even incorrect. For example, Dutta and Shekhar claim advantages for neural networks over linear regression when applied to predicting bond ratings, but overlook the basic similarity between these methods. There are two very different potential uses of neural networks for solving problems in the field of economics. The first is to apply the connectionist network purely as a computational device, without giving any meaning to the network structure. This is the usual approach found in the neural network literature. The second approach, which I proposed and rejected for modeling economic systems in this research, is to assign an economic meaning to elements within the network. Neural nets were successfully used to model the behavior of economic agents, and to solve several computational problems in economics.

Researchers in neural nets have done a good deal of high quality theoretical work, and have linked neural nets to stochastic estimators that are sometimes used to estimate economic phenomenon [Hornick et. al., 1990], [White, 1989]. It is also noteworthy that none of the reported research is based on a concept of the economy as a distributed computer. Rather, the work of White and associates emphasizes the network as a stochastic estimator of an unknown function. The network structure proposed by White is based, appropriately, on a vision of a computing, rather than an economic, system.

The conclusions resulting from this research are reported and summarized in Chapter IX.

CHAPTER II

SYSTEMS APPROACH TO ECONOMICS

INTRODUCTION

This chapter lays the theoretical foundation on which the empirical research is based. In the first section, systems are defined in a way that is consistent with the ideas and vocabulary of systems science. This section relies heavily on the definition of a system that was proposed by Lendaris [Lendaris, 1986], and in subsequent sections, I adapt and build on this definition in the context of economic decision making. The characteristics of economics that are relevant to the systems approach are also described in this chapter. Furthermore, the characteristics of economic phenomena and of systems are brought together to define economic systems. The second section of this chapter describes how the definition of economic systems can be used to help understand them as distributed computers.

Definition of Systems

Recent definitions of systems found in the literature insist that systems are more than a set of elements, attributes and relations. Modern definitions include emphasis on the importance of individual perception in the fundamental definition of a system. Lendaris, for example [Lendaris, 1986], gives the following two-part definition of a system.

"A system is:

A. A unit with certain attributes perceived relative to its external environment, and

B. That unit has the quality that it internally contains subunits and those subunits operate together to manifest the perceived attributes of the unit."

This definition specifies the requisite features of any system. The definition consists of two parts that explicitly indicate the simultaneous existence of macro and micro components in the system. First, a system can be perceived at the level of a unit within its environment, and second, as a collection of subunits within the unit. The levels exist simultaneously, yet can be observed separately. Furthermore, the definition requires that the attributes of the system are perceived relative to the external environment by an observer, thus specifying the relationship of the system to its environment, and insisting on the existence of an observer. The observer's focus (relationship to the system and the environment) provides the context against which the unit is perceived by the observer. The environment does not consist of the internal elements of the system, but rather as the background against which the system is observed.

As mentioned above, the word "perceived" indicates that a human is present to observe the system. Thus, the human element of perception exists in the primary definition of a system. Perception is inherently a subjective experience that relies on the observer. As a consequence of the subjective nature of perception, no system is perceived with complete objectivity. Alternative perspectives may provide radically different understandings of the same "system".

The second level of this definition provides for the internal subunits that compose the system. Again, the reference to perception internalizes the observer in the definition of the system. The subunits operate together to manifest the perceived attributes of the unit. The attributes of the system are not simply the sum of the attributes of the subunits. Instead, the attributes of the unit emerge from the interaction of the subunits. Decomposition is the usual method of scientific analysis of physical phenomenon. However, not all systems are decomposable, and if the subunits are strongly coupled [Simon, 1981], serious errors are introduced by decomposition without the proper redefinition of the environment, system and subsystem relationships.

When taken together, this definition requires perception at three adjacent levels: the environment, the unit and the subunit. Statement A requires perception of the environment and the unit, while statement B specifies perception of the unit and subunits. The statements overlap by requiring perception at the unit level.

Figure 1 depicts a two dimensional representation of the multiple views feasible for any system. It is possible for the perspective of the observer to shift up and down the vertical axis. This shifting corresponds to changing the focus of the observer, and consequently the unit or system level will become the environment of the new system. The subunit of the previous level will become the unit level from the new focus, and the new units must contain subunits to satisfy the definition of a system from the new focus. Figure 1 shows how the same object may be interpreted at the environment, system and subsystem levels, depending on the focus of the observer. At the same time, shifting along the horizontal axis provides a whole new perspective of the system. The horizontal axis indicates the perspectives that emerge when the observer adopts different roles, while the vertical axis represents the relationship of the environment to the unit and the subunits.

	Environment		
Focus 1	System 1	Environment	
Focus 2	Subsystem	System 2	Environment
Focus 3		Subsystem	System 3
			Subsystem
	Perspective 1	Perspective 2	Perspective 3

Figure 1. Multiple views of a system. (Adapted from [Lendaris, 1986].)

ECONOMIC SYSTEMS

Existing Definitions

Nearly every author who writes about economics in a general way devotes a paragraph or two to the concept of an economic system. However, there is no widespread agreement among the definitions of different authors, and none of the definitions contain all the elements required for an adequate definition from the systems perspective.

Many authors rely on the basic definition attributed to Robbins that economics is the study of scarcity, and economic systems are thus seen as systems in which scarcity plays a major role in allocation of resources [Robbins, 1935, p16]. Some authors, on the other hand, define economic systems in terms of the behavior that is attributed to economic agents. From this approach, economic systems are analyzed by methods that presuppose optimizing behavior [Intriligator, 1971].

In "Microeconomics as an Experimental Science", [Smith, 1982], Smith defines microeconomic systems in terms of "an environment and an institution". By environment he means the economic actors in the system and by institution he means the control mechanism of the system. While Smith's definition is appropriate for his purposes, it does not address the basic nature of a system, and is distinctly different in character from the concept of a system that prevails in the systems literature. Some of the early writers in the systems field were economists, such as Kenneth Boulding and Herbert Simon, who wrote extensively about economic systems. In Economics as a Science [Boulding, 1970], for instance, Boulding discusses several perceptual viewpoints of economics. While his overall outlook is in many ways compatible with modern systems views, he still does not clearly define economic systems. Boulding does however, reject the view that economics is founded on the premise of scarcity in favor of the concept that trade is the central feature of economic systems. It appears that Boulding was particularly interested in isomorphisms between ecology and economics [Boulding, 1981], and drew heavily on these relationships. Economics systems, however, are unique and can be studied as distributed systems in their own right. While isomorphisms exist between other competitive or distributed systems such as neural or ecological systems, economic systems appear to operate on other principles. In particular, economic systems are made up of self directed decision making units that deal with allocation of scarce resources by trading among themselves.

The writings of Herbert Simon are somewhat closer to the mark. Simon specifically points out the dual nature of economic systems as distributed and hierarchic [Simon, 1979, 1981, 1982]. However, he does not provide a definition of an economic system which is consistent with both of these views. Indeed, Simon distinguishes between the distributed nature of a free economy and the hierarchic nature of the firm. Simon was particularly interested in how decisions are made, and rejected the optimization hypothesis. Simon proposed that humans actually use very simple decision making rules rather than complex optimization procedures.

A recent collection of articles published as Systems Economics [Fox & Miles, 1987], discusses economics from a systems viewpoint, but does not provide an overall frame for visualizing economic problems. It is also clear from a comparison of the works of Boulding, Simon and Fox & Miles that there is not complete agreement within the systems field about the concept of an economic system.

Economics is the study of decision making problems that include scarcity as a fundamental consideration. Problems involving efficiency, alternative uses of resources, and trade are typical economic problems. Robbins [1935, p16] defined economics as "the science which studies human behavior as a relationship between ends and scarce means which have alternative uses." Simon [1981, p31] also maintained that the fundamental concern of economics is the allocation of scarce resources:

"Scarcity is a central fact of life. Because resources - land, money, fuel, time, attention- are scarce in relation to our uses of them, it is a task of rationality to allocate them. The discipline of economics has taken the performance of that task as its focal concern."

Both of these economists draw attention to the fact that economics is a discipline that studies problems posed by the scarcity of resources and the surplus of competing applications. Generally, economists do not study the actual process by which decisions are made, but the outcomes of such processes. It is widely believed in the discipline that economic agents are rational, and that economic decision processes are guided by optimization. That is, economic agents compare the feasible alternatives, and employ the combination of resources that is preferred above all others, given the associated constraints and benefits. An alternative hypothesis is based on the concept of bounded rationality, wherein the economic agents satisfice rather than optimize. The present research studies the decision making process, and documents the consequences of different decision making procedures.

In a somewhat different approach, [Boulding, 1970] Boulding emphasized the importance of one particular type of activity in the economy: "The economy consists of that segment of the sociosphere which is organized through exchange, and especially commodity exchange." In his conception then, exchange is the quintessence of economic activity, and all other economic actions are related to exchange. Furthermore, like conversation, exchange is a behavior that is typically human, while scarcity is a phenomenon that is ubiquitous in the animal world.

Economic decision processes have been defined in relation to allocation of scarce resources between alternative uses. The role of decision maker is taken up by some entity such as a country, market, institution, firm, household, individual, etc. Economic decision units are sometimes called economic agents in the present work. The activities that these economic decision units are engaged in will provide the context in which the decision process is embedded. Some activities that are frequently mentioned in economics include production, distribution, trade and consumption.

Recognizing the importance of exchange is crucial for the development of the concept of economic systems that follows because economic agents do not operate independently. (With the possible exception of Robinson Cruso). Rather, the decisions of one unit may affect the decisions of others. Motives for conflict and cooperation frequently arise between economic entities. We are interested not only in the decisions of a single unit, but also in the relationships of the complex of economic units.

Economic Systems

An economic system exists in some environment or context that is defined by the observer. The system is perceived by some observer who can distinguish the system from the environment. There is no reason that the observer cannot play a role in the economic system. The system has attributes that are not shared by the environment and is therefore distinguishable from the environment by the observer.

The attributes of an economic system that can be perceived by the observer are distinguishable from the attributes of the goods and services that the economic system deals with. Economic variables such as perceived quality are attributes of goods and services, rather than attributes of an economic system in which such goods are traded. The attributes of an economic system are embodied in the relative prices and quantities of goods produced, and in the way the system adjusts to changes in the economic environment.

The price of a good emerges from the interaction of buyers and sellers who come together to trade in an economic market. Exactly how the price emerges is not known with certainty, but a number of models have been suggested over the years. The discussion herein will explore some of the better known options and discuss the implications of price emergence under various models using a systems approach. Price is sometimes called the "informational link" because it summarizes all of the information that is available to participants in the market. The price emerges from interaction in the market and is used by economic agents as a basis for economic decision making, which in turn affects the price.

The subunits in an economic system are decision making units. The decision units may share the same information but not all arrive at exactly the same decision. In other words, the decision units are not identical and do not necessarily weigh all the information in the same manner. As a result, not all decision units come to precisely the same economic decisions.

As mentioned above, an economic system is composed of decision making units which are the subunits in the system. Decision making units may be simple or complex (e.g., made up of many subunits), but in either case each unit must make decisions based on the rules, constraints, and data. In this model, the decision of each unit is some function of weighted input data. The decisions of these units are not necessarily independent, and in some cases the units are competing for the same scarce resources.

Most economic systems are open, e.g. the participants in the system can enter and leave the system at will, but the system remains intact. Furthermore, the decision-making units are not physically connected, but rather are connected only by the information that they share. Attributes of economic systems that may be of interest are price and substitution elasticities, changes in prices and incomes, and efficiency in transformation of resources.

Consider an economic system in the context of production decision making. In this case, the inputs into the system are resource prices and the total quantity of output desired. The percentage of the budget expended on each input (budget share), and the total cost of production emerge as the system operates. The problem that is given to the production system is to allocate resources given the resource price and the desired quantity of finished goods to be produced. The decision making units must choose the appropriate quantities of production factors so that the production facility is capable of utilizing the factors to produce the corresponding output. Of course, the expenditure on factors of production must be less than the total value of the output in order for the producer to earn a profit. At the level of this discussion, the internal decision units could represent alternative firms or production decision making units. In my approach, the production technology is represented internally within the system and is not specified by the modeler. In addition, decision rules such as cost minimization or profit maximization are included implicitly in the system if they exist.

A Systems Approach Applied to Economic Markets

Application of the systems approach to economic markets is straightforward. A market exists within some larger economic and social environment. A market is a trading arena in which buyers and sellers come together to trade specific goods and services. In this context, the market is the principal unit of study and is consistent with the definition of an economic system as a system wherein scarce resources are allocated by trade among independent agents. It is composed of agents that can be identified by the roles that the sub-units take, such as buyers and sellers. The buyers and sellers operate together in a manner that manifests the attributes of the market.

The variables that emerge from trading are the price and quantity of goods that are traded, the rate and timing of trades, patterns of trade and so forth. These attributes cannot be determined by studying the buyers and sellers independently, but must emerge or unfold over a period of time as interactions take place. Certain markets may have attributes that are peculiar to themselves, or may depend on the particular sub-units that participate in the market. Fundamental characteristics of markets are determined by the control system or market regulator. In economic terms, the market regulator is known as the "institution". In some cases, the institution may participate in the market as a subunit that filters all the interactions between buyers and sellers.

The researcher has a role as a meta-level observer who does not participate in the workings of the market, but is able to view the interactions at meta-system, system and subsystem levels as necessary. For instance, taking only a system view would be analogous to learning how a radio works by observing the radio's inputs and outputs. A reductionist approach, on the other hand, would study the independent workings of the individual sub-units, without ever gaining an understanding of the joint interactions among the sub-units. A meta-system view permits a further change in scope that enables examination of the system as it interacts with other systems in its environment. The systems approach requires an observer that is cognizant of each of the meta- system, system and sub-system levels, as well as the various roles that can be assumed within each of these levels.

An economic system such as a market is composed of "smart" agents that can learn and change their behavior. This is a crucial observation because the subunits in a market, and economic agents in general, have the ability to study the market in the same way as the researcher. That is, the sub-units in a market can conceptually take any role and are capable of "stepping outside the system" and, studying the market in the same manner as the researcher. This allows the subunits to form strategies based on knowledge gained by studying the market and the individuals in the market from an external vantage point.

However, it is vitally important to avoid the pitfall of assuming too much about the abilities of the market and the economic agents within the market. In general, I assume that economic agents are capable of learning from experience and adapting to new situations including changing market conditions. These are fundamental properties of the sub-units in the market system. Of course it would be desirable for a model of a market to possess the same properties as the genuine market at the system and sub-system levels. But a model is intended to be an abstraction and simplification of reality, not a duplicate it.

The relationships between economic systems, economic agents, and the goods and services that are traded in economic systems by economic agents may be less than clear at this juncture. The following discussion is included to illuminate these relationships.

Attributes of an Economic System. The economic system is perceived (by the researcher) as a unit relative to its external environment. An economic system can be distinguished from its environment because it has attributes that the environment does not share. The attributes that the meta- level observer perceives relative to the environment are that the economic system has inputs, outputs and contains sub-units. The observer may also note some pattern or hierarchical relationship among the subunits. The inputs and outputs from the system are economic variables and are sometimes known as instruments in economic analysis. The sub-units will be called economic agents or just agents in this discussion.

A distinguishing characteristic of all economic systems is the presence of trade between the subunits. This feature distinguishes economic systems from other types of social, biological and physical systems where exchange of goods and services is not the focus.

Emergent Properties. The outputs that emerge from the economic system may not be predictable from the characteristics of the agents comprising the system. Such qualities are defined as emergent properties. In the case of economic systems, the prices and quantities of goods exchanged are important emergent properties of the system. Money emerges as a good that facilitates exchange in all but the most primitive economic systems. The values of other economic variables including interest and growth rates are system properties, not properties of goods, services or agents. They emerge from the interaction of choices of economic agents.

The Sub-Units of an Economic System. The sub-units of an economic system are economic agents that operate together to manifest the perceived attributes of the economic system. In microeconomics, the sub-units of an economic systems have traditionally been defined as consumers and firms, however, other types of economic entities, such as households, governments, labor unions, churches and schools exist and are recognized.

Economic entities such as firms, households and governments also contain subunits whose interactions manifest the perceived attributes of the entity. The subunits within economic entities have multiple roles, because each is part of some larger entity and is also an atomistic consumer. Thus, many economic agents can be perceived as economic systems in their own right.

A firm, for example, can also be understood as an economic system by appropriately defining the environment in which it resides, observing that the firm has attributes relative to its external environment, and identifying the subunits that operate together to manifest the attributes of the firm. The employees of the firm usually exchange labor for money and other benefits. Additionally, firms are typically organized in a hierarchical fashion, which is one of the attributes of a firm that distinguishes it from other types of economic systems. Firms may also have other types of subunits that are organized according to function, such as accounting, production, administration, transmission and distribution. Of course, large corporations may own many firms that interact in complex ways.

Economic Activities. Three economic activities are generally recognized; production, consumption and trade. These are activities that economic agents engage in, and make decisions about. To these fundamental activities I would add control, as the function of governmental entities. These activities are interdependent because the completion of each activity depends on the action of other agents, and as mentioned above, the agents may have multiple roles in production and consumption. To complete any activity, the agents must trade, and therefore, trade is taken as the basic underlying economic activity in which all economic agents engage. Uncoerced trade is mutually beneficial to the participants because each exchanges for something that they deem to be of greater or equal value.

Competitive Economic Systems. It is not my purpose here to analyze all types of economic systems that can exist in the economic environment. Instead, I am interested in the common factors shared by what can be recognized as competitive economic systems. A firm is a cooperative economic system that enlists individuals by trading money and other benefits for labor. There are some elements of conflict between the goals of the sub-units and the goals of the firm. However, the firm is cooperative in the sense that the sub-units are not in direct competition with each other, but have, at least to some degree, a common goal. The organizing principle of the firm is cooperative. On the other hand, conflict and competition are inherent in market economic systems, yet trades are usually mutually beneficial.

Competition can exist in at least three ways in market economic systems. There can be competition between producers for market share and sales. At the same time, consumers compete among themselves for scarce goods and services. There is also a conflict between consumers and producers over the terms of trade. Although the exchange between the consumer and producer is mutually beneficial, the gains from trade are not necessarily divided evenly between them.

<u>Motives of Economic Agents</u>. One motivation behind trade is the prospect of increased consumption of goods and services. The primary motivation behind production is to produce a good or service that can be exchanged for more desirable goods and services. The sub-units in production systems trade their skills and time for the ability to consume. However, other personal and social factors may influence trade.

In classical economic analysis, the basic operating principle ascribed to economic agents is constrained optimization. Consequently, the mathematical tools of constrained optimization are used for analyzing economic problems. In modeling economic behavior, classical analysis assumes that the economic agents have objectives and constraints. Firms are assumed to maximize profits or minimize costs subject to the constraint of their production function. On the other hand, consumers are assumed to maximize utility subject to the constraint given by their income. In the first case, the producer has a well defined objective function (cost), but a poorly defined constraint (a production function or technology constraint), while in the second case, the consumer has a poorly defined objective function (utility) but a well defined constraint (income).

Unfortunately, the optimization paradigm begins to become very complex when the producer must maximize profits based on what the producer believes all other producers and consumers plan for the future. (The same comment holds for consumers). For this reason, strategies other than direct optimization are often encountered in practice. While formal optimization may lead to interesting insights about economic outcomes, few believe that such processes are used by individuals to solve every-day economic problems.

Interpretation of an Economy as a Distributed Computer

The above definition of an economic system leads naturally into the interpretation of the economy as a distributed computer. By a distributed computer I mean a computer wherein both computation and storage of knowledge are spread throughout the system. In a free economy, each individual operates in his own self interest. Operation is local but the combined decisions of all the participants of the economy is in some sense optimal. The prices and quantities of goods available in the economy emerge from such a system without external coordination or control. The correct quantities of ball bearings and oranges are produced at the prices corresponding to their relative scarcity and by virtue of the productivity or quality of the resources. No omniscient individual computes how much of each quantity to produce and the prices to set for each item. Instead, these prices and quantities emerge from the millions of interactions between buyers and sellers in the markets.

Buyers and sellers are the basic decision making units of such a system. Messages consisting of potential exchange prices pass between them, however, some information may be held privately by the individual decision making units and is not shared. Many economic systems also include some type of institution or control system. Patterns of exchange are formed in an economy over time. However, search is an important part of economic exchange. Learning by the economic units can be understood as learning, not just of the individual agents, but of the whole economic system. A type of self organization takes place where entities specialize and differentiation takes place. As this process continues, the driving force is at the decision making level, and results in a change in organizational structure of the market. The relative price of a good in this system is the informational link that summarizes the good's status in the system. If each individual begins with a given allocation of wealth, the system will change through a series of local trades between the economic units until the system comes to equilibrium. In the present research, the exchange prices, quantities and stored wealth are all computed by individual units in the system. Each unit acts with local knowledge depending on the messages that are passed to it from other agents. There is no stipulation that all agents are directly inter-connected or have perfect knowledge. Instead, each element receives input signals or messages from many, possibly thousands, of other agents and in the same manner sends signals to others that are dependent on the signals received as well as any private information that is available to the unit. In the model presented here, the economic agent weights the inputs and sends a signal that is some function of the weighted inputs.

Distributed computers constructed of silicon chips are a comparatively recent development when compared to distributed economic systems. However, the ability to build and simulate distributed computers leads to the possibility of simulating an economy by actually constructing a distributed model of an economic system and allowing the agents to interact according to any principles the experimenter chooses. Thus the experimenter has control over a broad array of assumptions.

Economic systems can be understood as a type of parallel distributed computer. That is, a computer made up of a large number of computing elements all operating at the same time, with little (if any) central control. A free market economy comprises a massive number of activities that proceed simultaneously (massively parallel), and has little or no central control. However, as mentioned by Simon [Simon, 1981, 1982], both hierarchical and distributed aspects are present in any economy.

The concept of the economy as a distributed computer is consistent with the idea of general equilibrium in an economy. Assuming that each individual in an economic system acts according to utility maximization, then the economy does indeed act as a distributed computer, since there is no central processing unit that computes the prices or the quantities of goods in the economy at the macro level. However, some centralized control is apparent in business firms, governments and markets. On closer inspection, this central control can be seen as a feature that emerges in certain types of economic systems.

Herbert Simon appears to be the first modern writer to identify economic systems as distributed computers. Earlier writers observed that system-wide economic prosperity depends on the decisions and interaction of individuals, but did not have the modern computer model to work with. The alternative to distributed economic systems are systems that exhibit central control, which removes the decision making from the individual. Both hierarchic and distributed economic systems are observed in the world, and it appears that most economic systems embody some combination of control.

The significant points can be summarized as follows:

1. An economic system is competitive at some level, so that conflict is inherent in the system.

2. Trade takes place when it is mutually beneficial to the interacting participants.

3. Some of the criteria for trade are not observable by the participants or by the researcher.

The researcher is dealing with imperfect information about the agents and cannot predict in advance what properties will emerge from the interaction of the agents. However, according the systems paradigm, even if the researcher had perfect information about the agents, there are some emergent properties from the interaction of the agents that cannot be predicted solely from the known attributes of the agents.

As mentioned earlier, the traditional approach to answering questions in economics relies on aggregation to reduce problem complexity. Unfortunately, this aggregation masks the variability that is inherent in individuals in the market, and reduces the observed diversity and complexity of the world. The research reported here looks for what emerges from a disaggregated approach that is not forthcoming in the usual collective approach. Additionally, the approach by which the system comes to equilibrium under various assumptions about how the independent agents behave can be studied directly with this technique. The behavior of agents may be frustrated and the agents may not be able to satisfy all their needs.

DISCUSSION

It is proposed that some type of group or system learning is demonstrated by economic systems. At the start of such an operation, prices are not known to any of the participants of the system, but after trading for a short time, all the participants come to some common knowledge or consensus about the prices and quantities of traded goods. The system has learned the prices, quantities, relative efficiency and scarcity of goods that are exchanged. Note that data about available resources or wealth are introduced from the external environment and the physical resources remain outside the system. Data about changes and transfers of resources leave the system to the external environment. The system learns in an unsupervised way about the allocation of goods and about the relative prices of the goods.

Not all economic systems need adhere to the framework described above. An economic production system, for example, can be observed by isolating a subunit of a market system that specializes in economic production processes. Inputs into such a system are the prices that arrive from its external environment. This data is used by the producers to allocate their expenditures among the factors of production. Again, the system is distributed because knowledge of the technology and relative scarcity of the goods is distributed across the network. In addition, the computation of needs for each unit is done locally without any central control, but the final result depends on all the producers and how they interact.

An interesting aspect of this conception of a system is that the decision making units can have multiple roles. The decision making unit may act as a consumer of goods, but also as a factor of production. The decision units may be buyers in one market but sellers in another market. Consequently, defining the perspective of the observer is critical.

Economic systems are open in the sense that individuals may enter and leave the system at will. The economy adjusts dynamically when economic actors enter and leave the system. In addition to the distributed computation of the relative prices and quantities in economic systems, the market structure is also computed. By market structure, I mean the number and type of decision making units, and how they are employed.

Distributed economic systems may provide some interesting insights regarding market structure and unemployment. For instance does efficiency in a market gradually degrade as the number of decision making units falls? In addition, what does this distributed computational model say about monopoly theory and the tendency of a single producer to dominate some types of markets while other markets are made up of many small buyers and sellers? Distributed computational models are a novel way to study questions about market structure. Not only are price and quantity computed by the market, but also the ultimate number of decision making units in the market. New subunits enter or leave the system when certain conditions exist in the system.

The place of institutions and governments is important in distributed models. Typically, distributed economic systems contain buyers and sellers, as well as control units that permit only certain types of transactions and punish inappropriate behavior. However, the institution is not created by a greater power, but is an instance of self organization and control.

CHAPTER III

MODELS OF ECONOMIC SYSTEMS

Classical economic models are based on the concept of optimization in a perfect economic market, as well as assumptions about the motives of the individual. The underlying model assumes that buyers and sellers are rational and are optimizers. That is, consumers maximize utility or preferences, while producers minimize costs (a somewhat weaker assumption than profit maximization). The assumption of rationality is critical for these models and gives a guide for interpreting the models. A second critical assumption is made regarding the expectations of consumers and producers in these models.

All the models that will be considered here have a one period lag between production and consumption of goods. In other words, the production decision is made in the period prior to the period in which the consumption decision is made. The total quantity of goods available in period t+1 is determined in period t, based on the price that is expected to emerge in period t+1. Consumption occurs in period t+1 based on the fixed quantity of goods that are available. The price emerges in period t+1 from the interaction of buyers who bid for the fixed quantity of goods. When supply of the good exceeds demand, the price falls relative to the price of the previous period. Conversely, when demand exceeds supply, the price increases.

THE ROLE OF EXPECTATIONS

The quantity of goods available in period t+1 is set by each producer in period t based on the expected price in the next period. Three models of price expectations have been widely discussed in the economics literature. These models are static expectations, adaptive expectations, and rational expectations. In the linear form the model can be set up as follows:

Supply: $Q^s = a + bP^e$

Demand: $Q^d = c - dP$

Demand = Supply: $Q^s = Q^d$

In these equations Q represents the quantity of goods available in period t+1, and P^e is the expected price for the t+1 period. The supply computation takes place in period t so the actual price in period t+1 is not known until it emerges. The price that actually emerges in period t+1 is denoted P. The stability of the linear model can be determined from the coefficients, a,b,c,d. A more general formulation drops the linearity assumption and just states:

Supply: $Q^s = f(P^e)$

Demand: $Q^s = g(P)$

where f and g represent the supply and demand relations. An assumption that is usually made in these models is that the supplier knows the supply relation. The supplier then makes plans for the next period based on the expected price. Exactly how the price expectation is formed is not known with certainty, however, many models have been proposed. Most models propose a subjective estimate of price for the next period.

Simple supply and demand models do not allow speculation by the buyer or seller. Inventories do not carry over into the future, implying that all production in period t is sold and consumed in the same period.

Static Expectations

In the static expectations model, the producer expects the price in period t+1 to be the same as the price in period t. In another context, this is similar to predicting tomorrow's weather by assuming that it will be the same as today's weather. In economics, the model is frequently simplified by assuming linear supply and demand curves. This combination of assumptions results in predictable oscillations in price that can easily be observed by participants in the market if they have meta-observer capabilities, or even a short term memory. The linear model can be set up as follows:

Supply: $Q^s_t = a + bP_{t-1}$

Demand: $Q^s_t = c - dP_t$

The relative magnitude of the parameters a,b,c,d determine the stability of the model. In the static expectations model, the equilibrium price P^* is reached when $P_{t-1}=P_t=P^*$ and there is no further price adjustment. It can be shown that the equilibrium price is $P^* = (c-a)/(b+d)$, however, the economic agents do not discover this relationship. Furthermore, the price at any time t could be predicted with certainty if the parameters b and d are known, because

$P_t = [P_0 - P^*](-b/d)_t + P^*$. Again, the economic agents in the static expectations model not discover this price vs time relationship.

Note that the model as shown above is perfectly deterministic rather than stochastic. The model becomes stochastic by adding independent error terms to the supply and demand equations.

At a meta level, static expectations appear to be naive on the part of the producers because regular oscillations emerge, but the producers do not incorporate this information in their price expectations. Additionally, the producers do not attempt to solve the equations or reduce the errors in their forecasts.

The cobweb model from economics is a well known example of a static expectations model. This model is often explained in the context of pig farming and is called the hog cycle. In the hog cycle, the producers look only at the price that hogs are selling for in the current period. They plan production for the next period based on the market price of hogs in the current period. The time lag needed for breeding and raising hogs that are ready for market is one period in this model.

If the price in the next period does turn out being equal to the price in the previous period, then the hog farmer produced the correct quantity of hogs. However, any perturbation resulting from unexpected shocks such as weather variations could lead to the cyclical fluctuations described above. The (hypothetical) hog farmers never catch on to the cyclical price pattern and continue to follow the static expectations assumption.

An example of static expectations that may be closer to home in the academic community is the choice of majors by college students. When students enter college, they may survey the market and choose a major that is currently in high demand and consequently is paying a higher than average market wage. Large numbers of students may choose such a major, leading to a large number of graduates within a few years. By the time the students complete their college degree, the market has an excess supply of new graduates in the field and the new entrants to the field may not find jobs at the wages they expected. The news of a glut on the market and the relatively low wages will then discourage students from entering the field, which of course results in a shortage of qualified graduates within a few years and the cycle begins anew. Such cycles have been reported for several fields including engineering.

Adaptive Expectations

A somewhat less naive model of an economic market allows for learning by economic agents. Agents still form subjective expectations about future prices, but the error in previous price projections is taken into account. They calculate the price expectation for period t based on the error in previous expectations. The price expectation rule becomes:

$$P^e_t = P^e_{t-1} + h(P_{t-1} - P^e_{t-1}) \quad 0 <= h <= 1.$$

In the adaptive expectations model, the coefficient h is called the learning rate. As a result of applying this price adjustment rule, errors in forming the subjective estimation of price in the next period become increasingly smaller in each period. The producer learns at each round and the error rapidly approaches zero. The adjustment rule applies a weighted average of last periods price and the expected value of the price as follows: $P^e_t = hP_{t-1} + (1-h)P^e_{t-1}$

The adaptive expectations model is often demonstrated with linear supply and demand functions. The supply function is written: $Q^s_t = a + bP^e_{t-1}$, and the demand function is written: $Q^s_t = c - dP_t$. Under these conditions, the error approaches zero from above, in a manner similar to a geometric series. Again, the error at any time can be computed by the producers who know only the supply relation. What the supplier "learns" is how to adjust for the error in the price forecast of the last period. No overall "meta" learning is implied by this model, and the adaptive expectations model can be criticized for this reason. Each time there is a disturbance, the economic agents must learn again.

Econometrically, this type of model is estimated by including the previous quantity term in the model as follows: $Q_t = ah + bhP_{t-1} + (1-h)Q_{t-1} + u_t$.

It can be shown that this is equivalent to estimating $Q_t = a + bh\Sigma(1-h)_s P_{t-s} + u_t$ where the sum is over s=0 to ∞. However, the estimation is finite in the first case, and easily estimatable, while the second case has an infinite number of terms and is impossible to estimate directly. The result of the mathematical manipulations is that a new variable (Quantity in the t-1 period) is included as an independent variable in the supply equation.

Taking a meta level view of the adaptive expectations model results in a criticism similar to the criticism of the static expectations model. Although the economic agents adapt to errors in price estimation in the model, a decided improvement when compared with the static model, there remains a consistent bias in the errors. The economic agents do not learn to adjust for the bias, and as a result, there is the opportunity for economic agents to make a systematic profit by exploiting the predictability of such models. It is widely believed that economic agents in markets can learn and observe patterns at the meta level. Thus the problem is not to estimate the price in the next period, but the equilibrium price P^*. At the meta level it is supposed that if such patterns are observed, then the agents making the observations will change their behavior to take advantage of them.

It was the shortcomings of models similar to the adaptive expectations models that gave rise to a new generation of models in the 1960's. Two of the critics of theses models worked together in the 1960's at Carnegie Mellon University but came to very different conclusions about the capacity of humans to understand economic systems in which they participate. Herbert Simon and John Muth were part of the same working group that was studying economic decision making. Muth developed the idea of perfectly rational actors that operate as optimizers, while Simon developed the idea of bounded rationality, where economic actors make decisions that are not based on an optimizing paradigm at all.

Rational Expectations

Muth introduced the rational expectations approach with his paper "Rational Expectations and the Theory of Price Movements," [Muth, 1961]. Rational expectations models demand that the participants in economic markets are rational, and use all information available to them to make decisions about the future. In the rational expectations model, the producer would use all tools available, going so far as to discover the correct functional form and coefficients of the market supply and demand curves. In the case of the linear model, the producer could solve for the equilibrium price $P^* = (c-d)/(b+d)$. This information could then be used to form the price expectations:

$$Pe_t = P^* = (c-d) / (b+d).$$

This model is free of the predictable oscillations of the static and adaptive models. In fact, under the rational expectations model, the price reaches equilibrium instantly because the producers know that this is the optimum price, and they cannot improve their lot by forecasting any other price. The economic actors are able to observe the system and make meta level optimizing decisions. One of the implications of the rational expectations model is that the economic actors cannot use information in the market to "beat" the market because the same data is available to all. By the same token, the agents cannot rationally ignore useful data. The stochastic error term remains in the model resulting in small random errors that cannot be eliminated.

Note that the supply and demand models described here address only the behavior of the producer, while ignoring the decision making behavior of the consumer. Adding storage capacity for the demand side of the model would allow the consumers to take advantage of low prices by purchasing more than enough goods for consumption in the current period. My analysis is also focused on the producer's decision making process.

MARKET STRUCTURE

The number of agents in a market and their awareness of each other plays an important role in the production decision. Consider a market with only 2 producers (duopoly). The market price depends on the quantity that each producer brings to market in the period. If each producer is bound by a production function:

$$q^1 = f^1(x^1{}_1, x^1{}_2, \ldots, x^1{}_n)$$

$$q^2 = f^2(x^2{}_1, x^2{}_2, \ldots, x^2{}_n)$$

where q^1 and q^2 are the outputs of producers 1 and 2, respectively, f^1 and f^2 represent the respective production functions of the 2 producers, and $x^i{}_j$ is the level that the ith producer uses of the jth input resource. The price at time t is then some joint function of the quantity of goods produced by each producer in the previous period: $p_t = g(q^1,q^2)_{t-1}$

The price, and thus the profits of each producer, are affected by the production decision of its competitors. The input prices $w_j, \ldots, w_n$ are also jointly determined by the demand of the producers: $w_j = w(x^1{}_j, x^2{}_j)$, $j=1,2,\ldots,n$. Assuming that each firm is attempting to maximize profits (π), firm 1 may set the problem up as follows:

$$\max \pi = p(q^1,q^2)q^1 - \Sigma_j w_j(x^1{}_j, x^2{}_j)x^1{}_j \qquad (1)$$

with the production constraint:

$$q^1 = f(x_1, x_2, \ldots, x_2)$$

by choosing the output quantity q^1 and the input quantities $x^1{}_1, \ldots x^1{}_n$.

The n+1 first order conditions from the Lagrangian method are:

$$\left[p+q^1\left(\frac{\partial p}{\partial Q^1}+\frac{\partial p}{\partial q^2}\frac{\partial q^2}{\partial q^1}\right)\right]\frac{\partial f^1}{\partial x^1_j}=w_j+x^1_j\frac{\partial w_j}{\partial x^1_j}+\frac{\partial w_j}{\partial x^2_j}\frac{\partial x^2_j}{\partial x^1_j},\ j=1,2,\ldots,n \qquad (2)$$

$q^1=f^1(x^1{}_1,x^1{}_2,\ldots,x^1{}_n)$.

The key point is that each producer must take into account the other producer's actions in order to maximize profits. The term $\frac{\partial q^2}{\partial q^1}$ is especially important to this analysis because it indicates the change in the output of producer 2 as the first producer changes output. This term is known as the conjectural variation (CV), because it is subjectively arrived at by the producer. Various assumptions can be made by the producer about the reaction of the other producer. The most naive assumption is that the other producer will not react at all, in other words, the conjectural variation is 0 in Equation 2. Cournot was the first to analyze the case where both producers assume that the conjectural variation is 0. From the meta level, this assumption leads to predictable oscillation in output, much like the static expectations model. Contrary to the evidence, both producers continue to assume that the competitor is not making any adjustment to output.

The Stackelberg analysis allows for conjectural variations that are not zero. For instance, producer 1 may conjecture that producer 2 has assumed a 0 conjectural variation, and has adjusted production accordingly. If this set of assumptions is accurate, the result is a Stackelberg equilibrium wherein producer 1 receives a higher share of production and profits than producer 2. However, if both producers use the Stackelberg assumption, then the result is a Stackelberg disequilibrium, that is not Pareto optimal for the producers. The Stackelberg analysis in this case is identical to a 2 person non-zero sum game that yields a

Prisoners Dilemma. Figure 2 shows an example of a payoff matrix under various conjectural variations.

		Producer 2: Cournot Reaction	Producer 2: Stackelberg Reaction
Producer 1	Cournot Reaction	Cournot Equilibrium	Stackelberg Equilibrium For Producer 2
	Stackelberg Reaction	Stackelberg Equilibrium For Producer 1	Stackelberg Disequilibrium

Figure 2. Payoff matrix for Stackelberg agents.

The Stackelberg analysis is not limited to the duopoly case, because more producers can be added to the market without affecting the outcome appreciably. However, as the number of producers approaches infinity, the perfect competition price and quantity solution emerges. One of the insights resulting from the Stackelberg analysis is that there may not be a clear optimization solution available to the producers, even if they are capable of forming and solving Equations 1 and 2.

BOUNDED RATIONALITY

Herbert Simon rejected the optimizing model of economic behavior that was (is) widely accepted in economics. In its place, Simon developed models that are based on a concept that he called satisficing. Simon coined the word "satisfice" to indicate a strategy wherein economic agents choose any solution that is "good enough" to satisfy their subjective criteria. The central idea is that economic agents are not capable of performing the computations that are required for optimization. In fact, computers and non-linear programming techniques that are required to solve such problems have only recently become available, but economic markets have existed for thousands of years. Furthermore, according to Simon, individuals do not even attempt to optimize but are satisfied by any solution that meets their subjective criteria. An interesting question is whether markets in which computers are available to the participants perform in ways that are significantly different from markets where computers are not present. Even when computers are available, it is not apparent that all participants would use optimizing procedures. It is not that economic agents are purposely irrational. On the contrary, non-optimal solutions to real economic problems may occur because people simply do not base all economic decisions on optimizing techniques. For instance, futures traders use a variety of techniques, ranging from fundamental analysis to scalping (trading on microscopic fluctuations in prices).

Simon proposed that real economic agents use heuristic strategies for problem solving and to simplify economic decision making. For instance, a knowledge based technique that relies on discovering a relationship in historical data could be used to form expectations about future demand in the market. Actual market behavior varies considerably, with different actors favoring their own proprietary approach to forming expectations. Such heuristic strategies can also use meta level observations about market behavior in forming expectations about future market behavior.

FRACTALS AND CHAOS IN ECONOMIC SYSTEMS

In the early 1960's, Mandelbrot pointed out that many distributions of economic data are not necessarily Gaussian [Mandelbrot, 1961, 1963a, 1963b]. Distribution of prices, income and other economicfquantities are often skewed rather than centrally distributed. Furthermore, it is not uncommon for economic data to contain outliers in the distribution tails. Mandelbrot's contention is that economists overlook these details and continue to assume that their data is Normally distributed (Gaussian), even though the evidence of the data sometimes rejects the Normality hypothesis. Mandelbrot proposed the use of the Paretian distribution as an alternative to the Gaussian. The Paretian and its relatives, such as the stable distributions of Levy agree better with the empirical evidence of economic data.

Mandelbrot's study of corn prices is of special interest. First, Mandelbrot points out that in reality, prices are not continuous, as is usually assumed. Large changes in prices occur frequently in speculative markets. This is common knowledge among market analysts and traders. (For instance see [Commodities Trading Handbook, Chicago Board of Trade, 1990]). The evidence does not suggest a smooth or even continuous price generating function in real markets. Where such constraints are imposed by economists when choosing a functional form, the departure from reality may be stunning. It is precisely these large and sudden price movements that market participants would like to forecast, but are ignored by theory.

Second, Mandelbrot shows that price movements follow a generating process that is consistent, regardless of the time scale of measurement. (e.g., the generating function is the same, regardless of whether the prices are monitored on a daily, weekly, monthly, or annual basis). Further the generating process showed only minor changes in over 100 years of monitoring, indicating that the underlying process is very stable.

Mandelbrot goes on to name the Paretian exponent the "fractal dimension" (also known as the Hausdorff dimension), and links this approach to a wide variety of physical phenomenon. Since that time, a good deal has been written about deterministic price formation that links the fractal dimension with chaos. In the late 1980's several researchers reported finding evidence of deterministic relations in financial market prices [Brock, 1988], [Scheinkman & Le Baron, 1990].

The introduction of non-linear methods into economic models acknowledges that actual supply and demand functions may be non- linear or discontinuous. But an even more important contribution is the notion of determinism in economic systems. That is, under these models, economic agents may follow a deterministic, rule based approach, that incorporates discontinuous or non-linear adjustments. With the exception of Simon's model, the traditional economic models mentioned above do not allow for these features.

MODELLING DISTRIBUTED ECONOMIC SYSTEMS

The usual methods used for modeling economic problems are based on a theoretical or top down approach. That is, the researcher is guided by existing notions or theories about the economy, usually based on the concept of the human being as an optimizer. A refinement of this approach is the axiomatic work of Lin and Perry [Lin and Perry,1988, 1989]. In these works, economic knowledge about the world is stated explicitly as a set of axioms which are expressed in PROLOG, a logical programming language. Data observed in the economic world can be tested directly for consistency with the axioms of economic theory. Furthermore, the axioms and the observed data can be used together to forecast economic behavior with a logic program. In the axiomatic approach, knowledge of economic behavior is encoded in a logic program. However, in the research reported here, a variety of economic behaviors are feasible; it is the interaction of these behaviors that manifest the system level attributes.

The main goal of my research is to determine how different types of agents affect the emerging price, quantity, number of agents, and stability in a distributed economic market. The distributed market simulation is composed of independent computer programs that act as producers and consumers. The market is distributed in the sense that the programs represent agents that make decisions independent of any central control.

The agents are programmed as either producers or consumers of some imaginary good. The research concentrates on the producer's decision making process. Three general types of strategies are incorporated in the agent's decision making process: optimization, satisficing and Stackelberg.

Optimizing Agents

These agents maximize expected profits in the next period based on historical information. Two optimizing programs are constructed, but multiple instances of each program may operate simultaneously. These agents use neural networks to recognize patterns and make decisions. The networks apply a hill climbing (optimizing) algorithm known as backpropagation to adjust weights in the network. Taken together, the pattern of weights summarize the knowledge gained in past experience, but do not easily reveal the explicit knowledge of the rule based approach.

The actual outcome is observed and is compared to the expected outcome. The weight adjustments are based on the difference between expected and actual outcomes. The problem is to adjust weights to maximize profits where the inputs include the state of the market in the past, as well as the current state.

Satisficing Agents

The satisficing agents use simple subjective decision making criteria to adjust price and quantity. These agents do not attempt to find the optimal solution, but are satisfied with any solution that meets their subjective criteria. The satisficing approach is equivalent to the rule based approach in artificial intelligence. This type of strategy is consistent with a knowledge based or heuristic approach. The agents may incorporate generalized knowledge (supplied by the researcher or programmer) into their decision making criteria.

Stackelberg Agents

The Stackelberg agents are based on the decision making strategy put forth by Henrich von Stackelberg in 1952 [Stackelberg, 1952]. These agents have knowledge of the consuming agent's average market demand, but do not know the intentions of the other producers in the market. With knowledge of the demand curve, and the actual quantity of goods in the market, the Stackelberg agents are able to compute the theoretically expected market clearing price (e.g.the price at which the supply and demand for goods are exactly equal).

The key feature of the Stackelberg approach is that the Stackelberg agents make a conjecture about how the other producers in the market will respond to changes in production. This conjecture is then incorporated in the Stackelberg agents production decision. Stackelberg agent's explicitly solve classical optimization equations based on the average demand curve, and on their conjecture about the production intentions of the other producers in the market.

CHAPTER IV

NEURAL NETWORKS

Designers of neural network models have used biological brain as their primary inspiration. The structure of the brain allows mental processes such as thought and learning to exist in humans. From this perspective, we see that the type of tasks humans are able to perform depends on the biological character of the brain and its components.

The brain is composed of millions of specialized cells known as neurons (see Figure 3). Neurons have sensitive sites that can send, and others that can receive electrical and chemical signals. Each neuron receives inputs from approximately a thousand other neurons through junctions known as synapses. Additionally, neurons send signals to approximately a thousand other neurons via its output member, called an axon. The signal that is sent is a function of all the input signals that are received. Neurons do not send a continuous signal, but rather a charge builds up until some threshold is passed, and when the threshold is exceeded, the neuron "fires", sending a signal (bust of pulses) to other neurons connected to it.

Each of the approximately 10^{11} neurons in the brain may be connected to as many as 10^3 other neurons (i.e, when a single neuron fires it sends a signal to about a thousand of its neighbors). Overall brain activity does not depend on a single neuron, but on the collection of neurons and the input stimulus. Memories and knowledge are not stored in a single location in the brain.

Instead, knowledge is stored as a pattern among the millions of neural connections.

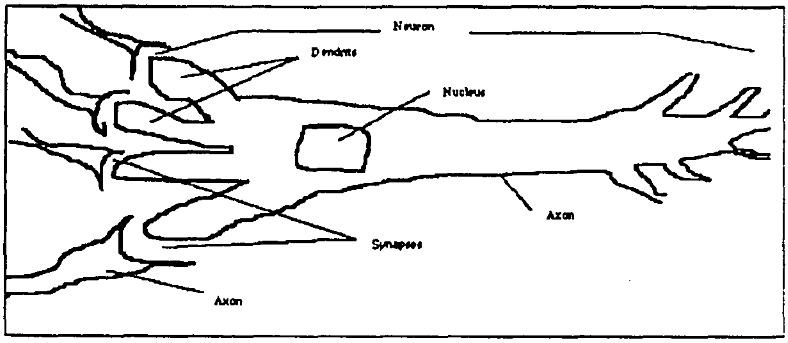

Figure 3. The parts of a neuron.

The brain is fault tolerant because of the way information is stored. For instance, loosing a single brain cell does not result in a person forgetting their grandmother or how to count. Rather, such memories are stored in the connections between many neurons. This apparent redundancy allows the brain to be very resilient to damage. As damage increases, the response of the neural system gradually degrades.

The nature of the brain may also determine why some tasks are relatively easy for the typical human while others are quite difficult. For instance, recognizing a familiar face or voice is naturally easy for most people, but precise mathematical computations are difficult, or at least time consuming, for most of us. On the other hand, the situation is reversed for serial computers. For most computers, recognition of facial features and voice recognition are difficult while mathematical computations are done quickly and accurately. Researchers speculate that the micro-structure of the brain facilitates some tasks but is not well suited to others.

The Venn diagram in Figure 4 compares the capabilities of the human brain and serial computers or more generally, von Neuman machines. The motivation behind neural networks comes from the type of problems that the brain is able to solve. The mundane problems of every day experience have proved to be quite difficult for serial computers programmed with problem solving algorithms. It is a requirement of the technology used in serial computers that each bit of data is stored in a unique site in the computer's memory. The processing units of these computers are complex, and as the name implies, require a single stream of input and output data. They are called serial computers because they only perform one task at a time.

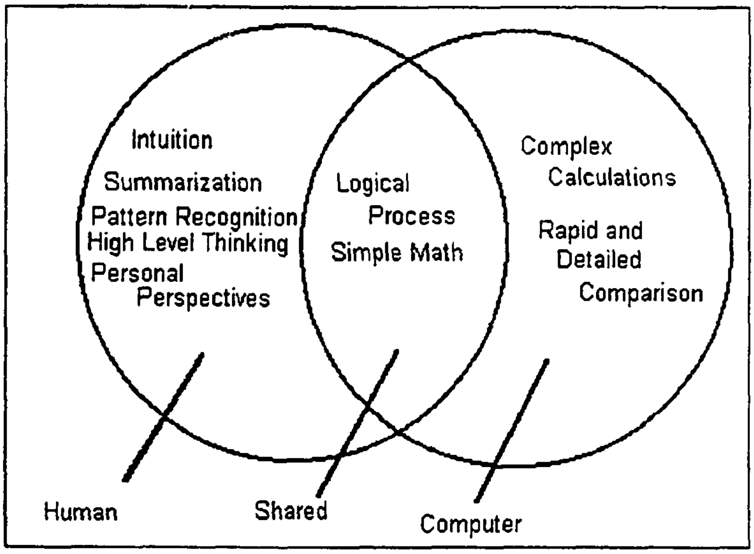

Figure 4. The capabilities of humans and computers.

Confusion sometimes arises when speaking of different computer architectures such as serial, parallel, and distributed. Serial computers as defined above, have a single complex processing unit and can perform only one task at a time. Parallel computers are made by linking the complex processing elements of independent serial computers. Some advanced parallel computers have thousands of linked 386 or 486 processing units. Other parallel computing designs use networks to link independent PC's. These parallel computing schemes all work by decomposing a large problem into small chunks and storing knowledge in memory. The chunks are distributed to local processing elements so that processing takes place on different segments of the problem simultaneously. Synchronization and communication take place by passing messages over the links between processors.

The concept underlying neural networks, on the other hand, is based on the interaction of thousands of simple processing elements. In this scheme, knowledge is distributed over the network of connections and is not held in any particular processing element. The networks are massively parallel in that thousands of processing elements are operating locally and simultaneously. In practice, neural models can be simulated on serial or parallel computers. Furthermore, in recent years several organizations have implemented neural models on silicon chips.

The conceptual force behind neural network models comes from an abstraction of the micro-structure of the human brain. Figure 5 shows an abstraction of a single processing element in a typical neural model. This example processing element has 4 inputs and a single output that splits into 3 branches. The inputs X1 through X4 are multiplied by the corresponding weights W1 through W4 before entering the processing element. The processing elements in neural models are very simple, usually having 2 segments or functions.

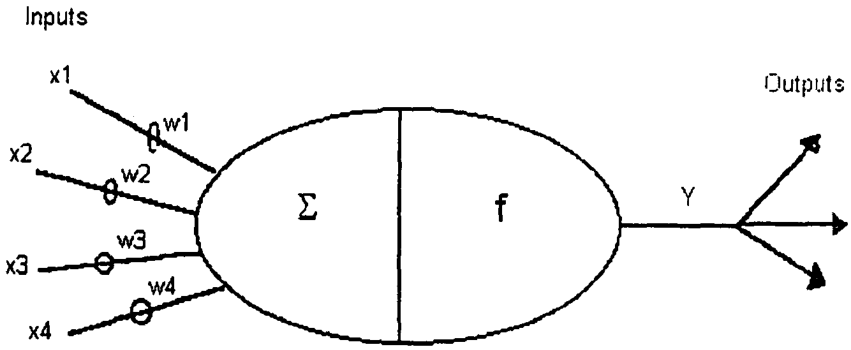

Figure 5. The structure of a neural processing element.

First, the product of the weights and the inputs is summed. Mathematically speaking, this is simply the inner-product of a vector representing the inputs and a vector representing the corresponding weights. Second, the output Y is some function of the result from the inner product calculation. This function is called a transfer function and is chosen by the network designer. The transfer function is usually non-linear, but combinations of linear and non-linear transfer functions may be used in a single network. A logistic (or sigmoid) function is a common choice for the transfer function, although many other choices are feasible. The sigmoid transfer function results in a continuous output between 0 and 1. Furthermore, it has an advantage from a mathematical perspective because it is differentiable.

Neural networks typically contain many processing elements that are connected in parallel. The elements are usually organized in layers, but many arrangements or architectures are possible. The number of processing elements and the choice of network architecture are critical variables that are chosen by the network designer. Since the mid 1980's most network architectures contain at least 3 layers of processing elements. The input and output layers are usually separated by one or more "hidden" layers of processing elements. Figure 6 depicts a simple network architecture showing the input layer, one hidden layer, and the output layer. Input signals are applied to the network simultaneously and move through the hidden layer to the output layer in one forward pass through the net. No processing takes place in the input layer; it merely acts as an input buffer. The processing elements in the hidden and output layers act as described above. The hidden layer is an important substructure in the network because it permits recoding of the inputs before reaching the output layer. The implications of the hidden layer will be discussed in more detail later.

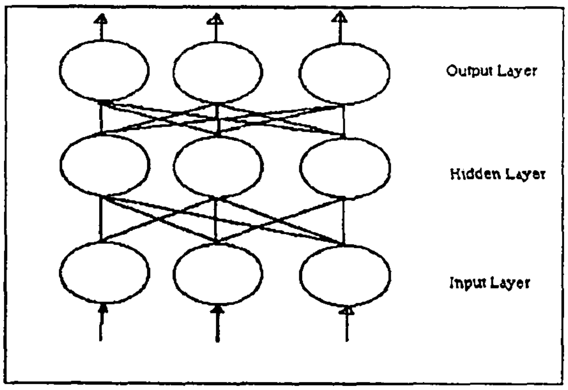

Figure 6. A simple neural network.

The network shown in Figure 6 is an example of a feedforward neural net. This simple structure has proven to be effective for solving a wide variety of problems. Another type of network that is often encountered in the neural network literature includes feedback between processing elements within and between layers. Networks containing feedback have a dynamic character as opposed to the static nature of feedforward nets. The feedforward vs feedback distinction is an important design criterion that depends on the type of problem or application.

So far, the network described mimics the brain by performing distributed and parallel computations and passing the results to the next layer. Knowledge can be stored in such networks by adjusting the weights in a process that is analogous to learning. As mentioned above, knowledge in the brain is not stored in specific neurons but in the patterns among the connections between the neurons. Neural nets have the ability to mimic learning processes by changing their weights. One of the major choices to be made by the network designer is the learning algorithm that will be applied to the network.

Learning algorithms can be distinguished by the presence of a "teacher" that knows the output desired from the network in advance. The learning algorithm is called supervised when the desired output is known in advance and is used to train the network. An unsupervised network learns without pre-existing knowledge of desired network outputs.

To summarize the design choices open to the network researcher, the net is composed of:

Processing Elements -- Type of transfer function.

Architecture -- The number of processing elements and the connections between them. The connections between the elements may form layers and other structures. In addition, the connection paths determine whether feedback will be present in the network.

Learning Algorithm -- The major distinction is whether the network training is supervised or unsupervised. Supervised learning is a process that uses global information and a "teacher", while unsupervised learning uses local information, and is self-organized.

The major distinctions between architectures and learning algorithms are shown in one possible classification scheme in Figure 7.

		Training: Supervised	Training: Unsupervised
Connection Structure	Feedforward (static)		
	Feedback (dynamic)		

Figure 7. Classification of neural networks.

THE MAPPING PROBLEM

In an abstract sense, neural nets perform some mapping from inputs to outputs. The network exists within some problem context such that the problem is represented by the inputs into the net. The measurement (sensory data) may be partially processed before it reaches the neural net, much like the primary processing that takes place before visual or auditory signals reach the brain. By the same token, the outputs from the net also are some representation of the problem. What takes place within the net is some transformation of inputs to outputs. In neural networks, the mapping problem is solved by adjusting the weights in such a way that the network is able to perform the desired input-output mapping.

In terms of binary inputs and outputs, the problem can be represented as in Figure 8. Let the figure represent a black box with n binary inputs and a single binary output that conveys a representation of a problem in some given context.

The condition that the inputs and outputs are binary is only for convenience and will be dropped shortly.

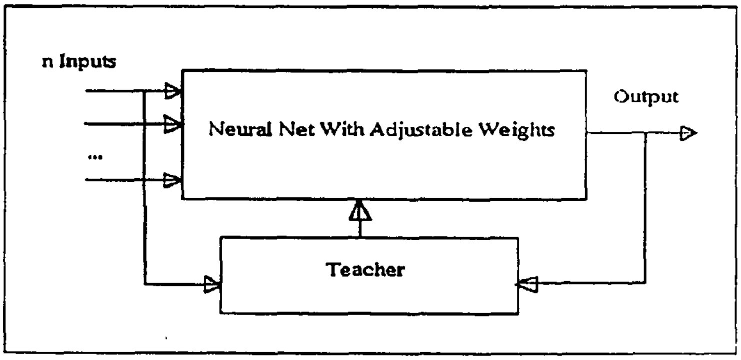

Figure 8. A supervised neural network.

Figure 8 represents a supervised learning context and includes a "teacher". The teacher monitors the inputs into the net and the corresponding outputs. If a given input does not produce the desired output, then the teacher adjusts the weights in the network according to some prespecified procedure called the training algorithm. With the restriction that the inputs and outputs are binary, there are a known number of possible input-output mappings: 22^n where n is the number of inputs, 2^n is the number of possible input patterns, and 22^n is the number of possible mappings. It should be apparent that the number of possible mappings increases extremely rapidly as the number of inputs into the network increases. For instance, 4 inputs results in 256 mappings but 10 inputs results in over 10^{300} (substantially over a trillion, trillion) possible mappings. Thus there is an explosion in the size of the mapping space as the number of inputs increase.

A given network may not be capable of performing all the theoretically possible input-output mappings. Generally, the number of mappings that the network can be made to perform is orders of magnitude smaller than the number of possible mappings. The network search is confined to those mappings that a given net can perform by adjusting the weights in the network. Using the notation of Lendaris [Lendaris, 1990b], the space of mappings that the network is capable of performing by adjusting the weights is called the performance space of the network. The input-output mapping that the teacher specifies is a point in the set of all possible mappings. Usually, however, the number of input patterns for which a specified output is required is smaller than the whole set. In this circumstance, the teacher specifies a partial function over what we call the "care" terms. Each unspecified input pattern can be given any output, i.e, we "don't care" what value is assigned to them. If there are n don't care terms, there are 2^n ways to specify them while still having the desired assignment to "care" terms. These are called extensions of the partial function defined on the care terms, and comprise a 'care set' of allowable total functions that perform the desired mapping on the care terms. The given neural network is capable of performing a specified mapping only if its performance space and the care space have at least one point in common. If there is not any overlap between the care space and the performance space, then no solution to the mapping problem exits for the given network. These situations are depicted in Figure 9. In Figure 9A, the performance space of a given network does not overlap with the care space associated with a specified desired mapping, so no solution to the mapping problem exits. Figure 9B depicts a situation where the neural network has a number of possible weight configurations that will perform the desired partial mapping; the task of the learning procedure is for the network to find one of the solutions in the overlapping region. Usually it is not possible to determine in advance if the care space and the performance space overlap, i.e, if it is possible for the given neural network to perform the given input-output mapping. Solving the mapping problem is accomplished by searching the performance space of the network with some learning algorithm.

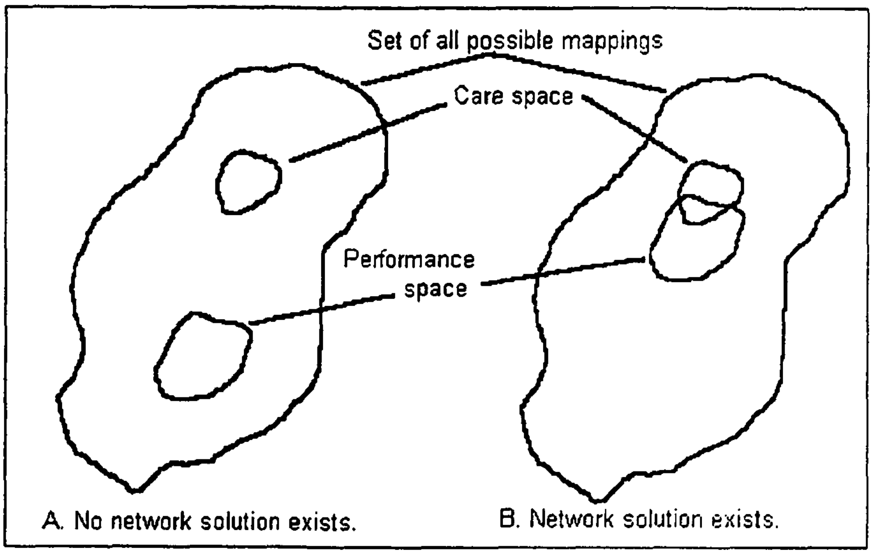

Figure 9. The problem space.

In neural network parlance, the search strategy is known as the training algorithm. As mentioned above, the type of search strategy used to train the network is a major design consideration. Training algorithms are usually couched in terms of learning the given set of input-output responses by some type of error correction or adaptive algorithm. Such algorithms will be discussed in more detail later.

In the case where the performance space and care space overlap, it is possible to find a solution. But no such overlap is guaranteed, and even if it does exist it may take a very long time to find it. In the event that no solution to the input-output mapping is forthcoming from a network, alternative strategies include that of moving the performance space (i.e, changing the network structure, and hence the set of mappings that the neural network can perform). Changing the network architecture results in a new performance space. Some attempts at tailoring the neural network architecture have begun to appear in the literature e.g. [Weigend et. al. 1990, 1991]. Only recently are guiding principles starting to emerge for designing an architecture for a given problem [Lendaris, et. al., 1993]. This task amounts to searching the set of all possible mappings by incrementally changing the network architecture.

Another alternative for moving the performance space of the network is to change the problem representation. A problem that is difficult or impossible to solve in one representation may prove to be trivial in another representation. Problem representations may be changed by transforming the input-output data in some way. A typical example from human experience is the perceptual difference between textual and graphic representations of the same problem. Problems that are displayed graphically are often easier to solve.

The problems discussed above were confined to binary inputs that have exact representations. The problems are compounded when the data is continuous and stochastic. In such cases, the set of possible mappings increases without bound because the inputs and outputs are continuous. Furthermore, with stochastic relations, identical inputs can result in different outputs.

PRINCIPAL RESEARCHERS

Scholars in a broad range of disciplines have conducted neural network research since the 1940's, but widespread interest in neural computing has been irregular over time. In 1943 McCulloch and Pitts, published the first paper to discuss neural computing. McCulloch and Pitts had backgrounds in neurobiology and statistics, respectively. Since that time, neural nets have been investigated by researchers from widely different fields.

Donald Hebb is credited with one of the first learning rules for neural elements. In 1949 Hebb proposed a learning rule based on reinforcing weights between neural elements that fire simultaneously [Hebb, 1949]. Rumelhart and McClelland [Rumelhart and McClelland, 1986], restate this rule as follows:

"When unit A and unit B are simultaneously excited, increase the strength of the connection between them."

This general approach has been widely used and modified over the years, but is not used in its original form anymore. However, the Hebbian principle underlies virtually all training algorithms used at this time, whether supervised or unsupervised.

Marvin Minsky has shown a long-term interest in learning machines. In 1951 he constructed his first learning machine from hundreds of tubes, motors and belts and clutches. This device actually learned by adjusting its own control knobs through a series of clutches. Later, in 1956, Minsky was one of the organizers of the first artificial intelligence conference. It is reported that the participants of this conference outlined several targets for artificial intelligence research including natural language processing, pattern recognition, learning, acquisition and application of expert knowledge, and so on. The problems that were assumed to be easier were the mundane problems of every day human experience, while acquisition and application of expert knowledge were deemed more difficult. Time has shown that the more difficult problems to solve are the mundane and commonplace. As of this time, Minsky is still actively working in the artificial intelligence arena.

Frank Rosenblatt and Marvin Minsky were associates and rivals in the early development of neural networks. Rosenblatt developed a neural like element that he called a Perceptron in 1957. By 1962 he had done further research and published a book called Principles of Neurodynamics [Rosenblatt, 1962] in which he develops the Perceptron learning theorem. This theorem asserts that for a network containing a single adaptive Perceptron, if a solution to a given mapping problem exists (i.e, the solution is contained in the Perceptron's performance space) then the Perceptron will find it in finite time. The Perceptron proved to be a simple but powerful learning mechanism and was widely applied in the 1960's. Unfortunately, the then existing theoretical results applied only to the single-Perceptron case, and provided no guidance for training more complicated networks.

Minsky and Papert published a thorough mathematical study of Perceptrons in 1969 [Minsky & Papert, 1969]. This book was credited with reducing the popularity of neural network research for over 10 years. Minsky and Papert found that Perceptrons could solve only those classification problems that are called linearly separable. These restrictions on Perceptrons implied, in Minsky and Papert's view, that Perceptrons could not solve most interesting problems. The combinatorial explosion was so serious that the problem domain was restricted to toy problems.

After the revitalization of neural network research in the 1980's, Minsky and Papert revised Perceptrons in light of new knowledge [Minsky & Papert, 1988].

However, many of their basic conclusions remain unchanged. Combinatorial explosion results in enormous search areas that must be traversed to find solutions. The newer network models use learning algorithms that are equivalent to well known hill climbing methods with all their faults. In addition, Minsky and Papert claim that the new generation of neural researchers are repeating many of the same mistakes as the earlier researchers, especially in regard to critical evaluation of research work. Furthermore they criticize the continuous rediscovery of what was well known to an earlier generation. In the other direction, many current researchers criticize the Minsky and Papert books as overly pessimistic.

A few researchers such as James Anderson, Teuvo Kohonen and Steven Grossberg continued to study and experiment with neural networks throughout the 1970's. Anderson worked with linear associators and various extensions. Kohonen did pioneering work in adaptive memories and competitive learning. Grossberg developed neurologically inspired models and developed what is known as Adaptive Resonance Theory.

John Hopfield is credited with breathing new life into the neural network community with the delivery of a research paper in 1982 [Hopfield, 1982] which successfully treated a significantly more complex neural network architecture. Hopfield, who is a physicist, presented a new view for thinking about and developing training algorithms. His algorithm was based on the notion of finding the minimum of an energy function and proved to be quite effective for solving certain types of problems. Unfortunately, later research has shown that Hopfield networks often find very poor local minima rather than the global energy minima. Many modifications are being introduced to alleviate some of these difficulties. More significantly however, Hopfield's success motivated other researchers to work with a variety of new architectures.

In their two volume 1986 book, Parallel Distributed Processing, Rumelhart and McClelland [Rumelhart and McClelland,1986] provide a foundation for the new neural networks movement. Rumelhart and McClelland are editors of the book and leaders of a research team known as the PDP group (then located at UC San Diego). Rumelhart rediscovered and effectively applied what he called the backpropagation algorithm using a "generalized delta rule". This algorithm broke the barrier of a single element, single layer network that plagued the Perceptron algorithm, and has become one of the most widely used neural learning procedures. The algorithm effectively performs a type of gradient descent, or hill climbing on a surface defined by a least-squared-error criterion function. Another important contribution of the PDP group is the demonstration of the importance of hidden layers for solving problems that were not solvable by single layer Perceptrons. The authors relate an excitement and enthusiasm for their work that is not shared by Minsky and Papert, as mentioned above. Indeed, their Generalized Delta Rule solved some of the problems that Minsky and Papert complained about so much in their books.

More recently, Linsker has introduced another new concept for thinking about and analyzing the training process. Linsker introduced an Infomax principle which effectively maximizes entropy in each layer of the network while still using a Hebb type rule [Linsker, 1988]. This new approach shows promise for solving problems of self organization. Linsker has developed a careful mathematical analysis to guide his approach.

In the late 1980's Halbert White, a well known economist, published several research papers that tie feedforward neural networks with stochastic approximators [White, 1989]. Further ties with statistical theory and non-linear approximation are discussed in his papers published in 1990 [Hornick, Stinchcome & White, 1990], [White, 1990]. These papers detail methods for statistically testing the significance of coefficients or weights resulting from neural estimation. In addition, White shows that with certain modifications, neural models can be shown to be statistically efficient.

While many of the early developments in neural computing were based on empirical data and heuristic insights, a broad theoretical base is beginning to emerge in the field. Simultaneously, there is a veritable explosion in the numbers and breadth of applications of this technology. This dissertation puts forth another such application.

CHAPTER V

THE BLACKBOARD MODEL: AN EXAMPLE OF A DISTRIBUTED ECONOMIC COMPUTER

INTRODUCTION

Blackboard Systems

Recently certain types of systems have been studied via a class of distributed models known as blackboard systems [Nii, 1986], [Engelmore & Morgan, 1988]. A blackboard model typically consists of a system with subunits of two general types:

1. Blackboard structure

2. Multiple knowledge sources

The blackboard is a shared resource to which all of the knowledge sources have access. The system has data inputs from the environment that are written directly on the blackboard. The knowledge sources simultaneously respond to the state of the world as it is represented on the blackboard structure. The knowledge sources are able to read from and write to the blackboard in real time, thus altering the state of the blackboard.

If the blackboard reaches a steady state, a consensus has emerged among the knowledge sources. Depending on the context, the steady state may be a solution to a problem which emerges from the interaction of multiple knowledge sources, or to the data that entered from the environment.

There is no central control or processing unit in the general blackboard model. Instead, control may reside in the blackboard, the knowledge sources, in a separate unit, or in some combination of the above. In general, communication between knowledge sources occurs only through the blackboard.

The knowledge sources may have specific domain knowledge about some aspects of the problem presented by the data on the blackboard. Thus, the knowledge sources may deal with entirely different portions of the problem space, where there may be starkly different representations of the problem, and bring radically different problem solving strategies to bear on the problem. In fact the knowledge sources may not even agree on the definition of the problem.

A type of debate develops in a blackboard system that is engendered by the multiple views and specific knowledge of the participants in the system. The blackboard system provides an ideal environment in which to study the effects of multiple approaches to problem solving, and the interaction among various types of agents that have different goals, and use different problem solving techniques.

This view of a blackboard system is broader than that normally expressed in the blackboard systems literature, where the focus is typically on cooperative problem solving. In such cases, the researcher designs a blackboard model for a particular problem context, and the knowledge sources represent a well ordered and cooperative approach to solving the specific problem selected by the researcher. However, I have taken the position that there is no reason to restrict blackboard systems to cooperative problem solving.

The following quote from Engelmore and Morgan illustrates the common thinking about control in blackboard systems:

"The focus of attention indicates the next thing to be processed. The focus of attention can be either the knowledge sources (that is, which knowledge sources to activate next) or the blackboard objects (i.e. which solution islands to pursue next), or a combination of both (i.e. which knowledge sources to apply to which objects). Any given system usually employs one of the three approaches, not all"

I believe that this approach to control and problem solving in blackboard systems is unnecessarily restrictive. The blackboard application that I will describe will not have a directed, cooperative problem solving approach. Instead the approach used in this research is competitive and is not centrally directed. The agents that share the blackboard have conflicting goals.

History of Blackboard Systems

The blackboard concept is very general and has given birth to several distinct problem solving strategies. An area of common or global computer memory is the central concept that is shared by blackboard models and applications. The earliest reference to a blackboard structure in the artificial intelligence literature was by Allen Newell:

"Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and to judge when he has something worthwhile to add to it. This conception is just that of Selfridge's Pandemonium (Selfridge, 1959): a set of demons, each independently looking at the total situation and shrieking in proportion to what they see that fits their natures ..." [Newell, 1962, as printed in Englemore and Morgan, 1988]

Newell's collaborator and research partner, Herbert Simon also made occasional references to blackboards. By 1972 the original concept of the blackboard evolved into the production system. [Newell and Simon, 1972]. A number of goal-directed, generate-and-test systems use the basic concept of global shared memory within a given problem solving context. Simon defined the blackboards as follows: "... I will call the information about the task environment that is noticed in the course of problem solution and fixated in permanent (or relatively long-term) memory the 'blackboard' ..." [Simon, 1977] The blackboard metaphor for a shared memory structure has fallen into disuse in the rule based areas of artificial intelligence. However, the blackboard concept continues to be widely used in the area of speech understanding and recognition.

Applications of Blackboard Systems

Herbert Simon is credited with suggesting the blackboard concept to Reddy and Erman, who applied the concept in the Hearsay II speech understanding project in the 1971-1976 period. In the Hearsay II implementation, the blackboard emerged as a recognizable entity with multiple levels of problem solution. The blackboard was operated on by knowledge sources such as signal acquisition, word spotting, phrase-island generation, phrase extension, rating and interpretation.

Hearsay II generally uses a cooperative approach for solving a speech understanding problem. Problems of competition among the different knowledge sources are of two types. First, there is a scheduling competition that is generally solved by the control module, which decides which knowledge sources can access the blackboard first. Second, ambiguity problems arise where different knowledge sources come to different solutions based on the same evidence. Similar problems can arise in the rule based systems that use and alter global memory. Competition of this nature is resolved within the system, often by a control function. This is distinctly different from the type of competition found in economic systems.

The approach used by Hearsay II was successful enough to spawn many followers in speech understanding research. The flexible approach of blackboard systems has encouraged researchers in numerous other fields to apply them as well. For example:

Three dimensional structural modeling of proteins [Hayes-Roth et. al., 1985] .

System for designing construction layouts [Tourmaline et al., 1987a]

Control problems [Hayes-Roth et al., 1985].

Sonar interpretation [Ni and Feigenbaum, 1978].

Vehicular tracking and testing [Lesser and Corkhill 1978].

Protein crystallography interpretation [Terry, 1983].

Planning systems [Hays-Roth et al., 1986].

Blackboard systems seem to be particularly effective for problems that:

1. Require many distinct kinds of knowledge for solution.

2. Integrate disparate information.

3. Contain natural domain hierarchy (i.e. micro and macro level data)

4. Have continuous data and thus may not have a "final" solution.

5. Have uncertain knowledge and data.

In short, blackboard systems provide a flexible and adaptable conceptualization for modelling real world parallel and distributed systems where information and knowledge are readily shared, but different goals, values and perspectives interact.

For instance, some may have microscopic knowledge of particular areas, while others have global system or meta-system level knowledge. The intent is to collect knowledge sources that interact synergistically in real time. This research expands the blackboard concept to allow for competition rather than just cooperation among the agents, and discards the control restrictions that are not consistent with economic systems. This model of a distributed economic system is more consistent with the nature of an economy than the traditional classical models. This approach allows agents with disparate stratagems and goals to exist side-by-side in the same market. While a neural network is a distributed computing system, it does not provide a conceptually satisfactory way to model an economic system. However, a neural network is a reasonable computational technique to use for an agent that learns about the market that it participates in.

The next section describes my implementation of the distributed computer market and details the format of the input and output from the market. The distributed market consists of several computer programs that operate independently but share information through a blackboard structure. The programs run in the DOS environment and thus are not truly parallel, but simulate a parallel system. Timing and sequencing are provided by the top level program t.exe. The top level program has no awareness of the decisions or interaction in the market, so the system is distributed in the sense that decisions are made by independent agents.

ASSUMPTIONS

A market exists within an economic environment where agents exchange at least two goods. All consumption occurs within a single time period. In the initial time period a quantity of goods is assigned to each agent by the researcher. A production decision is made at the end of each period. Each agent has an internal value, in terms of the unit of exchange, that is assigned to the goods in their possession. The value of the goods is known only to the agent that possesses the good and varies among the agents. These value schedules comprise values from within the bounds of the region in which the agents can make gains from trade (the feasible region). Trade takes place at a value in the interval between the buyer's and seller's internal values.

The interval between the buyer's internal value and the seller's internal value is known as the feasible region. The gains from trade are not necessarily evenly distributed, but depend on the sales price that is negotiated between buyer and seller.

The price variable (denoted p in the supply and demand equations) is a key emergent variable in the system. All the producers and consumers may not use the same value for p. The overall supply in any period is given by the sum of the quantities supplied by the producers, and is fixed at the beginning of the period. Each producer sets its expected quantity and price at the beginning of the period, and advertises them on the blackboard. The consumers are then faced with an array of prices and quantities for the good that is available in the market. The consumers are price takers; they make demand decisions based on the price in the market and cannot bargain for a better price. Producers do have the ability to change prices at any time, but must wait until the end of a period to change the supply. Thus a producer may adjust prices within a period, but may adjust quantity only between periods. How agents make decisions is not under any central control. The agents are free to use any means to make trades that are deemed favorable to themselves.

Time is treated as a finite number of discrete chunks that are aggregated into intervals of longer length called periods and rounds. A period is made up of a given number of time steps. A round is made up of a given number of periods. An experiment is made up of a given number of rounds. The actual number of steps per period, periods per round, and rounds per experiment are assigned by the researcher, and are common knowledge for all agents. Knowledge that is available to the agents includes

The number of buyers and sellers

The identity of each buyer and seller

Number of time steps, periods and rounds

The current time, period and round

The price and quantity of previous trades

The agent's own internal valuation schedule

Knowledge that is not known to the agents includes

Equilibrium price

Equilibrium quantity

Quantity that is produced in the next period

Internal value schedule of other agents

After the initial period, the producers decide how much of the good to produce in the coming period. This quantity of goods remains fixed within each period, and goods cannot be held over from one period to the next.

The blackboard is equally accessible to all agents in the market. The blackboard contains all information that is common knowledge. The blackboard is analogous to a newspaper where prices and terms of trade are advertised. The sellers advertise the price at which they are willing to trade on the blackboard in each time step.

At each step, all consumers make a buy decision that entails buying a quantity of the good from a particular seller at the seller's advertised price. The consumers cannot spend more than their income for the period. All exchanges are made directly between buyer and seller without relying on a middleman, monitor or auctioneer.

If multiple orders are received by a given seller in a time step, then one of the orders is chosen at random to fill first. If all the seller's goods are not sold, then another order is chosen at random to fill, and so on until all orders have been filled or the seller's supply is exhausted. Orders can be partially filled, but on a first-come, first-served basis.

PROGRAMS AND INPUT FILES

market

The market file is a text file that contains information about the market characteristics that are used in the market simulation. This file can be changed by the researcher with a text editor. An example of a market file is shown in Figure 10.

5	number of time segments in each period
4	number of periods in each round
90	number of rounds in the market simulation
1000	initial wealth for each producer

Figure 10. Listing of a typical market setup file.

agents

Each line of the agents file contains information about a single agent in the market. This information consists of the program name of each agent, the role the agent is assuming in the market, (p for producer or c for consumer), and a name that identifies the agent. The agents file is a text file and can be changed by the researcher.

Any number of agents can be listed in the agents file, and a single agent program can be used more than once, allowing multiple copies of the same program to compete. The agents do not have to be listed in any special order, but arrangement of information on each line is fixed. An example of the agents file is shown in Figure 11.

Agent	Type	Name
prod1.exe	p	producer0
prod2.exe	p	producer1
prod3.exe	p	producer2
prod4.exe	p	producer3
prod1.exe	p	producer4
prod2.exe	p	producer5
cons1.exe	c	consumer0
cons1.exe	c	consumer1
cons1.exe	c	consumer2
cons1.exe	c	consumer3
cons1.exe	c	consumer4
cons1.exe	c	consumer5

Figure 11. Listing of a typical agents file.

This example file shows 4 different producer programs and 1 consumer program in a market consisting of 6 producers and 6 consumers (i.e. each of the consumers is implemented by a separate copy of the same consumer program). While the code for consumer programs is identical, in application, the programs are distinguished by supplying them with random parameters that will be discussed later.

t.exe

The top level program is responsible for timing and sequencing the calls to the individual agents as well as taking care of bookkeeping and other maintenance tasks. The basic structure of the market is determined by this program. For instance, the program randomizes the order of agents in the queue to insure that the agents are not called in the same sequence repeatedly. Questions such as whether the producers can be called only between periods are determined by the structure of the top level program.

t1.exe

This program is called only once during a market simulation. The purpose of t1.exe is to set up the blackboard after reading the agent and market files. The program t1.exe also assigns a set of random coefficients to each agent that are used by the agent throughout the market simulation. These coefficients distinguish and individualize agents of the same type by providing some unique characteristics to each agent.

prod.exe and cons.exe

Programs describing the behavior of producers and consumers are independent of each other and of the top level program. These programs read from the blackboard file, make some decisions about production, consumption and trade under the current market conditions, and write the results of these decisions on the blackboard, thus modifying the state of the market. As mentioned above, the names of the programs are stored in a queue that is shuffled between calls to prevent favoring agents solely because of their position in the queue.

Basic characteristics of the agents such as supply and demand relations are directly under the researcher's control. Each agent receives a set of coefficients that are stored at the start of the market and retains them throughout the market.

update.exe

The update program reads the blackboard and records the prices and quantities that are advertised at the end of each period. In addition, the time, period and round counters are updated by this program.

record.exe

The record program transfers data from the blackboard file to a text file when the market closes.

OUTPUT FILES

bb.bin and bb.txt

Data accumulates in several files as the market evolves. bb.bin is a binary file that is available to all agents for reading and writing. As mentioned above, this file is converted to a text file when the market closes, thus recording the final conditions in the market.

trade.txt

Each completed trade is noted in the trade file by the consumer that completes the trade. Reading from left to right the trade file records: round, period, time, producer, consumer, price and quantity traded. The trade file can accumulate large quantities of data, mostly useful for debugging, and is not generated for most tests.

hist.txt

The history file records the state of the blackboard at the end of each time interval. It records from left to right: round, period, time, price advertised by each producer, and quantity that each producer has in inventory. This file is updated by the update.exe program. The history file is most useful for debugging and is not generated for most tests.

ave.txt

The average file is updated after each time step. This file records cumulative quantities, values and average prices within each period, round and market. The data are grouped as follows:

ave price

total quantity

total value

Total trades in the period are updated after each time step in the period. The value is computed as P_iX_i at each time step, and the average price is computed as: average price = total value/total quantity. The figures accumulate throughout the period but are reset to 0 at the beginning of the next period.

DESCRIPTION OF THE AGENTS

The following variables are defined and are used in the description of the agents:

P = Market Price

Pe= Expected price

Q = Market Quantity

Qe= Expected Sales Quantity

T = Number of time steps per period

S = Number of consumers in the market

N = Number of Producer in the market

i = Index identifying an individual agent in the market

<u>cons</u>

The same consumer program is used for all the market simulations. Many instances of the same program run at the same time, but are differentiated by supplying each agent with unique coefficients. The consumer program operates as an agent that is a price taker. That is, the consumer takes the advertised price as given, and does not attempt to bargain with the seller. The consumer does optimize by choosing the lowest price available. Each agent examines the blackboard and chooses to trade with the producer offering the lowest price.

The quantity that the consumer offers to buy is a function of the individual agent's demand curve. At a given price, each consumer has a different demand. Some may not be willing to purchase anything, while others may purchase a substantial amount. The offers are filled by the producer that offered the lowest price on a first-come-first-serve basis. The producer may not be able to fill all the orders from inventory, so some orders may go unfilled or partially filled. In order to simulate a parallel system, the offers are shuffled on the offer queue to prevent always filling the same consumer's order first. The consumer program also has several maintenance functions. These include updating the screen, the blackboard and the trades file.

SATISFICING AGENTS

The logic used by the satisficing agents was not based on optimization in any explicit way. These agents make decisions based only on the inventory held and the historical market price. The basis for making the production quantity decision for the next period varies widely among the satisficing agents.

ps1

This agent adjusts price in the period based on the time remaining in the period, and the quantity remaining in inventory. Fixed expectations are used as the guideline for adjusting price. That is, if after α percent of the time in the period has expired, and more than β percent of the starting inventory remains in stock, then decrease price. Alternatively, if less than α percent of the available time has expired, and more than β percent of the agent's starting inventory has already been exchanged, then raise price. These relations are shown below in Figure 12.

		Time	
		$t>\alpha$	$t<=\alpha$
Quantity	$q>\beta$	Raise Price	No Change
	$q<\beta$	No Change	Decrease Price

Figure 12. Price adjustment scheme for ps1.

The size of the price adjustments employed for these experiments was 5.0 percent in each time step. At the beginning of each period, agent ps1 attempts to capture market share by bidding slightly lower than the competition's price. For these experiments, the agent opens the period with a price that is (1-α) less than the average price in the previous period.

$Pe= \alpha \Sigma P_i Q_i / \Sigma Q_i$ for all transactions i, $0 < \alpha <= 1$.

The value of α was 0.99 in the experiments reported here. This agent makes the quantity decision for the next period by examining the change in profit over the previous two periods, and additionally, the unsold inventory at the end of the period (See Figure 13).

		Excess Stock	
		Yes	No
Profit	Increase	Small Decrease	Large Increase
	Decrease	Large Decrease	Small Increase

Figure 13. Quantity adjustment scheme for ps1.

ps2

The second satisficing agent, ps2, was identical to ps1, with the exception of the period opening price formation rule. ps2 employed the average price from the previous period as the opening price.

$Pe = \alpha \Sigma P_i Q_i / \Sigma Q_i$ for all transactions i, $\alpha = 1.0$

As a result of using this price rule, this agent does not engage in price wars, but follows the market.

ps3

Agent ps3 follows the pattern set by agents ps1 and ps2, with the exception that it increases price at the beginning of each period using average price in the last period as the base. This removes some of the risk of selling below market price in an uncertain market. The agent has the opportunity to adjust price if fewer than expected sales are made in the early part of the period. The value of α was 0.1 in these experiments.

$Pe = (1+\alpha) \Sigma P_i Q_i / \Sigma Q_i$ for all transactions i, $0 <= \alpha <= 1$.

Agents ps4, ps5 and ps6 were eliminated in the preliminary experiments and are not reported on here.

ps7

This agent bases it's production decision on capturing a "fair share" of the market. The agent produces for the next period based on the total quantity available in the previous period, divided by the total number of producers. e.g. $q_{(t+1)} = Q_{(t)}/n$. Thus the agent assumes that each producer will supply an equal portion of the market, and that the demand in the next period will be unchanged (e.g., the agent uses a static expectations model to forecast production). The opening price is based on the 10 percent over average price rule shown for agent ps3.

ps8

The satisficing producer ps8 bases its quantity decision for the next period on the remaining inventory at the end of the last period. The end-of-period equilibrium inventory for this agent is between preset limits. An ending inventory below the limit results in an increase in the quantity produced, while an ending inventory over the limit results in a decrease in quantity produced.

An objection to this rule is that it is not consistent with economic rationality. That is, if the producer comes to equilibrium with between α and β percent of stock remaining in inventory at the close of the period, then the producer could increase profits by decreasing production. However, such rules are used in the case of some products where production costs are low, and the price changes infrequently while the demand fluctuates daily. Newspapers are an example of such a product. Another response to this objection is that these agents are not optimizers and thus do not necessarily follow the classical economic logic.

Ending Inventory =q

Expected Sales $=q^e$

$q <= \alpha q^e$	Increase q^e
$\alpha q^e < q < =\beta q^e$	No Change
$q >= \beta q^e$	Decrease q^e

The value of α was 0.03, and β was 0.05 for these experiments. The opening price for agents ps8 is based on the 10-percent-over-average-price rule as shown for agent ps3.

STACKELBERG AGENTS

The Stackelberg class agents include a substantial gain in knowledge about the market when compared with the satisficing agents. The Stackelberg agents have knowledge of the average demand relation used by the consuming agents. Thus, if the total quantity produced by all agents in the period is known, the Stackelberg class agents can compute the market clearing price. These agents, however, do not know the intentions of the competing producing agents. The Stackelberg agents are assigned different coefficients that represent their conjecture about the response of other agents to a change in their own production. The conjectural variation (CV) used by these agents are shown in Table I.

TABLE I

CONJECTURAL VARIATION APPLIED TO STACKELBERG AGENTS

AGENT	CV
pc000	0.0
pc025	-0.25
pc050	-0.50
pc100	-1.00

The Stackelberg agents compute the quantity for the next period based on the following equation:

$$q^e = q^e_{(t-1)} + \alpha_0 - \beta_1 - 0.5^* [\, Q_{(t-1)} + q^e_{(t-1)} - CV * q^e_{(t-1)}] \,/ (T^*S)$$

Where q^e is the expected production for the next period, α_0 and β_1 are coefficients from the cost and demand functions that are known to the Stackelberg class agents. T represents the number of times trading is allowed in each period, while S represents the number of consumers in the market. CV is the conjectural variation assigned to the particular implementation of the Stackelberg agent. The agents use their knowledge of the average demand curve of the consumers to set an expected market clearing price.

$$P^e = \beta_0 - (Q^e / \beta_1 {}^* T^* S)$$

Where P^e is the expected price for the next period, Q^e is the expected total production for the next period, β_0 and β_1 are coefficients from the demand function that are known to the Stackelberg class agents. T represents the number of times trading is allowed in each period, while S represents the number of consumers in the market.

The Stackelberg agents also adjust price in each time step by using the above equation, but substituting the actual inventory that is in the market for Q^e. The market inventory is available to all agents after the first time step when quantities are advertised on the blackboard. These agents take advantage of the market inventory data to compute the market clearing price and make adjustments to their price as necessary.

OPTIMIZING AGENTS

The optimizing agents have no specific knowledge about the form or coefficients of the demand curve, or about other the producers that they are competing with. Instead, the optimizing agents learn about the environment that they are in. Two optimizing agents were created for this research, and both use neural networks to make their decisions.

The agent pn1 uses two internal neural nets to make quantity decisions, but bases opening price decisions on the average price in the market. The agent pn2 incorporates three neural networks, and uses them to make opening price and quantity decisions. The logic used in the neural optimizing agents is shown below in Figure 14.

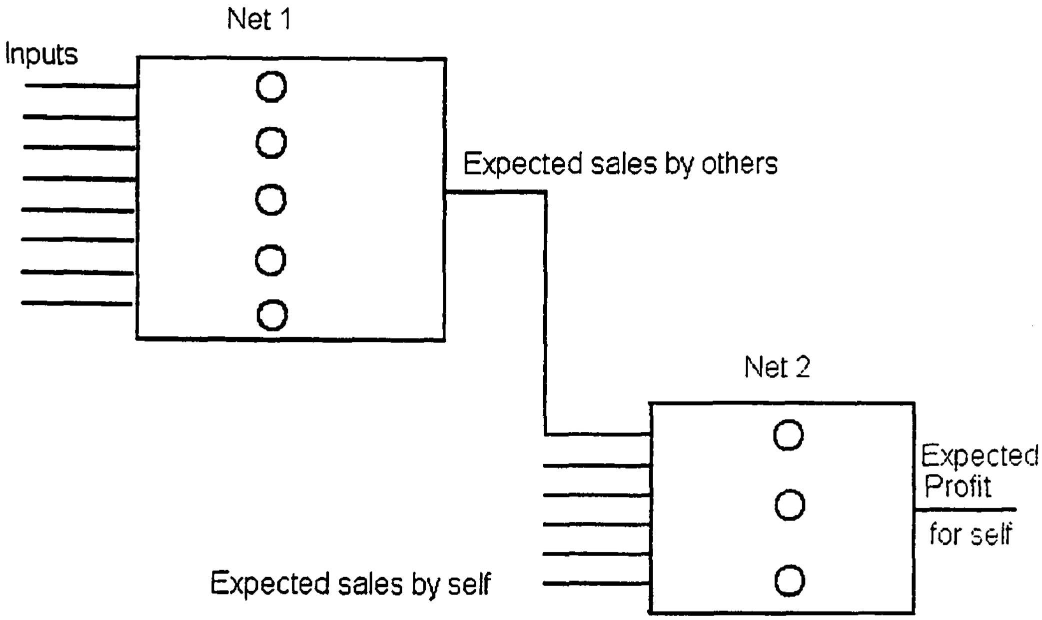

Figure 14. Neural nets for computing output and profit.

The first net has 7 inputs (plus a bias term), 1 output, and 5 elements in its hidden layer (See Table II). The function of the network is to make the best possible guess about the total expected sales by other agents. That is, the net is attempting to forecast the amount of sales that all of the other producing agents expect to fulfill in the next period. Internally, the network uses a sigmoid transfer function in each element.

This agent avoids the problem of making a predetermined and unchangeable conjecture about the response of other agents in the market, since this is implicitly included in the reaction of others, and is learned by the network through repeated exposure to the data.

TABLE II

INPUTS AND OUTPUTS FOR NET 1

Inputs(Time = t-1)	Output (Time = t)
Number of Producers	Expected Sales by Other Agents
Number of Consumers	
Times per period	
Expected Sales by Agent	
Unsold Inventory Owned by Agent	
Expected Sales by Other Agents	
Unsold Inventory Owned by Other Agents	

The function of the second network is to compute the expected profit in the ensuing period. The second network includes 5 inputs (plus a bias), one output, and three elements in the hidden layer. This net uses a hyperbolic tangent as a transfer function, which allows negative values (losses instead of profits), and ultimately produced much better results than the sigmoid transfer function. The price is not explicitly included in the network, but is included implicitly as a function of the exogenous variables.

TABLE III

INPUTS AND OUTPUTS FOR NET 2

Inputs(Time= t-1)	Outputs (Time = t)
Number of Producers	Expected Profit by Agent
Number of Consumers	
Times per period	
Expected Sales by Agent	
Expected Sales by Other Agents	

After updating the weights and computing the expected sales by others in the first net, the only remaining unknown is the expected sales by the agent. The agent optimizes its expected profit by adjusting its expected sales. That is, the agent starts with a low level of expected sales and computes the corresponding expected profit with the network. The agent then increases the expected sales and recomputes the expected profits. This process continues for an interval surrounding the value of expected sales in the previous period. The agent then chooses to produce the quantity that yields the highest expected profit.

The network cycles through the data 100 times at the end of each period, adjusting weights with the backpropagation algorithm at each cycle. After updating the weights to include the most recent data, the weights are fixed and the data from the most recent period is used to compute the expected sales by others in the next period.

The neural optimizing agents must be "matured" by exposing them to a variety of markets for the purpose of learning. A naive agent has a random set of coefficients and has no experience on which to base decisions. As implemented in these agents, the learning process is actually a dynamic weight adjustment procedure that proceeds both on and off line. Each neural agent maintains weight files where the connection weights are stored. Separate weigh and data files are maintained for quantity, profit and price projections for each agent. The weights are adjusted on-line at the end of each period. The data on which training and decision making are based accumulates in files at the end of each period.

The on-line adjustment procedure calls for 100 passes through the last 100 observations in the data file. The number of passes and observations included in the training set were chosen to balance speed and quality of projections. Longer training periods with more data may produce somewhat better results but make the computational load excessive. This was compensated for by accumulating data from market simulations and training the agents off-line for up to 30,000 passes through the data retained from other markets. The mature weight files used for the tests of agent pn2 were developed through exposure to a wide variety of market types.

The on-line training allows the agents to respond to local conditions in the market in which they are participating. Additionally, limiting the data to the last 100 observations encourages response to current conditions rather than to older historical averages.

The dynamic nature of the weight files indicates that the behavior of a given agent evolves over time. This has important consequences for experimental design. For instance, when measuring the performance of a neural agent in a round robin competition, the order of the trials could have serious consequences because the weight file changes during each exposure. This was observed in some preliminary experiments. This affect was avoided in the experiments represented here by starting all trials with the same mature weight file.

pn1

Agent pn1 uses the two networks described above to make the profit maximizing quantity decision, and in addition uses an adaptive expectations scheme to set the opening price in the new period. This is based on the expected price at the beginning of the last period, and the average price that emerges. The agent adjusts price to the average price of all trades during the period.

Adaptive Expectations Price Rule:

$P^e(t) = hP_{(t-1)} + (1-h)P^e_{(t-1)}, \ 0 <= h <= 1.$

This rule is used only for setting the opening price at the start of each period. The learning rate h was instantiated at 0.5. The price adjustment in each period was based on the average price observed in the period.

pn2

Agent pn2 uses the two networks described above to make the profit maximizing decision, and also incorporates a third network to compute the expected price in the next period. The price net has 6 inputs, one output, and 4 elements in the hidden layer. The inputs and outputs used by the price estimation network are shown in Table IV. This network uses a sigmoid transfer function in the network elements, and is trained 100 times at the end of each period. (i.e., 100 passes through the data).

TABLE IV

INPUTS AND OUTPUTS FOR NET 3

Inputs(Time=t-1)	Outputs (Time = t)
Number of Producers	Expected Price
Number of Consumers	
Times per period	
Period	
Round	
Total Quantity in Market	

CHAPTER VI

VERIFICATION OF THE DISTRIBUTED MARKET

The computer market was verified by performing a number of control market simulations. The control experiments were designed to imitate the well known static and adaptive expectations models of price formation. The purpose of these tests was to verify that the programs comprising the computer market come to the results predicted by the theoretical models.

The market simulation model is a distributed model composed of control and utility programs, several input and output files, and the agent programs as shown below in Table V.

TABLE V

PROGRAMS AND FILES IN THE DISTRIBUTED COMPUTER MARKET

CONTROL	UTILITY	INPUT	OUTPUT	AGENTS
t.exe	t1.exe	agents.	trades.txt	prod03.txt
	update.exe	market.	ave.txt	cons03.txt
	record.exe		bb.txt	
			hist.txt	

Assumptions regarding the behavior of agents in static and adaptive expectations models are quite restrictive and are dropped in the following research. They are employed here only for testing the accuracy and robustness of the market mechanism under various conditions. In this case, the market is composed of identical producers and consumers. The producers make a production decision based on the price of goods in the previous period, while consumers base their consumption decisions on the price in the current period. However, the price in the current period depends on the quantity available. The way that the price is set in the control models is distinctly different from the price formation in my research models. Traditional models do not investigate the behavior that results in the emergence of this important variable. Rather, traditional models simply assume that the price is automatically and instantaneously adjusted to correctly ration the quantity of goods that is available. This is the assumption that is used in the present control simulations.

In the control experiments, the agent's price adjustments are made immediately in a single time step. No trading takes place until the equilibrium price for the given quantity of goods is reached. As a result, all goods are actually sold in each period.

The parameters that were adjusted within the agent programs were the slope of the supply curve in the consumer programs, and the learning rate in the producer programs. Characteristics of the market, such as the number of agents in the market, and the length of market trading were also varied. The decision equations used in the programs are shown below:

Supply Quantity: $q_s = 20 + 2P^e$

Expected Price in next period: $P^e_{t+1} = P_t\, h + (1\text{-}h)\, P^e_t$

Demand Quantity: $q_d = 100 - mP_t$

Market: $q_d = q_s$

In these equations, q_s and q_d signify the quantity supplied and demanded, P and P_e represent the actual and expected prices, while h and m are parameters representing the learning rate and slope of the demand curve respectively. The parameters (h and m) are internal constants that are stored in the compiled agent programs in contrast to the quantity and price variables that are computed by the interaction of the agents.

Given the assumptions above, and values of h and m, the equilibrium price is given by:

$$P^* = 80 / (m + 3h - 1)$$

The price in the market at any given time by can be computed if the initial price P_0 is known. In the control experiments, the initial price P_0 was set to 27.0 or 30.0 and h was set to 1.0 for the static expectations model. Under these conditions the price at time t is computed as:

$$P_t = P_0 - P^*(-2/m)_t + P^*$$

A market composed of these agents was tested and found to reproduce the theoretical results under a variety of conditions as shown in Table VI. The market was tested over various time periods with up to 500 time steps. Figure 15 graphically shows that the market converges to the equilibrium price when the parameters m and h equal 2.1 and 1.0, respectively. The market oscillates predictably toward the equilibrium price. In the cases where the market does reach equilibrium, the equilibrium is stable and there is no tendency to deviate from it. The market was found to perform well with up to 970 agents, however, the data produced by such large numbers of agents is overwhelming. In order to minimize the data collection effort, the file hist.txt was not generated for lengthy simulations comprising large numbers of agents.

TABLE VI

MODEL VERIFICATION WITH AGENTS PROD03 AND CONS03

Description	m1 bb1	m1 bb2	m1 bb3	m1 bb4	m1 bb5	m1 bb6	m1 bb7
Demand Slope (m)	2.1	2.1	2.1	2.1	2.1	2.1	2.1
Learning Rate (h)	1	1	1	1	1	1	1
Number of Producers	1	1	16	4	64	256	485
Number of Consumers	1	1	16	4	64	256	485
Total Agents	2	2	32	8	128	512	970
Time Steps per Period	1	1	1	1	1	1	1
Periods per Round	1	1	1	1	1	1	1
Rounds	200	500	100	200	50	50	25
Price in Final Round	19.51	19.51	19.57	19.51	20.17	20.17	17.30
Quantity in Final Round	59.02	59.02	58.90	59.02	57.65	57.65	63.67
Computed Price in Final Round	19.51	19.51	19.57	19.51	20.17	20.17	17.30
Computed Quantity in Final Round	59.02	59.02	58.90	59.02	57.65	57.65	63.67
Computed Equilibrium Price	19.51	19.51	19.51	19.51	19.51	19.51	19.51
Computed Equilibrium Quantity	59.02	59.02	59.02	59.02	59.02	59.02	59.02

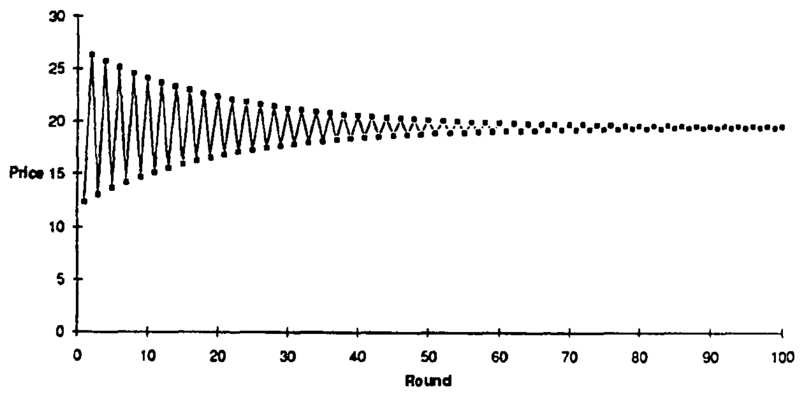

Figure 15. Price vs time outcome for a static expectations market (m=2.1, h=1.0).

As expected, the market converged to equilibrium or diverged according to the relative size of the parameters in the demand and expected price equations. The case m=2.0, h=1 produced stable oscillations around the equilibrium as shown in Figure 16, while m=2.1, h=1 resulted in a gradual approach to equilibrium with prices alternating above and below the equilibrium price (See Figure 15 and Table VII). On the other hand, the market diverged for m > 2.0, h=1 with larger alternating deviations around the equilibrium price until the market collapsed.

TABLE VII

MODEL VERIFICATION WITH CYCLICAL PRICE VARIATION

Description	m2 bb1	m2 bb2	m2 bb3	m2 bb4
Demand Slope (m)	2	2.1	2.6	3
Learning Rate (h)	1	1	1	1
Number of Producers	4	4	4	4
Number of Consumers	4	4	4	4
Total Agents	8	8	8	8
Time Steps per Period	1	1	1	1
Periods per Round	1	1	1	1
Rounds	100	100	100	100
Price in Final Round	30.00	19.57	17.39	16.00
Quantity in Final Round	40.00	58.90	54.78	52.00
Computed Price in Final Round	30.00	19.57	17.39	16.00
Computed Quantity in Final Round	40.00	58.90	54.78	52.00
Computed Equilibrium Price	20.00	19.51	17.39	16.00
Computed Equilibrium Quantity	60.00	59.02	54.78	52.00

A second cause of market collapse was found when the starting price P_0 was set at 30 or above, and the slope of the demand curve was steep. Under these conditions, the producers do not recover sufficient revenue to meet costs and as a result, wealth falls below zero and they are eliminated from the market.

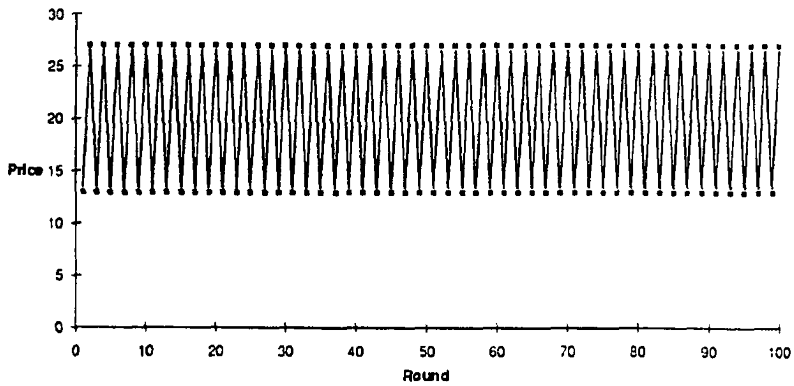

Figure 16. Stable price cycles in the static expectations model (m=2.0, h=1.0).

Table VIII summarizes the results of market simulations when the parameter m was fixed at 2.0 and the learning rate varied over the range 0 <= h <= 1 . The market performed as expected, converging more rapidly as the value of h approached 1/2 from above.

TABLE VIII

MODEL VERIFICATION WITH ADAPTIVE EXPECTATIONS

Description	m3 bb1	m3 bb2	m3 bb3	m3 bb4	m3 bb5
Demand Slope (m)	2	2	2	2	2
Learning Rate (h)	1	0.9	0.6	0.25	0
Number of Producers	4	4	4	4	4
Number of Consumers	4	4	4	4	4
Total Agents	8	8	8	8	8
Time Steps per Period	1	1	1	1	1
Periods per Round	1	1	1	1	1
Rounds	100	100	100	100	100
Price in Final Round	27	20	20	20	10
Quantity in Final Round	46	60	60	60	80
Computed Price in Final Round	30.00	20.00	20.00	20.00	
Computed Quantity in Final Round	40.00	60.00	60.00	60.00	100.00
Computed Equilibrium Price	20.00	20.00	20.00	20.00	20.00
Computed Equilibrium Quantity	60.00	60.00	60.00	60.00	60.00

These markets approach price equilibrium by alternating above and below the equilibrium price. Values of h such that $0 < h < 1/2$ approach equilibrium from below, and never overestimate price, as shown in Figure 17.

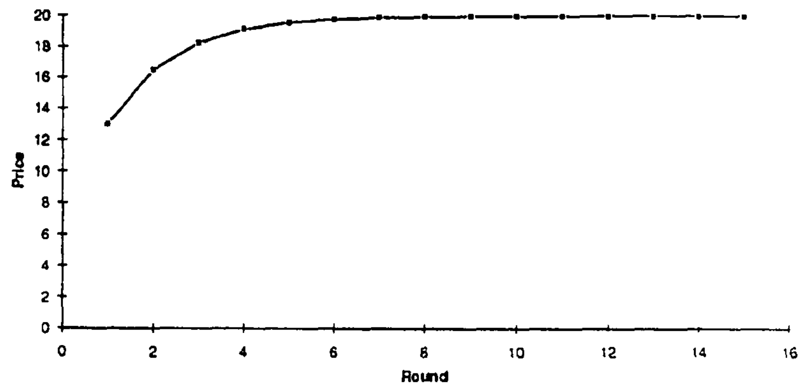

Figure 17. Model verification with adaptive expectations (m=2.0, h=0.25).

In conclusion, the simulated market produced the theoretically expected output in every case where sufficient time was allowed for the market to come to equilibrium. The problems encountered were operational rather than theoretical. Markets composed of over 100 agents generate large amounts of data and it was determined that the file hist.txt could be eliminated unless specifically needed to examine the trading behavior of an individual agent. Additionally, larger markets take a proportionately longer number of periods to come to equilibrium. It was found to be especially important to save the file bb.txt, because this file summarizes the market and records the results as the market closes.

PRELIMINARY EXPERIMENTS WITH STACKELBERG AGENTS

One set of preliminary experiments focussed on the appropriate number of consumers to include in the markets. Table IX shows the theoretical and experimental results of a market composed of 4 producers of the Stackelberg class. The CV for these agents was 0.0, making them true Cournot strategists.

TABLE IX

STACKELBERG THEORETICAL AND REALIZED OUTCOMES
CV = 0.0

Number of Consumers	Theory Quantity	Realized Quantity	Theory Price	Realized Price
4	1,440	1,440	14.0	13.8
8	2,880	2,875	14.0	13.8
16	5,760	5,761	14.0	13.9
32	11,520	11,478	14.0	14.01
64	23,040	21,660	14.0	16.06
128	46,080	34,810	14.0	22.77

The agents were allowed to adjust price for a maximum of 5 time steps in each period. Each round included 4 periods, and the market was conducted for 90 rounds, yielding a total of 360 periods. In all cases, the markets were homogeneous, and all agents were started with 1,000 units of wealth. At the end of 360 periods the markets were discontinued, and the results examined. The markets composed of 64 and 128 agents had not come to equilibrium, and the price was still falling. The markets composed of fewer agents showed some consistent variation around an apparent equilibrium price. The values reported were the average price and quantity traded in the last round, and as a result, the figures may diverge somewhat from the expected average price of 14.0. The quantities traded in the market are also very close to the theoretical quantities.

These results were used as a guide for choosing the market structure for further experimentation. Unless otherwise noted, all subsequent markets included 4 producers and 16 consumers. This structure resulted in a price and quantity equilibrium within 90 rounds for the Stackelberg agents. However, subsequent results will show that this was not adequate for all classes of agents, and that some markets may never come to equilibrium. The 16 consumer agents used in these markets differ only in the coefficients that are assigned to them, so the agents are not completely identical. This procedure may result in small quantities of goods left on the market. Briefly, the agent chooses the largest quantity available, at the lowest price. As a result, this agent can be viewed as a close approximation to an optimizing consumer.

CHAPTER VII

DISTRIBUTED MARKET EXPERIMENTS

INTRODUCTION

These experiments show the strengths of the distributed approach for modelling economic systems. The effect of individual agents and classes of agents can be tested in artificial markets composed by the researcher. These experiments show that variety in agents and their strategies is an important component of markets. The experiments also show the importance of learning, knowledge, and the advantage of incorporating more complex agents that are able to optimize by learning about the other agents in the market.

The three classes of agents used for the distributed market simulations are: Stackelberg agents, satisficing agents and optimizing neural agents (hereafter called neural agents). Each class of agents employs distinctly different methods to make price and quantity decisions, and each produces a distinctly different outcome.

CRITERIA FOR EVALUATION

The focus of this study is on the attributes of the emergent variables such as wealth, price, quantity, and their values in distributed economic markets. The attributes of interest are the amount, variability and stationarity of the emergent time series. Total market quantity traded per period, as well as the quantities accruing to individual agents are of interest. That is, the values at both the unit and the subunit level are of interest.

Wealth

All agents (both consumers and producers) enter the market with 1,000 units of wealth. For the markets reported here, this amounts to 20,000 units of total wealth at the market's inception. The design of both consumers and producers is to increase wealth by production and consumption, respectively. Producer's intend to trade their product for more than the cost of producing it, thereby increasing their wealth. The profits from each trade add to the producer's wealth, while losses from trade reduce the producer's wealth. Consumers, on the other hand, attempt to pay less for the product than it is worth to them, and gain utility by consumption.

For the consumer, wealth is a proxy for utility, and measures how well off the agents are. A market that generates more wealth is deemed to be superior to a market that generates less wealth, regardless of how the wealth is distributed.

RESULTS

The results of the market experiments are reported for the markets as a whole, and for the agents individually. The general results are reported first, followed by the results emerging for each class of agents. Statistical properties of the markets, such as stationarity are also discussed. A separate set of experiments reports on the results of repetitions of the same market, with and without agents that have the ability to carry over information between markets. In these experiments, markets containing the same agents are repeated multiple times in order to monitor the variability of results, and the ability of the agents to adapt to slightly different conditions. Another important test of adaptability includes shifts in the demand curve of the consumer. Together, these experiments provide evidence about learning and the characteristics of these experimental markets. The discussion of results ends with remarks about the interaction of agents in the markets and the accumulation of wealth over time.

Table X summarizes the results of the market simulations, including the homogeneous and heterogeneous markets. The 11 homogeneous markets are identified as H1 through H11, and indicate that all producers in the market were identical, except for minor variations in the coefficients. The producing agents that are included in the markets are listed across each row of Table X under the heading "Agents". For instance, market H1 is a homogeneous market and includes four identical agents (pc000). The 31 mixed markets are indicated by M1 through M31 in the table, and each market includes four different producers. The mixed markets include mixtures of all three classes of agents, satisficing, Stackelberg and optimizing. For instance, M22 includes pc000, ps3, ps7 and pn2.

This table is organized according to the type of market, with homogeneous markets appearing first, followed by the mixed markets. The total accumulated wealth, average price and total quantity in the ending period are shown in the table. Tables XI, XII and XIII show the same data organized according to increasing total wealth, average ending price and total quantity traded in the last period, respectively. Many of these markets were still in transition when they were interrupted after 90 rounds.

TABLE X

MARKET SUMMARY

Market	Wealth (Millions)	Average Price	Total Quantity	Agents			
H1	65.40	14.0	5,760	pc000	pc000	pc000	pc000
H2	67.93	17.2	5,234	pc050	pc050	pc050	pc050
H3	65.20	20.0	4,797	pc100	pc100	pc100	pc100
H4	34.18	4.9	6,291	ps1	ps1	ps1	ps1
H5	1.94	26.1	120	ps2	ps2	ps2	ps2
H6	12.14	45.1	498	ps3	ps3	ps3	ps3
H7	2.92	48.5	120	ps7	ps7	ps7	ps7
H8	10.61	45.8	468	ps8	ps8	ps8	ps8
H9	64.97	14.2	5,735	pn2	pn2	pn2	pn2
H10	69.49	15.7	5,490	pc025	pc025	pc025	pc025
H11	2.46	23.7	139	pn1	pn1	pn1	pn1
M1	65.15	16.3	5,362	pc000	pc025	pc050	pc100
M2	18.59	42.3	1,112	ps1	ps2	ps3	ps7
M3	14.74	45.9	707	ps1	ps2	ps3	ps8
M4	19.49	42.5	1,139	ps1	ps2	ps7	ps8
M5	16.30	43.5	923	ps1	ps7	ps8	ps3
M6	15.20	42.9	926	ps7	ps8	ps2	ps3
M7	62.48	19.1	4,914	pc000	ps1	ps2	ps3
M8	59.86	21.8	4,486	pc000	ps1	ps2	ps7
M9	55.39	25.8	3,841	pc000	ps1	ps2	ps8
M10	64.20	17.3	5,242	pc000	ps1	ps7	ps3
M11	63.22	13.5	5,868	pc000	ps1	ps7	ps8
M12	63.45	16.7	5,305	pc000	ps7	ps2	ps3
M13	64.02	14.5	5,681	pc000	ps7	ps2	ps8
M14	63.68	17.2	5,223	pc000	ps7	ps3	ps8
M15	53.95	17.4	5,203	ps1	ps2	ps3	pn2
M16	50.82	14.5	5,660	ps1	ps2	ps7	pn2
M17	58.38	22.1	4,446	ps1	ps2	ps8	pn2
M18	60.10	6.6	6,952	ps1	ps7	ps3	pn2
M19	49.81	15.2	5,557	ps1	ps7	ps8	pn2
M20	45.49	19.7	4,844	ps7	ps2	ps3	pn2
M21	59.27	9.5	6,475	ps8	ps7	ps3	pn2
M22	66.80	11.5	6,172	pc000	ps7	ps3	pn2
M23	63.42	5.9	7,037	pc000	ps7	ps2	pn2
M24	63.57	17.7	5,138	pc000	ps7	ps1	pn2
M25	68.62	5.4	7,139	pc000	ps7	ps8	pn2
M26	61.69	27.7	3,547	pc000	ps1	ps2	pn2
M27	64.88	5.0	7,193	pc000	ps1	ps3	pn2
M28	64.28	6.7	6,939	pc000	ps1	ps8	pn2
M29	63.51	22.9	4,308	pc000	ps2	ps8	pn2
M30	66.10	16.2	5,395	pc000	ps2	ps7	pn2
M31	65.02	18.6	5,024	pc000	ps3	ps8	pn2

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

TABLE XI

MARKET SUMMARY SORTED BY WEALTH

Market	Wealth (Millions)	Average Price	Total Quantity	Agents			
H5	1.94	26.1	120	ps2	ps2	ps2	ps2
H11	2.46	23.7	139	pn1	pn1	pn1	pn1
H7	2.92	48.5	120	ps7	ps7	ps7	ps7
H8	10.61	45.8	468	ps8	ps8	ps8	ps8
H6	12.14	45.1	498	ps3	ps3	ps3	ps3
M3	14.74	45.9	707	ps1	ps2	ps3	ps8
M6	15.20	42.9	926	ps7	ps8	ps2	ps3
M5	16.30	43.5	923	ps1	ps7	ps8	ps3
M2	18.59	42.3	1,112	ps1	ps2	ps3	ps7
M4	19.49	42.5	1,139	ps1	ps2	ps7	ps8
H4	34.18	4.9	6,291	ps1	ps1	ps1	ps1
M20	45.49	19.7	4,844	ps7	ps2	ps3	pn2
M19	49.81	15.2	5,557	ps1	ps7	ps8	pn2
M16	50.82	14.5	5,660	ps1	ps2	ps7	pn2
M15	53.95	17.4	5,203	ps1	ps2	ps3	pn2
M9	55.39	25.8	3,841	pc000	ps1	ps2	ps8
M17	58.38	22.1	4,446	ps1	ps2	ps8	pn2
M21	59.27	9.5	6,475	ps8	ps7	ps3	pn2
M8	59.86	21.8	4,486	pc000	ps1	ps2	ps7
M18	60.10	6.6	6,952	ps1	ps7	ps3	pn2
M26	61.69	27.7	3,547	pc000	ps1	ps2	pn2
M7	62.48	19.1	4,914	pc000	ps1	ps2	ps3
M11	63.22	13.5	5,868	pc000	ps1	ps7	ps8
M23	63.42	5.9	7,037	pc000	ps7	ps2	pn2
M12	63.45	16.7	5,305	pc000	ps7	ps2	ps3
M29	63.51	22.9	4,308	pc000	ps2	ps8	pn2
M24	63.57	17.7	5,138	pc000	ps7	ps1	pn2
M14	63.68	17.2	5,223	pc000	ps7	ps3	ps8
M13	64.02	14.5	5,681	pc000	ps7	ps2	ps8
M10	64.20	17.3	5,242	pc000	ps1	ps7	ps3
M28	64.28	6.7	6,939	pc000	ps1	ps8	pn2
M27	64.88	5.0	7,193	pc000	ps1	ps3	pn2
H9	64.97	14.2	5,735	pn2	pn2	pn2	pn2
M31	65.02	18.6	5,024	pc000	ps3	ps8	pn2
M1	65.15	16.3	5,362	pc000	pc025	pc050	pc100
H3	65.20	20.0	4,797	pc100	pc100	pc100	pc100
H1	65.40	14.0	5,760	pc000	pc000	pc000	pc000
M30	66.10	16.2	5,395	pc000	ps2	ps7	pn2
M22	66.80	11.5	6,172	pc000	ps7	ps3	pn2
H2	67.93	17.2	5,234	pc050	pc050	pc050	pc050
M25	68.62	5.4	7,139	pc000	ps7	ps8	pn2
H10	69.49	15.7	5,490	pc025	pc025	pc025	pc025

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

TABLE XII

MARKET SUMMARY SORTED BY PRICE

Market	Wealth (Millions)	Average Price	Total Quantity	Agents			
H4	34.18	4.9	6,291	ps1	ps1	ps1	ps1
M27	64.88	5.0	7,193	pc000	ps1	ps3	pn2
M25	68.62	5.4	7,139	pc000	ps7	ps8	pn2
M23	63.42	5.9	7,037	pc000	ps7	ps2	pn2
M18	60.10	6.6	6,952	ps1	ps7	ps3	pn2
M28	64.28	6.7	6,939	pc000	ps1	ps8	pn2
M21	59.27	9.5	6,475	ps8	ps7	ps3	pn2
M22	66.80	11.5	6,172	pc000	ps7	ps3	pn2
M11	63.22	13.5	5,868	pc000	ps1	ps7	ps8
H1	65.40	14.0	5,760	pc000	pc000	pc000	pc000
H9	64.97	14.2	5,735	pn2	pn2	pn2	pn2
M16	50.82	14.5	5,660	ps1	ps2	ps7	pn2
M13	64.02	14.5	5,681	pc000	ps7	ps2	ps8
M19	49.81	15.2	5,557	ps1	ps7	ps8	pn2
H10	69.49	15.7	5,490	pc025	pc025	pc025	pc025
M30	66.10	16.2	5,395	pc000	ps2	ps7	pn2
M1	65.15	16.3	5,362	pc000	pc025	pc050	pc100
M12	63.45	16.7	5,305	pc000	ps7	ps2	ps3
M14	63.68	17.2	5,223	pc000	ps7	ps3	ps8
H2	67.93	17.2	5,234	pc050	pc050	pc050	pc050
M10	64.20	17.3	5,242	pc000	ps1	ps7	ps3
M15	53.95	17.4	5,203	ps1	ps2	ps3	pn2
M24	63.57	17.7	5,138	pc000	ps7	ps1	pn2
M31	65.02	18.6	5,024	pc000	ps3	ps8	pn2
M7	62.48	19.1	4,914	pc000	ps1	ps2	ps3
M20	45.49	19.7	4,844	ps7	ps2	ps3	pn2
H3	65.20	20.0	4,797	pc100	pc100	pc100	pc100
M8	59.86	21.8	4,486	pc000	ps1	ps2	ps7
M17	58.38	22.1	4,446	ps1	ps2	ps8	pn2
M29	63.51	22.9	4,308	pc000	ps2	ps8	pn2
H11	2.46	23.7	139	pn1	pn1	pn1	pn1
M9	55.39	25.8	3,841	pc000	ps1	ps2	ps8
H5	1.94	26.1	120	ps2	ps2	ps2	ps2
M26	61.69	27.7	3,547	pc000	ps1	ps2	pn2
M2	18.59	42.3	1,112	ps1	ps2	ps3	ps7
M4	19.49	42.5	1,139	ps1	ps2	ps7	ps8
M6	15.20	42.9	926	ps7	ps8	ps2	ps3
M5	16.30	43.5	923	ps1	ps7	ps8	ps3
H6	12.14	45.1	498	ps3	ps3	ps3	ps3
H8	10.61	45.8	468	ps8	ps8	ps8	ps8
M3	14.74	45.9	707	ps1	ps2	ps3	ps8
H7	2.92	48.5	120	ps7	ps7	ps7	ps7

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

TABLE XIII

MARKET SUMMARY SORTED BY QUANTITY

Market	Wealth (Millions)	Average Price	Total Quantity	Agents			
H5	1.94	26.1	120	ps2	ps2	ps2	ps2
H7	2.92	48.5	120	ps7	ps7	ps7	ps7
H11	2.46	23.7	139	pn1	pn1	pn1	pn1
H8	10.61	45.8	468	ps8	ps8	ps8	ps8
H6	12.14	45.1	498	ps3	ps3	ps3	ps3
M3	14.74	45.9	707	ps1	ps2	ps3	ps8
M5	16.30	43.5	923	ps1	ps7	ps8	ps3
M6	15.20	42.9	926	ps7	ps8	ps2	ps3
M2	18.59	42.3	1,112	ps1	ps2	ps3	ps7
M4	19.49	42.5	1,139	ps1	ps2	ps7	ps8
M26	61.69	27.7	3,547	pc000	ps1	ps2	pn2
M9	55.39	25.8	3,841	pc000	ps1	ps2	ps8
M29	63.51	22.9	4,308	pc000	ps2	ps8	pn2
M17	58.38	22.1	4,446	ps1	ps2	ps8	pn2
M8	59.86	21.8	4,486	pc000	ps1	ps2	ps7
H3	65.20	20.0	4,797	pc100	pc100	pc100	pc100
M20	45.49	19.7	4,844	ps7	ps2	ps3	pn2
M7	62.48	19.1	4,914	pc000	ps1	ps2	ps3
M31	65.02	18.6	5,024	pc000	ps3	ps8	pn2
M24	63.57	17.7	5,138	pc000	ps7	ps1	pn2
M15	53.95	17.4	5,203	ps1	ps2	ps3	pn2
M14	63.68	17.2	5,223	pc000	ps7	ps3	ps8
H2	67.93	17.2	5,234	pc050	pc050	pc050	pc050
M10	64.20	17.3	5,242	pc000	ps1	ps7	ps3
M12	63.45	16.7	5,305	pc000	ps7	ps2	ps3
M1	65.15	16.3	5,362	pc000	pc025	pc050	pc100
M30	66.10	16.2	5,395	pc000	ps2	ps7	pn2
H10	69.49	15.7	5,490	pc025	pc025	pc025	pc025
M19	49.81	15.2	5,557	ps1	ps7	ps8	pn2
M16	50.82	14.5	5,660	ps1	ps2	ps7	pn2
M13	64.02	14.5	5,681	pc000	ps7	ps2	ps8
H9	64.97	14.2	5,735	pn2	pn2	pn2	pn2
H1	65.40	14.0	5,760	pc000	pc000	pc000	pc000
M11	63.22	13.5	5,868	pc000	ps1	ps7	ps8
M22	66.80	11.5	6,172	pc000	ps7	ps3	pn2
H4	34.18	4.9	6,291	ps1	ps1	ps1	ps1
M21	59.27	9.5	6,475	ps8	ps7	ps3	pn2
M28	64.28	6.7	6,939	pc000	ps1	ps8	pn2
M18	60.10	6.6	6,952	ps1	ps7	ps3	pn2
M23	63.42	5.9	7,037	pc000	ps7	ps2	pn2
M25	68.62	5.4	7,139	pc000	ps7	ps8	pn2
M27	64.88	5.0	7,193	pc000	ps1	ps3	pn2

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

Each class of markets results in a different pattern of results. In general, the Stackelberg agents produce more wealth and result in higher joint production than the homogeneous satisficing markets. This is reasonable because these agents have far more knowledge about the market. The satisficing agents do not have knowledge about the markets and produce low market wealth in homogeneous markets. The optimizing agent pn2 also does well using total market wealth as a criteria, but pn1 does not.

The mixed markets were composed of agents from each of the classes. Several tests included mixtures of satisficing agents. In addition, one mixed market was composed of only Stackelberg agents. The other mixed markets included combinations of all three classes. The Stackelberg agent pc000 was used in all tests to represent the Stackelberg agents because agents in this class were very consistent in operation. The optimizing class was represented by pn2 because this agent's performance was generally superior to agent pn1. The difference between these agents was the inclusion of a third network in pn2 for price estimation.

One feature that appears in the results shown in Tables XI , XII and XIII, is a large discontinuity that separates the results into distinct groups. These discontinuities are shown more clearly when the data are shown graphically (See Figures 18, 19,and 20). These figures show the results sorted by total wealth, price and ending quantity. In the case of total wealth, the low wealth markets are, with a single exception, composed of satisficing agents. The exception was neural agent pn1, which also produced very low market wealth. Total wealth jumps from 34.18 million units to 45.49 million units between consecutive markets when the markets are ordered according to increasing wealth. All markets in the high wealth group include at least one representative from either the Stackelberg or the optimizing class (pn2), while the low wealth group contains mostly satisficing agents.

The highest wealth was produced by the homogeneous market composed entirely of Stackelberg agent pc025. However, many combinations of satisficing, optimizing and Stackelberg agents produce similar values. Markets with at least one Stackelberg or optimizing agent usually produce more wealth than markets composed only of satisficing agents (See Figure 18).

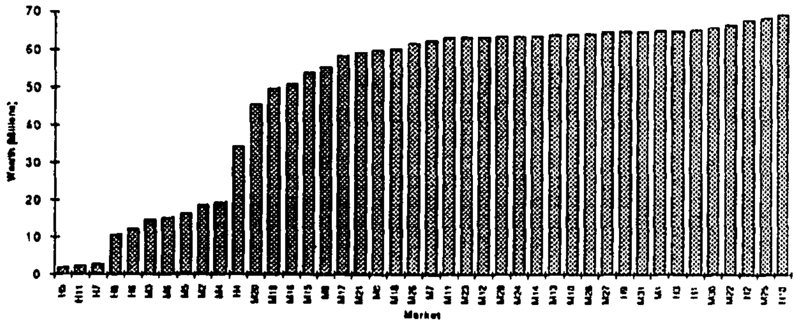

Figure 18. Markets sorted by wealth.

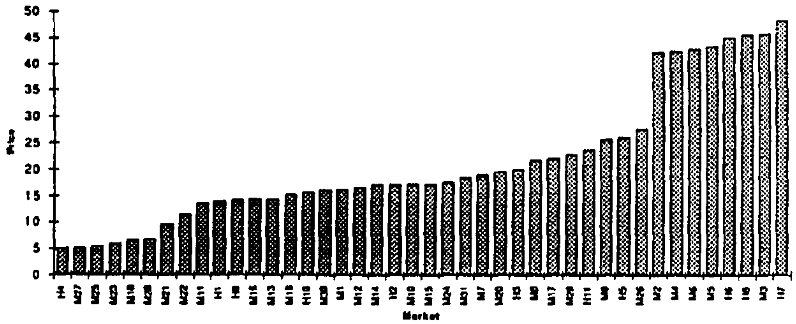

Figure 19. Markets sorted by price.

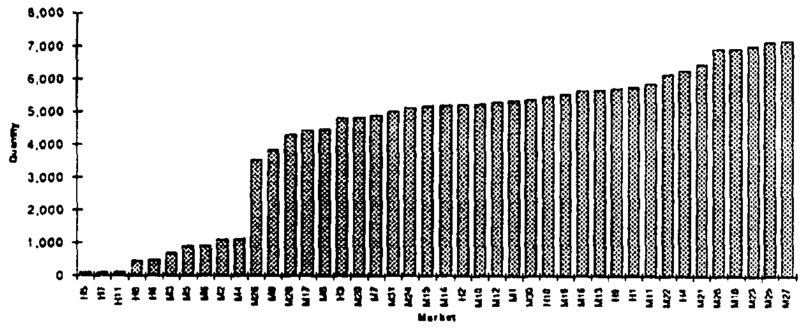

Figure 20. Markets sorted by quantity.

The quantity traded results are consistent with the results reported above for total wealth. There is a clear separation between the market composed entirely of satisficing agents and the other markets when the markets are ordered according to increasing quantity. The quantity traded more than doubles, jumping from 1,139 to 3,547 between neighboring markets in the sorted list. Again, the optimizing agent pn1 scored with the satisficing agents. The markets that produce the highest quantity were mixed markets containing Stackelberg, neural and satisficing agents.

The ordered price results tell a similar story. In this case, the price jumps from 27.7 to 42.3 between neighboring markets in the sorted list. The high price markets are exclusively composed of satisficing agents. However, the mid-range price results also contain a market composed only of satisficing agents, and the lowest price emerging from the simulations was from a market composed of satisficing agents (the price slashing agents ps1).

Results for Neural Agents

The two homogeneous markets composed of neural agents came to very different solutions. Agent pn1 performed poorly while agent pn2 was among the strongest agents. The difference between these neural agents was that an extra neural network was included in agent pn2 for computing the market opening price. The distinct difference in the results emerging from markets containing these agents shows the importance of accurate price forecasts in competitive markets.

The maturity and experience of the agent have a major impact on how the agent responds to various markets. For instance, Figure 21 shows a homogeneous market consisting of identical pn2 agents, all starting with identical weight files. After a transient period at the market's inception, the agents follow different paths. Figure 22 also illustrates a market composed of ps2 agents, but only one of the agents has a mature weight file (trace 4). Each of the other pn2 agents has a set of newly initialized weight files at the start of the market. As the agents learn by adjusting their weights, the resulting trace shows large triangular patterns, eventually evolving into smaller adjustments as the agents learn. The large triangular patterns at the beginning of the new market make the pn2 agent easy to identify in a mixed market.

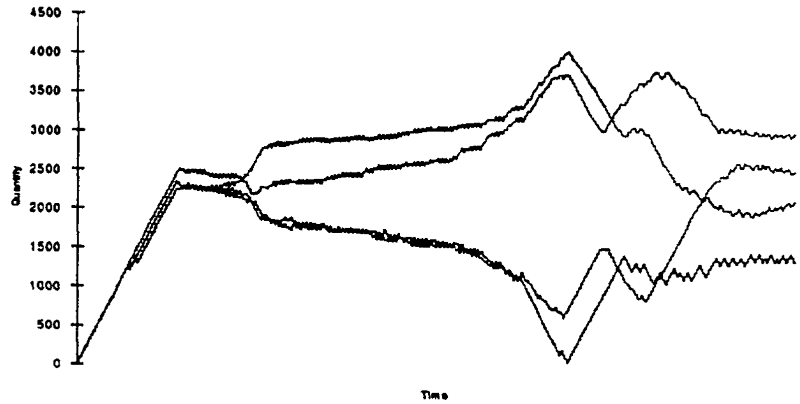

Figure 21. Output from market with identical neural agents.

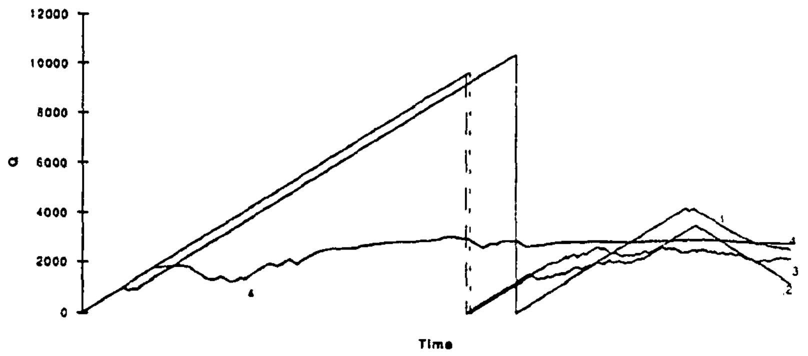

Figure 22. Output from market with different neural agents. Traces 1,2,&3 are newly initialized neural agents. Trace 4 is from a mature neural agent.

The neural agents have an important place in the experimental markets because markets containing at least one neural agent tend to produce considerably more wealth than markets composed of only satisficing agents. The neural agents appear to alter the characteristics of the market because of the large adjustments that are made as the market unfolds. Because the agents continue to evolve, the markets in which they participate may wander and not settle down to a fixed point equilibrium. The markets containing pn2 produce wealth comparable to the Stackelberg agents who have knowledge of the demand curve, however, the mature neural agent pn2 dominates most markets where it is the only non-satisficing agent. At times this agent tends to increase production to catastrophic levels, followed by an abrupt collapse in production. This behavior does not appear in all cases, but when it does, it may persist without sign of extinction. This appears to be similar to the results that sometimes appear in real world markets, indicating that instability may result from the interaction of particular combinations of strategies in the market. The interaction of these strategies may lead to irregular periods (I hesitate to call them cycles because of their nonperiodic nature) of overproduction and underproduction. The neural agent is generally superior to the satisficing agents in these markets. The performance (accumulated wealth) of all agents is improved by including the neural agent. It is also important to remember that the markets are stochastic in respect to the internal coefficients, and may not always produce identical results.

Results for Satisficing Agents

Markets composed of only satisficing agents tend to have very high prices and produce low quantities, so that the resulting wealth is also very low. The poor performance of the satisficing agents is a clear indication that following rigid rules may lead to less than optimal performance in markets of this type. The satisficing agents do retain enough wealth to survive, but reach equilibrium that is far from optimal. These markets can usually be recognized by abrupt adjustments in the total quantity traded series. (See Figure 23). The clear separation between the satisficing agents and others was discussed above.

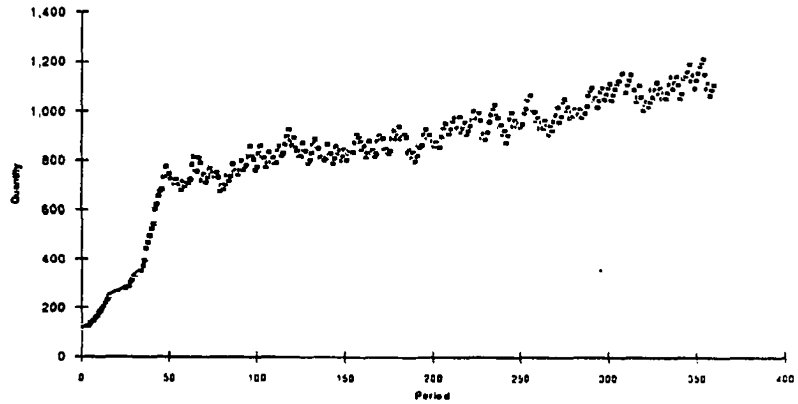

Figure 23. Total quantity traded per period in market M2.

ps1. The distinguishing feature of the ps1 strategy is the underbidding price adjustment rule. The homogeneous market containing only ps1 rapidly falls to marginal cost because the agents continue to underbid each other. The total quantity series shows an abrupt change where one agent rapidly increases production, while the other agents drop out of the market. In combination with other types of agents, ps1 frequently produces markets that are erratic. These markets show that the quantity traded in each period makes sharp and abrupt changes as a result of the price cutting behavior of ps1. Additionally, these results point out the importance of a single agent's strategy to the other agents, and to market stability indicated by abrupt changes in the levels of emergent variables.

ps2. This agent bases price decisions on the average price that emerges in the previous period. As a result, it does not have the means to adjust price at all in a homogeneous market. This feature makes this agent poorly suited to competing against agents that use the same strategy. The failure of such agents in homogeneous markets is not uncommon in these tests. Some strategies succeed only when there is sufficient variety in the market. Variety of this type can be introduced through adding a stochastic noise term to the agents, but this is an artifact of the experimental market.

Even when matched against other strategies, ps2 appears near the bottom in terms of production and share of wealth. This agent does not do well in combination with other agents or independently. Additionally, ps2 does not have a discernable impact on the total quantity traded series.

ps3. Agent ps3 produces irregular results in two ways. This agent makes comparatively large adjustments to price and quantity, which appear in the price and quantity traces. Additionally, the agent is sometimes very successful, and at other times fails. The agent may temporarily be successful, only to precipitously drop to the bottom of the market without warning. The success of this agent appears to depend on the other agents (the milieu) in the market which it exists. In a homogeneous market composed of ps3 agents, each agent temporarily dominates production. Large adjustments in production are sometimes apparent in the trace of total quantity traded per period. The performance of this agent is better than the similar agents ps1 and ps2. Recall that the difference between these agents is in the opening price bid. The high bid strategy of ps3 appears to improve performance and indicates that errors in forecasting price are more acceptable if they are biased upward. Agents ps4, ps5 and ps6 were eliminated in preliminary testing and were not included in the final testing.

ps7. The strategy used by agent ps7 for quantity adjustment makes it dependent on the other agents in the market. This agent does not perform well in homogeneous markets. However, in markets showing a greater variety of strategies, this agent does well. It attempts to capture a "fair share" of the market. This strategy keeps this agent from extreme high or low production, and is a means of reducing risk. However, this also eliminates the opportunity to earn high profits. This agent rides along with the market, but will not make the market. In terms of performance within a variety of markets, agent ps7 is superior to most of the other satisficing strategies.

ps8. ps8 is an agent that makes large adjustments and continually searches for better results. This agent does well in some markets but poorly in others for no apparent reason. In the homogeneous market, one of the four ps8 agents dominates the market and accounts for nearly all of the production. All-or-none outcomes are also observed in other situations. In some cases, this agent leads the market in production, but has a weak price adjustment rule that relies on average market price for its trading. ps8 frequently leads markets composed of satisficing agents, and also does well in the more optimal markets composed of Stackelberg and neural agents.

Results for Stackelberg Agents

The Stackelberg agents are, not surprisingly, strong contenders in these markets. The advantage of knowing the demand curve is large, but not insurmountable. Each of the Stackelberg agents was tested separately in a homogeneous market, and also in a mixed market composed only of Stackelberg agents. Each of the Stackelberg agents produces a smooth trace which is characteristic of that particular agent. For instance, in a mixed market of Stackelberg agents, the agent with CV= 0 (pc000), always produced the highest quantity. Agent pc000 was chosen as the representative of this class of agents and was included in the mixed markets with satisficing and neural agents.

The Stackelberg class agents come to a clearly different and by all standards better result than the homogeneous satisficing agents. This is true for the total wealth produced by the market, as well as the total quantity traded. The total quantity traded was sometimes nearly 2 orders of magnitude greater for Stackelberg agents when compared to satisficing agents.

pc000 added consistency and improved results when it was included in markets containing only satisficing agents. In most cases the wealth of all agents in the market increased when pc000 was substituted for a satisficing agent. The interaction typically improved the results for the satisficing agents, while also producing considerably higher wealth for the consuming agents. pc000 also reacted favorably with the neural agent pn2, but was often surpassed by the neural agent in terms of wealth produced in the market. Combinations of pn2, pc000 and satisficing agents sometimes favored agents from each class. That is, the agents from any given class did not dominate in all markets.

The Stackelberg agents possess knowledge of the demand curve, but are only able to conjecture the actions of their competitors. The knowledge of the demand curve gives these agents a decided advantage when compared to the satisficing agents, but the fixed conjecture about their competition means that these agents do not respond to obvious errors or shortcomings in their weaker competitors. Additionally, assuming the wrong coefficients for the demand curve can have disastrous consequences for an agent.

Stability of the Market Over Time

The experimental markets were tested to find evidence of stability (e.g., random deviations about a fixed point attractor) after a transient period. Initially, the behavior of the markets is analyzed by comparing changes in the mean and the variance over time. This analysis concentrates on the last 100 trading periods in order to eliminate the effects of the transient period that occurs as the market opens. The 100 periods are divided into 3 groups, consisting of 40, 20 and 40 consecutive observations. The central group of 20 observations is discarded, and the two groups of observations at the beginning and end of the sample are compared. These groups are designated "Top 40" and "Bottom 40", respectively.

The variance and mean of the total quantity traded series in the two groups are compared for each market in Table XIV. In the heterogeneous market, the variance of the Top 40 and Bottom 40 groups are constant in about a third of the cases. On the other hand, the means in the two groups differ significantly in most cases, with only 4 of 31 markets failing to show statistically different means. This contrasts with the homogeneous markets, which do not show statistically significant evidence of different variance and means in the two groups (e.g the mixed markets are less likely to be stable and tend not to have random deviations about a fixed mean).

The fact that the first difference of these markets is constant while the series themselves are not, indicates the presence of autocorrelation in the time series that can be removed by differencing. The term "random walk" is typically associated with a process $x_t = x_{t-1} + e_t$, where e_t is a random error. A "white noise" process is associated with a stationary process such that $x_t = e_t$. The white noise process is just the first difference of a random walk process. That is, differencing converts a random walk time series into a stationary, white noise time series. A random walk with drift is very common in actual economic time series and can be written $x_t = x_{t-1} + d + e_t$, where d is a constant drift term. In this case, the time series drifts in a predictable manner which can also be reduced to stationarity by differencing. (See [Pindyck and Rubinfeld, 1981], for more details about white noise, random walks, and stationarity). The autocorrelation functions and tests for stationarity of the markets is described in the next section.

The first difference of the quantity series ($q_t - q_{t-1}$) was also tested in the same manner as described above.

TABLE XIV

MEAN AND VARIANCE OF THE QUANTITY SERIES

Market	Top 40		Bottom 40		Z Test Stable	F Test Stable
	Mean	Variance	Mean	Variance	Mean	Variance
H1	5,759	0	5,760	0		1
H2	5,235	11	5,235	9	1	1
H3	4,794	12	4,794	16	1	1
H4	6,738	4,370	6,406	3,999		1
H5	120	0	120	0	NA	NA
H6	510	4,759	517	4,871	1	1
H7	120	0	120	0	NA	NA
H8	467	62	467	62	1	1
H9	5,695	54	5,726	46		1
H10	5,487	10	5,488	8		1
H11	163	1,635	166	1,826	1	1
M1	5,362	15	5,363	14	1	1
M2	1,003	2,414	1,105	2,599		1
M3	702	4,717	765	4,697		1
M4	1,037	1,391	1,085	1,256		1
M5	807	2,066	895	1,113		
M6	751	978	844	1,721		
M7	4,970	3,710	4,971	1,699	1	
M8	4,355	781	4,463	896		1
M9	3,752	310	3,846	144		
M10	5,276	1,533	5,136	1,717		1
M11	5,502	7,050	5,747	6,782		1
M12	5,358	6,971	5,143	9,923		1
M13	5,539	5,506	5,622	1,505		
M14	5,355	12,164	5,333	5,223	1	
M15	4,928	1,731	5,126	1,542		1
M16	5,138	6,949	5,531	5,149		1
M17	2,834	137,645	4,114	8,271		
M18	6,415	9,142	6,835	4,577		
M19	5,015	7,313	5,423	5,591		1
M20	4,301	7,123	4,708	5,622		1
M21	5,907	10,745	6,366	6,116		
M22	5,436	4,274	5,750	55,283		
M23	7,209	1	6,921	43,428		
M24	5,245	585,779	5,617	1,069,771		
M25	6,459	55,509	7,140	3,804		
M26	6,388	331,662	4,982	2,390,313		
M27	4,128	393,053	7,140	15,954		
M28	6,986	18,575	7,098	5,062		
M29	4,334	166,094	4,072	12,407		
M30	5,312	907,209	5,282	23,336	1	
M31	6,398	905	5,657	176,878		

TABLE XV

FIRST DIFFERENCE OF THE QUANTITY SERIES

Market	TOP 40		BOTTOM 40		Z Test Stable Mean	F Test Stable Variance
	Average	Variance	Average	Variance		
H1	0.03	0	0.00	0	1	
H2	-0.07	23	-0.12	16	1	1
H3	0.01	21	-0.25	31	1	1
H4	-5.67	0	-5.42	0		1
H5	0.00	0	0.00	0	NA	NA
H6	-1.34	1,086	0.99	765	1	1
H7	0.00	0	0.00	0	NA	NA
H8	-0.04	215	-0.04	215	1	1
H9	0.58	9	0.70	11	1	1
H10	-0.14	19	0.11	20	1	1
H11	-2.48	1,681	2.08	2,803	1	1
M1	0.08	23	-0.19	19	1	1
M2	-0.48	1,029	1.52	1,363	1	1
M3	-0.05	616	-4.44	655	1	1
M4	-0.54	1,010	4.06	1,026	1	1
M5	0.68	856	1.39	668	1	1
M6	0.43	644	3.07	793	1	1
M7	4.65	304	-3.37	74		
M8	1.72	128	1.91	167	1	1
M9	1.27	120	0.78	184	1	1
M10	-0.56	594	-2.21	484	1	1
M11	-5.86	190	7.37	220		1
M12	-8.61	976	8.43	1,313		1
M13	-6.15	321	1.21	258		1
M14	12.48	1,179	-1.93	872		1
M15	3.54	4	3.26	3	1	1
M16	7.13	1	6.21	0		1
M17	-23.92	2,621	11.75	442		
M18	8.26	11	5.90	5		
M19	7.36	1	6.42	0		1
M20	7.28	0	6.44	0		1
M21	8.56	35	7.03	27	1	1
M22	3.92	556	16.42	1,556		
M23	0.08	0	-10.05	1,783	1	
M24	67.28	732	-18.32	185,831	1	
M25	29.28	1,402	5.44	476		
M26	-13.56	122,865	-54.13	388,347	1	
M27	39.81	1,883	19.85	1,207		1
M28	15.80	206	-5.34	87		
M29	-37.21	397	8.58	58		
M30	-44.88	169,533	8.20	639	1	
M31	0.05	445	-30.98	504		1

In this case, the variance and mean of the total quantity traded series were constant in most cases for the heterogeneous markets (see Table XV). The exceptions were nearly all markets containing the neural agents. The presence of pn2 results in markets that have quantitatively different properties. The mean and variance of the total quantity traded series are not constant, and they are not changing in a (first order) constant manner (e.g., this is not random walk with drift). In most cases the neural agents made large changes in the quantity produced in these periods, resulting in markets that are unstable in the sense that they do not appear to have a fixed point attractor.

The impact of the neural agent on the market shows that an agent which learns about the market can have a destabilizing influence on the market. The agent continues on the same adjustment path as long as profits continue to increase. But this particular agent abruptly changes production under certain conditions, precipitating a rapid change in the market composition. Sometimes these changes lead to sharp increases or decreases in market production.

Stationarity of the Market Process

The autocorrelation function of the total-quantity-traded time series emerging from each market was examined to determine if the markets were stationary in the last 100 periods before they were interrupted. The autocorrelation function was computed for lags up to 15 periods, and up to the fifth difference (see Table XVI). The results reported here, with a few exceptions, are for the first difference of the total quantity traded series. The exceptions report the autocorrelation of the emergent series for some other lag (usually 2 periods). The sample autocorrelation function was computed for each lag k as described by Pindyck and Rubinfeld [Pindyck and Rubinfeld, 1981]:

$$\rho_k = \frac{cov(y_t, y_{t+k})}{\sigma_y^2}$$

Where y_t represents the total quantity traded in the period, σ is the standard deviation, and k represents the lag. ρ_k was computed for each time series, and its derivatives generated by differencing, up to the fifth difference. It was found that the first difference was usually useful for analysis.

The Q statistic [Box and Pierce], [Pindyck and Rubinfeld], is a joint test that all the autocorrelation coefficients up to k are zero:

$$Q = N \Sigma^k \rho k^2$$

Q is approximately distributed as chi square with k degrees of freedom. Table XVI shows the results of the Q test statistic for each market and the resulting conclusion when the critical value of chi square is 22.31. A one in the table indicates that the null hypothesis $r_1, \ldots, r_{15} = 0$ cannot be rejected at the 90 percent confidence level. Except as noted, the first difference of each series is tested. Several of the homogeneous markets did not vary in the last 100 periods, and the Q statistic is not interpreted in these cases.

Only 12 of the markets were not stationary in the last 100 periods before the markets were interrupted. Statistically speaking, a stationary process is random in the sense that it could have been generated by independently distributed random variables.

TABLE XVI

STATIONARITY OF THE MARKETS

Market	Stationary	Lag	Agents			
M1			pc000	pc025	pc050	pc100
M2	1		ps1	ps2	ps3	ps7
M3	1		ps1	ps2	ps3	ps8
M4	1		ps1	ps2	ps7	ps8
M5	1		ps1	ps7	ps8	ps3
M6	1		ps7	ps8	ps2	ps3
M7	1		pc000	ps1	ps2	ps3
M8			pc000	ps1	ps2	ps7
M9	1		pc000	ps1	ps2	ps8
M10	1		pc000	ps1	ps7	ps3
M11	1	2	pc000	ps1	ps7	ps8
M12	1		pc000	ps7	ps2	ps3
M13	1		pc000	ps7	ps2	ps8
M14	1		pc000	ps7	ps3	ps8
M15	1		ps1	ps2	ps3	pn2
M16	1	2	ps1	ps2	ps7	pn2
M17			ps1	ps2	ps8	pn2
M18	1		ps1	ps7	ps3	pn2
M19	1		ps1	ps7	ps8	pn2
M20			ps7	ps2	ps3	pn2
M21	1		ps8	ps7	ps3	pn2
M22	1		pc000	ps7	ps3	pn2
M23	1	2	pc000	ps7	ps2	pn2
M24	1	2	pc000	ps7	ps1	pn2
M25		2	pc000	ps7	ps8	pn2
M26	1		pc000	ps1	ps2	pn2
M27		2	pc000	ps1	ps3	pn2
M28		2	pc000	ps1	ps8	pn2
M29		2	pc000	ps2	ps8	pn2
M30	1		pc000	ps2	ps7	pn2
M31		2	pc000	ps3	ps8	pn2
H1			pc000	pc000	pc000	pc000
H2	1	0	pc050	pc050	pc050	pc050
H3	1	0	pc100	pc100	pc100	pc100
H4		2	ps1	ps1	ps1	ps1
H5	na		ps2	ps2	ps2	ps2
H6	1	2	ps3	ps3	ps3	ps3
H7	na		ps7	ps7	ps7	ps7
H8		0	ps8	ps8	ps8	ps8
H9			pn2	pn2	pn2	pn2
H10	1	0	pc025	pc025	pc025	pc025
H11			pn1	pn1	pn1	pn1

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

The agent pn2 was present in 7 of the 9 mixed markets that were not stationary. This observation leads to the conclusion that the variability induced by this agent leads to a trend in the markets that is distinguishable from a random walk with an underlying white noise process. The other markets appear to have an underlying white noise processes. The neural agent continues to change its strategy in response to changes by other agents in the market, and as a result, the markets in which it participates may never really approach stationarity. Alternatively satisficing agents such as ps2 make significant changes in several of the markets after periods of relative stability. These markets are punctuated by abrupt changes in production and price, as agents such as ps2 attempt to make corrections for poor profits.

In the homogeneous markets, these stationarity results fall into three groups:

1. Markets that stabilize at a fixed point, and do not vary at all in the last 100 periods. These markets are identified as "na" in Table XIV because the Q statistic cannot be computed when the variance is zero. These markets include the satisficing agents ps2 and ps7.

2. Markets that are stationary over the last 100 rounds. Most markets comprising Stackelberg agents were in this group including: pc025, pc050 and pc100. Additionally, the market comprising satisficing agent ps3 was stationary. In the case of homogeneous markets of Stackelberg agents, the time series of quantity traded per period was stationary, rather than the first difference of the series.

3. Markets that did not show evidence of stationarity over the last 100 periods. These markets include the satisficing agents ps1, and ps8, neural agents pn1, and

pn2, and Stackelberg agent pc000 . Each of these markets has a unique trace that is characteristic of the agent involved.

Mixed Markets With Repetitions

A critical feature of the simulated markets is that they are not exactly repeatable. In other words, the results are not always apparent until they unfold. The markets contain several sources of stochasticity such as the randomly assigned coefficients, and randomization of agents in the queues. Additionally, the neural agent intentionally changes its internal coefficient in response to the market. Thus, the total-quantity-traded trace, for example, does not always follow the same path. In order to determine the consistency and repeatability of the simulated distributed markets, a market consisting of a selection of the more interesting agents was repeated 10 times with and without learning between markets by the neural agent. These markets consisted of the following producers:

pc000	Stackelberg agent with conjectural variation = 0
pn2	Mature neural agent with 3 internal nets
ps3	Satisficing agent with conservative pricing strategy
ps7	Satisficing agent with fair game strategy

The neural agent pn2 used the same mature weight files for each trial (e.g the agent did not learn between markets in the first set of repeated trials).

There are a few factors that vary between markets, for instance, each consumer is randomly assigned unique coefficients for the demand curve that are used throughout the market. The intercept term is 100 +- 0.3 percent, and the slope was -2.0 +- 5.0 percent. This allows a certain amount of variety among otherwise identical agents, but the average demand does not vary between repeated markets. By the same token, the intercept term for the producer's cost curve was 50 +- 0.6 percent and the slope was 5 +- 2.0 percent. Additionally, the initial values of price and quantity were set at 25 +- 5 for the price, and 25 +- 10 for the quantity assigned to each agent. The last variation is the ordering of agents approaching the blackboard in each trading time. The agents are shuffled so that the ordering is random rather than repetitive. If all of these sources of stochasticity are held constant, the markets are completely identical between repetitions.

Table XVII gives the total and private wealth developed in the same mixed market with 10 repetitions. Examination of this table leads to the conclusion that on average, the satisficing agent ps3 performs better than any of the other agents in terms of total wealth accumulated (3.837 million units). The neural agent and Stackelberg agents, accumulated 3.767 and 3.360 million units respectively. These agents did not capture as much wealth as the satisficing agent ps3, even though the Stackelberg agent has knowledge of the demand curve, and the neural agent was capable of learning about the market. An interesting feature of these markets was that the neural agent accumulates more wealth than any other agent in 5 of 10 trials, yet does not accumulate the highest total wealth. This indicates that the variability of returns plays an important part in these results.

TABLE XVII

WEALTH DEVELOPED IN 10 REPETITIONS OF THE SAME MARKET

Market	Total Wealth	Average Price	Total Quantity	pc000	Wealth ps3	ps7	pn2
r1	67.378	6.78	6,922	3.190	3.322	3.104	4.052
r2	63.615	18.51	5,036	5.869	4.981	4.257	3.210
r3	62.866	5.16	7,174	3.752	3.201	3.494	4.855
r4	66.410	7.67	6,748	2.197	4.263	2.538	2.354
r5	63.231	13.53	5,813	3.137	3.837	3.226	3.890
r6	63.857	14.79	5,640	3.752	2.813	3.539	5.145
r7	63.867	7.16	6,845	2.003	3.810	2.572	2.659
r8	63.676	7.15	6,845	2.793	3.521	2.843	3.372
r9	65.576	13.33	5,866	3.923	3.812	3.853	4.830
r10	64.446	13.35	5,832	2.988	4.805	3.345	3.303
Average	64.492	10.743	6,272	3.360	3.837	3.277	3.767
Stdev	1.477	4.472	716	1.088	0.687	0.545	0.955

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

Overall, agent ps3 was the strongest competitor in the repeated market. The satisficing agent ps3 accumulated the most wealth and also had a moderate variability in the 10 market repetitions. Agent ps3 tends to battle with the neural agent pn2 for market share. A weakness of agent ps3 is that it tends to overproduce, which in turn keeps prices down, so that it does not always profit from its high production. The tendency to overproduce by a single agent affects the production decisions and profits of all the producers in the market.

The neural agent pn2 also had a strong showing in the market repetitions, accumulating more wealth than any other producer in 5 of the 10 repetitions. However, the variability of results was high and the total wealth accumulated over the 10 markets was second behind ps3.

Agent pc000 accumulated a moderate amount of wealth but had the highest variability of any producer over the 10 market repetitions. I believe that knowledge of the demand curve increases the efficiency of the market, but does not give the agent any advantage over the other producers.

Although ps7 accumulates the least wealth in these repetitions, the accumulation is substantial, and the variability of returns is low. The consistency of returns is due to the "fair game" strategy that dampens large changes in production by this agent. The agent does not use an internally based microeconomic strategy, but rather, a macro-based strategy that is dependent on the market. While such a strategy is suboptimal in the sense that the agent produced less wealth than the other agents, it is a "safe" strategy because of the lower risk associated with its use.

Evidence of Learning by Neural Agents

The neural agents "learn" by the backpropagation of errors process. In this context, the neural agent makes an estimate of the quantity produced by others, the profit, and the price in the next period. When the actual values of these variables are realized, the error between the estimated and actual value is used to adjust the weights in the network to more closely approximate the actual value. As an example, consider Figure 24, which shows the actual and expected values of profits from the first instance of the repeated market r1, with agents pc000, ps3, ps7 and pn2. As shown in Figure 24, the network used by the neural agent pn2 has captured the general movements of the profit curve, but does not capture the specific details.

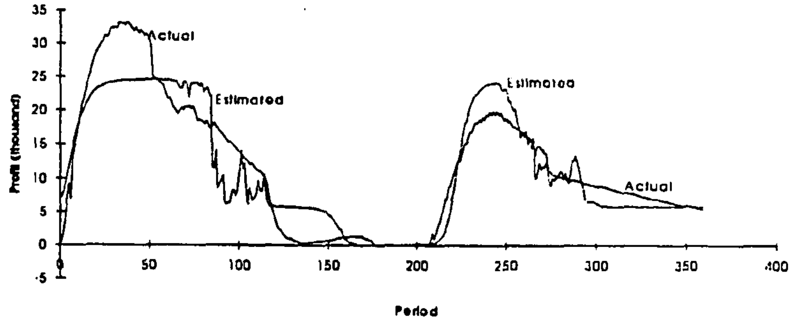

Figure 24. Actual and expected profits emerging in market r1.

The network used to compute the profits shown in Figure 24 was trained on the data that emerged from market r1. Figure 25 shows the mean squared error as a function of training presentations for this curve. Each cycle represents a complete pass through the training data, which in this case is 360 observations of profit from the repeated market. The error learning curve is typical of neural network learning by backpropagation. The mean squared error falls rapidly initially, and then stabilizes. Note that this network was trained on a wide variety of data from a range of markets before it was used to compete in market r1. The training shown in Figure 25 is conducted off line, using data that emerged from a specific market (r1).

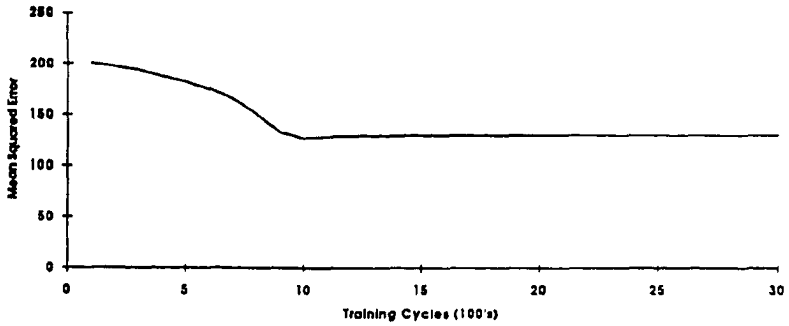

Figure 25. Mean squared error as a function of training cycles.

Figure 26 shows the actual profits in market r1 and the expected profits computed by the network after training for 1000 cycles. The network has apparently learned to more closely reproduce the profit curve, given the same input data. However, the details from one market may not apply to the next instance of the same market. It is the general principles that seem to transfer between markets.

To illustrate the idea that general principles transfer between markets, and to emphasize the importance of learning, a set of experiments was conducted in which the demand curve of the consumers was shifted. The average slope of the demand curve was set at 2.0 for all previous experiments, but the values assigned to the demand curve were 2.5, 2.3, 2.1, 2.0, 1,9, 1,7 and 1.5 in these experiments. The markets were interrupted after only 3 periods to provide a preliminary screening of the results. The wealth accumulated by each agent, and the total wealth produced by each market are given in Table XVIII.

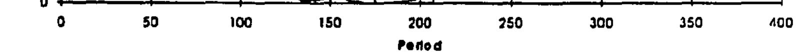

Figure 26. Actual and expected profits after training for 1000 cycles.

TABLE XVIII

ACCUMULATION OF WEALTH IN SHORT MARKETS WITH DIFFERENT CONSUMERS

Demand Slope	pc000	ps3	ps7	pn2	Total Wealth
2.5	-48	12,500	46,629	88,848	309,981
2.3	-48	14,401	51,997	101,325	322,747
2.1	-14	15,829	52,347	101,729	324,866
2	141,180	23,792	81,168	128,051	643,018
1.9	140,920	21,502	82,833	131,676	641,319
1.7	139,109	24,471	88,334	131,137	657,975
1.5	142,468	26,242	93,541	132,950	662,041

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

It is interesting to note that the Stackelberg agents were eliminated (wealth fell below zero) from all markets with slope greater than 2.0 in this short amount of time. The Stackelberg agents depend on knowledge of the demand curve, and if this knowledge is not correct it may have disastrous consequences for the agent in a very short time. In contrast, the neural and satisficing agents continue to prosper when the slope of demand curve shifts.

The preliminary results given in Table XVIII are extended by continuing two of the markets (slope 1.5 and 2.5), for a full 360 periods. Figures 27 and 28, show the individual quantities produced and wealth accumulated by each agent in the market when the demand slope was set at 1.5, and the market was continued for 360 periods. When the demand slope shifts to 1.5, the neural agent in particular continues to prosper and accumulates additional wealth. The satisficing agent ps7, also prospers because it follows the market, and in this case the market is prosperous. The Stackelberg agent accumulates approximately the same amount of wealth as it did in the repeated markets, but its share of wealth is declining as the market closes. The satisficing agent ps3 does not accumulate nearly as much wealth as it did in the repeated markets, where it was frequently the strongest performer. It appears that the success of this strategy is strongly associated with knowledge of the details of the demand curve, rather than to general principles.

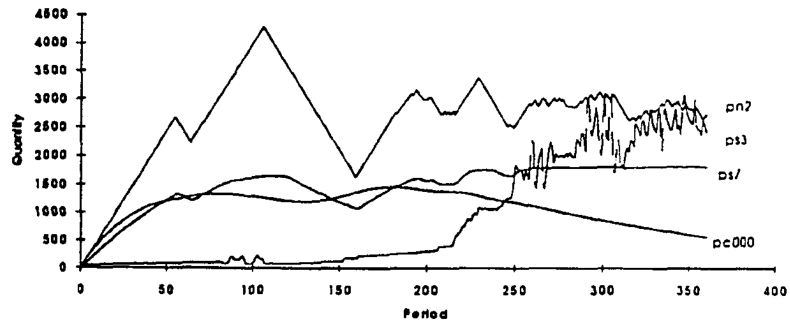

Figure 27. Quantities produced when demand slope is 1.5.

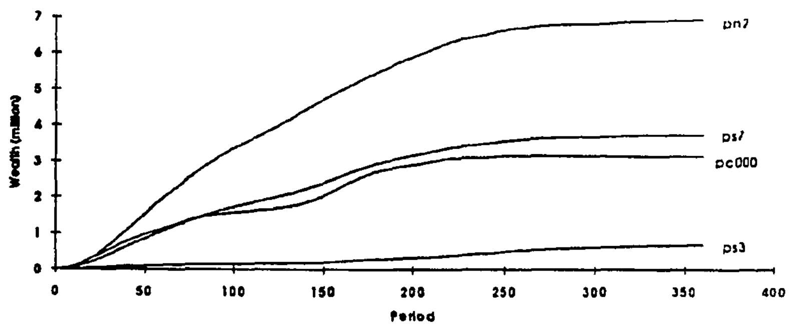

Figure 28. Wealth accumulated when demand slope is 1.5.

To further illustrate the importance of learning general principles, the consumers demand slope was set at 2.5, and the market was continued for 360 periods. The individual quantities and wealth produced are shown in Figures 29 and 30. As expected, the Stackelberg agent was eliminated early in the trading, leaving the neural and satisficing agents to satisfy the consumers demand. The neural agent appears to have taken advantage of the situation by keeping prices high, but producing more after the competition falls. The shift in the demand curve appears to create an opportunity, rather than a problem for the neural agent.

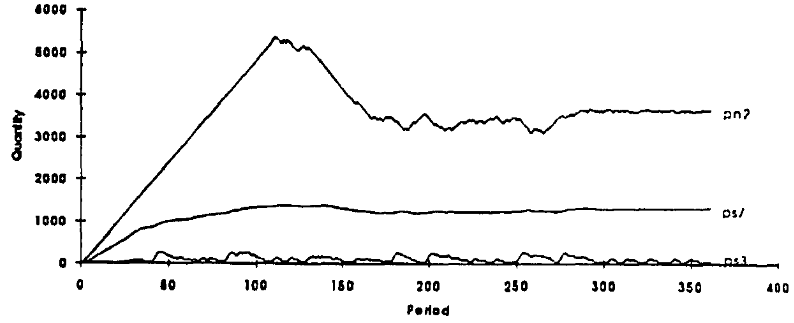

Figure 29. Quantities produced when demand slope is 2.5.

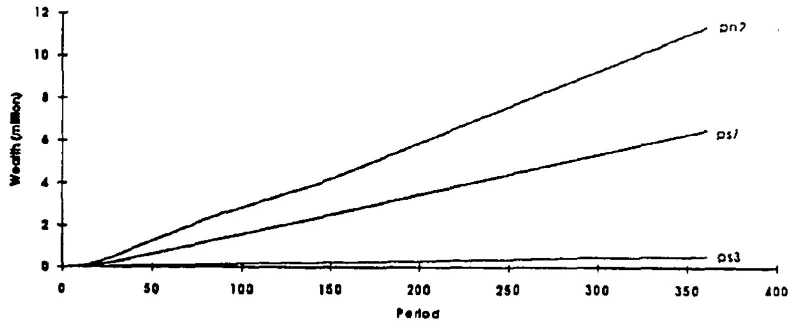

Figure 30. Wealth accumulated when demand slope is 2.5.

Again, the satisficing agent ps7 prospers by following the market, as shown in Figures 29 and 30. This fair game strategy is the most consistent strategy for producing a good, but not outstanding return across a variety of markets. In contrast, agent ps3 does very well in some situations but does not adapt to changes in the consumers easily. Table XIX shows the total and individual wealth produced in the average of the repeated markets, and in the markets with the shifted demand curves.

TABLE XIX

ACCUMULATION OF WEALTH IN EXTENDED MARKETS
WITH DIFFERENT CONSUMERS

Slope	2.5	2	1.5
pc000	-46	3,360,000	3,102,000
ps3	539,000	3,837,000	1,319,000
ps7	6,589,000	3,277,000	4,147,000
pn2	11,545,000	3,767,000	7,313,000
Total Wealth	49,764,000	64,492,000	69,066,000

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

Repeated Market with Learning

When agents are exposed to the same market repeatedly, the agents may learn both within the market, and between the markets. In this research the neural agents pn1 and pn2 are the only agents that have this capability. The weight files that represent the agent's knowledge are updated at the end of each period as the agent participates in the market. Additionally, the neural agents can learn off-line from the data that was developed and stored in other markets. The neural agents do not have to participate in a market to learn from the data emerging from the market.

The neural agents used in the research up to this point have always started each new market with identical weight files. The agents modify the weight files during the market according their experience, but always start with the same weights. In this set of experiments, the weight files for agent pn2 were carried over to the next market. The weight files represent the accumulation of knowledge from the on-line market interactions.

The repeated market with learning included the following agents: pc000, pn2, ps3, ps7. These are the same agents that were used in the repeated market without learning. The market structure was also the same: 4 producers, 16 consumers, 5 times per period, 4 periods per round and 90 rounds per market.

The wealth accumulated by each agent in the 10 repetitions of this market are shown in Table XX. These results do not indicate an improvement in performance by the neural agent as a result of learning from the market simulations. The neural agent consistently develops a substantial quantity of wealth, but its performance does not appear to have improved from learning. Since the market appears to have an underlying white noise process, it is not surprising that the network is unable to improve performance. This is typical of real world economic markets, where forecasting is not possible because the market exhibits white noise characteristics (i.e., unpredictable).

TABLE XX

WEALTH DEVELOPED IN 10 REPETITIONS OF THE SAME MARKET WITH LEARNING

Market	Total Wealth	Average Price	Total Quantity	pc000	ps3	ps7	pn2
rt1	61.461	5.46	7,125	2.670	3.397	2.617	3.316
rt2	64.459	5.11	7,189	2.682	2.844	2.814	3.425
rt3	67.036	6.43	6,977	3.245	3.843	3.476	4.183
rt4	61.362	16.74	5,295	4.456	5.308	3.349	1.928
rt5	66.201	21.52	4,539	4.758	3.763	3.821	4.165
rt6	63.541	5.51	7,122	2.156	2.203	2.269	2.966
rt7	66.807	7.84	6,736	3.941	3.399	3.266	3.751
rt8	63.141	7.86	6,748	2.936	3.069	3.023	4.361
rt9	60.821	5.21	7,159	3.392	4.072	2.725	2.276
rt10	68.572	7.81	6,747	3.332	4.436	3.270	3.443
Average	64.340	8.949	6,564	3.357	3.633	3.063	3.381
Stdev	2.715	5.593	904	0.822	0.869	0.462	0.808

Agent key: pc= Stackelberg, ps= satisficing, pn= neural.

Indication of Chaos

A suggestion of chaos in the distributed computer markets can be found by examining the quantity traded trace of the repeated markets. (See Figure 31). The figure shows that markets that are identical in nearly every respect produce traces that diverge after a short time.

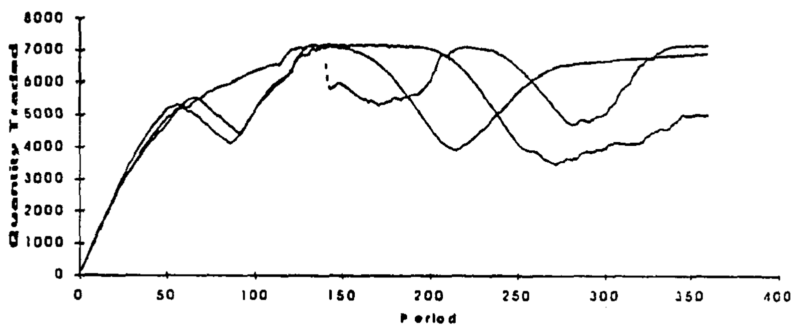

Figure 31. Three instances of the repeated market.

The difference between these markets is in the coefficients that are assigned to the agents at the markets inception, the ordering of agents as they approach the blackboard, and the initial values of price and quantity. After numerous trials, it was determined that the traces from repeated markets including these agents diverge, even if all sources of stochasticity are removed, but the agents are given minutely different initial values of price and quantity. Strong dependence on small changes in initial conditions is a classical indication of a chaotic attractor. The implication is that long range forecasting of quantity traded (or price) at a particular time is not possible with any degree of accuracy, even if precise details about the economic agents are known.

The market shown in Figure 31 is a mixed market consisting of Stackelberg, neural and satisficing agents. This market shows a variety of interesting behaviors. For instance, one trace shows a rapid build up in quantity traded followed by an abrupt decline. All of the markets shown here exhibit protracted declines in trading, followed by growth at different rates. One thing this figure does not suggest is an equilibrium that can be predicted by knowing about the agents in advance. The markets are continually changing because of the strategies and motivations of the agents. Figure 32 shows the result when the market consists only of Stackelberg agents. In this case the market approaches an equilibrium and remains there.

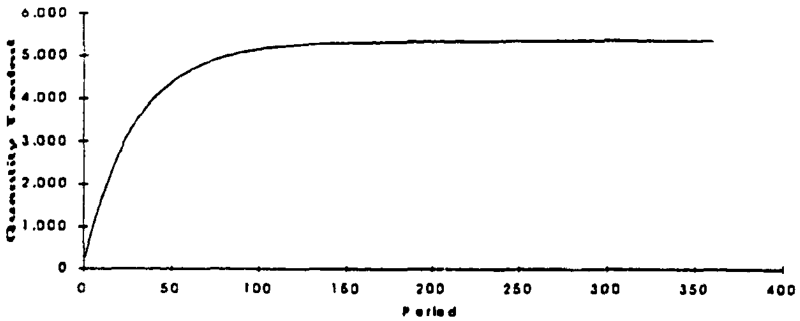

Figure 32. A stationary market of Stackelberg agents.

Many of the markets exhibit unexpected changes in the quantity traded over time. For instance, Figures 33 and 34 show abrupt changes in the traded quantity that are completely determined by the characteristics of the agents. The last figure in this series, Figure 35, shows the quantity traded in a market composed entirely of satisficing agents. This figure is of interest because it comes to apparent equilibrium (as a constant rate-of-change) far from the much higher equilibrium markets.

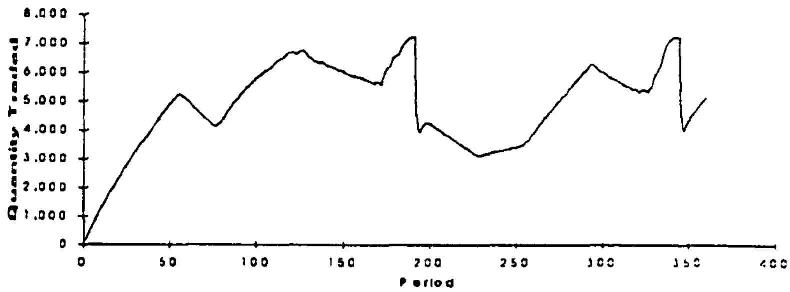

Figure 33. Irregular changes in the quantity traded over time.

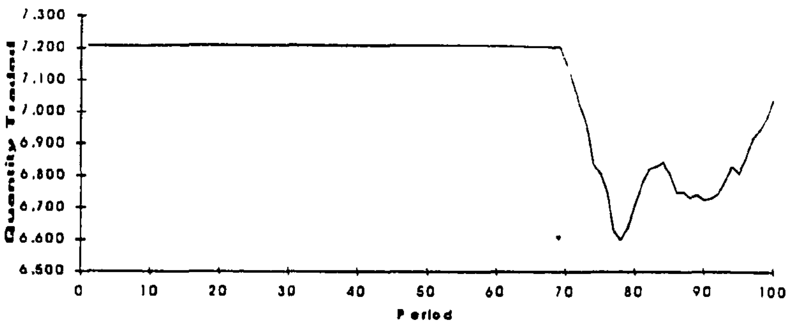

Figure 34. Abrupt change in the quantity traded.

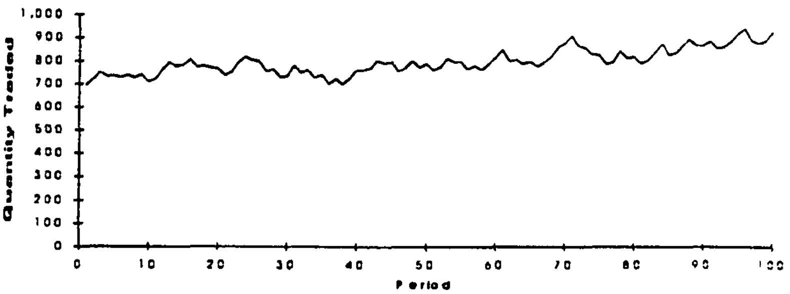

Figure 35. Gradual shift in the quantity traded in the last 100 periods.

From the viewpoint of dynamic systems, these markets appear to have different types of attractors. For instance, the market of Stackelberg agents appears to be attracted to a fixed point with small random variations. However, markets composed of a variety of strategies appear to be quasi-periodic, or even chaotic.

Figure 34 gives the impression of a punctuated equilibrium, where the market is stable for a time, and then suddenly shifts.

Interaction of Agents in the Market

The agents that produce the smoothest and most regular traces are the Stackelberg agents. These agents make small and predictable shifts in the production quantity in each period. An example is shown in Figure 36, where four different Stackelberg agents compete in the same market. The production quantity by each agent is determined by the agent's conjecture about the quantity the other agents will produce in the next period.

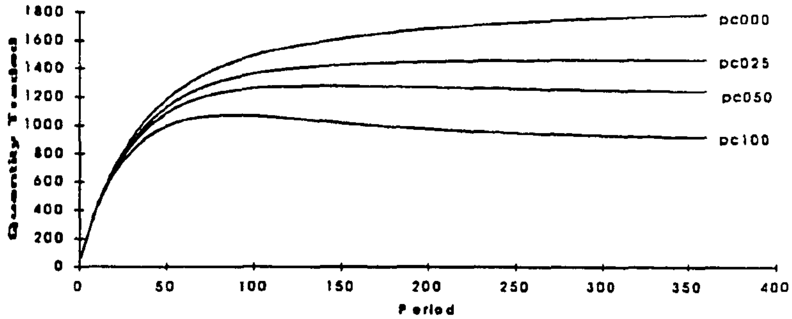

Figure 36. Quantities produced by Stackelberg agents in market m1.

In contrast, Figure 37 shows the production quantity of a mixed market containing neural and satisficing agents. The neural agent is recognizable by the large triangular path that it produces as it makes adjustments to the production quantity. The satisficing agents, on the other hand, tend to make sharp adjustments in production. Agents ps1 and ps2 clearly follow a much lower production path than the neural agent pn2 or the other satisficing agent ps8, but this is not always the case. Note that these charts show the actual production by each agent, which may be more erratic than the total quantity series.

Figure 38 shows the variability introduced into the market by the strategy followed by agent ps2. This market includes a neural agent, pn2, a Stackelberg agent, pc000 and two satisficing agents, ps2 and ps7. The satisficing agent ps2 follows a low production path in the first part of the market, but suddenly raises production rapidly, followed by a precipitous collapse in production quantity. At times, the other satisficing agents also produce sharp changes in production quantity, but agent ps2 generally introduces more variability into the markets than the other satisficing agents.

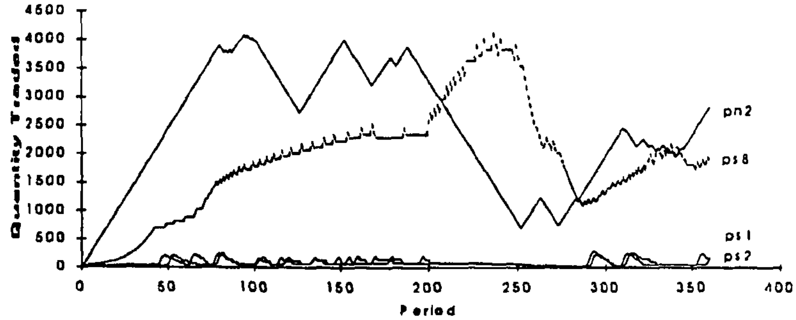

Figure 37. Quantities produced by each agent in a mixed market.

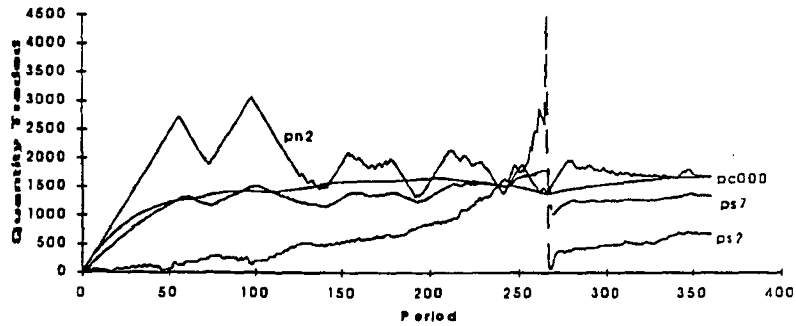

Figure 38. Abrupt change in quantity produced by agent ps2.

Accumulation of Wealth

The agents in these markets all begin with 1000 units of wealth, and accumulate more wealth by trading in the market. The total production by the each agent is limited by the wealth the agent has accumulated. A portion of the agent's wealth is risked in each production period, and it is possible for the agent to produce goods that are never sold at a profit, thus losing wealth. The charts in this section give some idea of the variability and interaction among the agents as they compete for wealth in specific markets.

Some agents seem to produce wealth in a very steady and reliable fashion (see Figure 39), while others produce wealth in a more erratic manner. Figure 39 shows the wealth accumulated in a market of Stackelberg agents, and Figure 40 shows the accumulation of wealth over time in a mixed market containing agents ps1, ps2, ps8 and pn2. Agent ps1 and ps2 accumulate wealth that is consistent with their low production quantity. Agents pn2 and ps8 produce far more wealth over time and also accumulate wealth at a much higher rate. In the first half of this market, the neural agent pn2 accumulates wealth at a higher rate than the satisficing agent ps8. The tables are turned temporarily, but by the end of the market, the neural agent is producing wealth at a higher rate that any of the other agents. This type of trade off in leadership is frequently observed in the competition among the agents in these distributed markets. Figure 41 shows a mixed market with a very successful agent ps3. However, this success is likely to be of limited duration because the other agents in the market will make changes over time that will probably reduce the success of ps3.

Figure 41 clearly shows how some of the satisficing strategies can at least temporarily best the neural and Stackelberg agents. But when the market is repeated, different results emerge each time. Figures 42 and 43 for example, show the wealth developed by each agent in another instance of the same market.

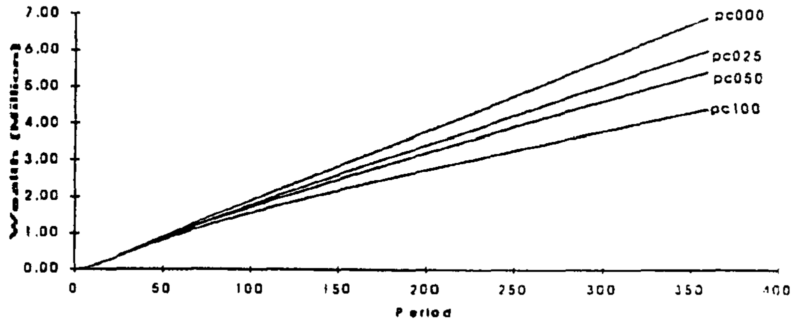

Figure 39. Steady accumulation of wealth by Stackelberg agents.

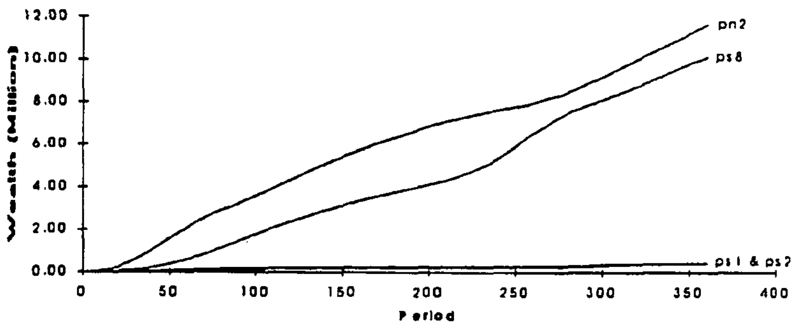

Figure 40. Irregular division of wealth in a mixed market.

In the case shown by Figure 42, the neural agent pn2 accumulates wealth far more rapidly than the other agents, and retains a healthy lead as the market closes. Figure 43 gives a sense of the interaction that sometimes emerges among the agents. In this case, pn2 produces a large amount of wealth early in the market, while ps3 lags far behind. A flat portion on the curves indicates that none of the agents produce much wealth for a time, but this changes as ps3 makes rapid gains, and appears to break the deadlock.

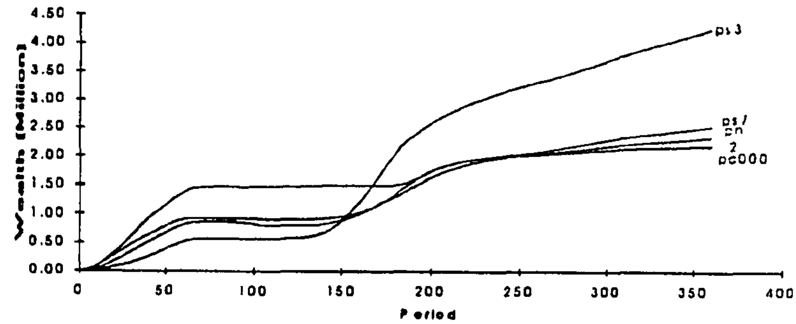

Figure 41. Satisficing agents sometimes accumulate considerable wealth.

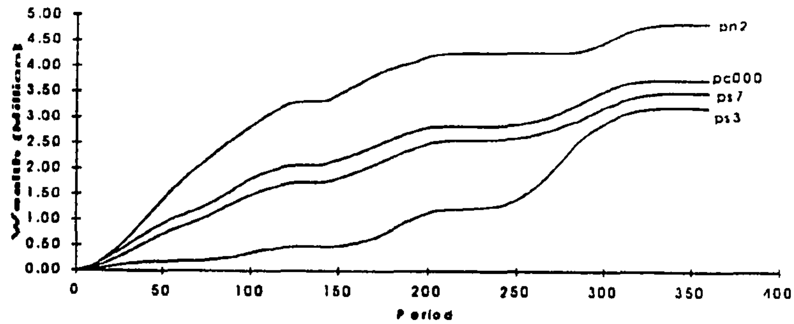

Figure 42. Neural agents frequently dominate in mixed markets.

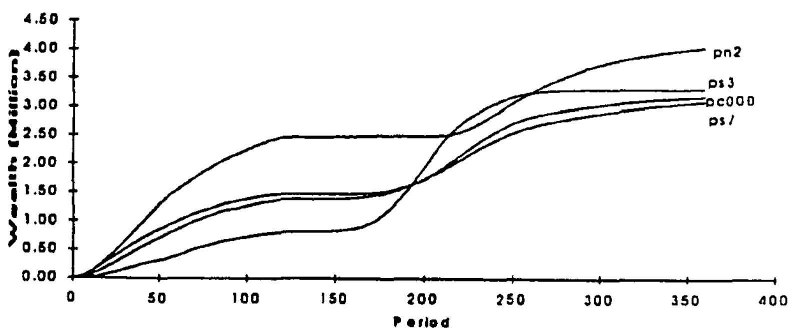

Figure 43. Interaction in the accumulation of wealth in a mixed market.

CHAPTER VIII

ANALYSIS OF ECONOMIC DATA WITH NEURAL NETWORKS

INTRODUCTION

A common statement in the neural nets literature is that neural nets have a natural advantage over traditional models such as ordinary least squares (OLS). One thing that I will show here is the relationship between OLS and connectionist models. In addition, I will show that for many problems in applied economic analysis, network models are not superior to simple alternatives such as OLS or better known non-linear methods.

A simple linear model can be written as follows:

$$y=a_0 + a_1x_1 + a_2x_2 +...+ e$$

Where y is the dependent variable, a_n are coefficients, x_n are independent variables, and e is an error term. The coefficients are chosen to minimize the sum of the squared errors.

It is known that (in vector notation)

$$\hat{A}=(X'X)^{-1}(X'Y)$$

and that this computation minimizes the sum of the squared errors. Several summary statistics such as R-Square and the F statistic are usually reported with such a regression in order to indicate the goodness of fit between the observed and computed values. The statistical reliability of the coefficients $a_1, \ldots, a_n$ are indicated by the associated t-statistics.

Such models have been widely used in economics because of their simplicity as well as their easy and rich interpretation. Simple changes in representation, such as the logarithmic transformation, extend these models to a wider variety of functions, including the well known Cobb-Douglas function (See [Varian, 1984] for details).

Figure 44 shows a feedforward network with 2 layers and a linear transfer function. This network is equivalent to the linear model mentioned above. The term "feedforward" indicates that all computations follow a forward path through the net, and feedback is not permitted. There are n+1 inputs into the network, including the constant term x_0. In network terminology, the constant term is known as the "bias". Each connection between the input and output node is weighted with an adjustable coefficient. Inputs follow the paths forward through the net to the output node. In this model the transfer function is linear so that the output from the net is just the weighted sum of the inputs.

It is at this point that connectionist and ordinary least squares models begin to differ. The OLS model simply computes the coefficients by solving for the vector $\hat{A}$ in:

$\hat{A} = (X'X)^{-1} X'Y$. However, a feedforward network is trained by backpropagation of errors. The weights are initially assigned at random and then trained by making multiple presentations of data, comparing the computed output with the expected output, and making adjustments to the weights. The adjustment procedure is known as backpropagation [Rumelhart & McCelland, 1986] and is a form of hill climbing or gradient descent of the error surface. Like other gradient descent search procedures, the backpropagation algorithm is not guaranteed to find the coefficients that produce the global minimum error. Error adjustment continues until some preset criterion is satisfied.

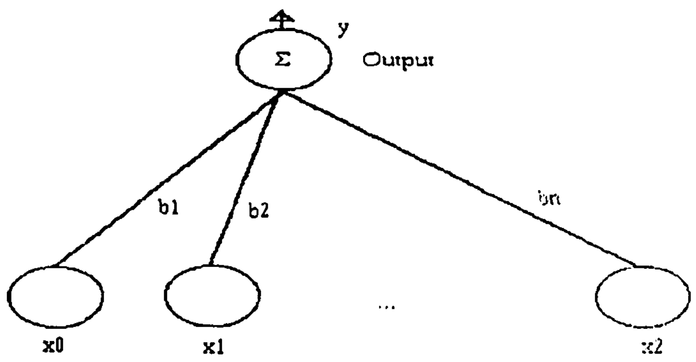

Figure 44. A linear network.

Backpropagation adjusts weights according to the gradient of the error surface, and minimizes the sum of squared errors, just as in OLS. However, the transfer function used in network models is usually a non-linear equation such as the logistic equation shown below.

$$o_j = \frac{1}{1+e^{-\Sigma a_i x_i}}$$

where o_j is the output from the j^{th} element, a_i are coefficients, and x_i are inputs into the j^{th} element. Backpropagation, like other non-linear methods, is much more computationally intensive than OLS, and is not guaranteed to find the global solution. By chance, the random starting values of the coefficients may be near the global solution, and in this instance the network method would find the solution very quickly, perhaps even faster than OLS, because OLS requires a matrix inversion. But in the general linear case there is no apparent advantage for using any non-linear technique because of the computational disadvantage. Theoretically, the network solution and the least squares solution are identical, but in practice, there may be good reasons for selecting between the methods. This becomes clear in the example applications that are presented below.

White has shown that feedforward networks with non-linear elements, and at least one hidden layer are equivalent to stochastic estimators [White, 1989, 1990], [Hornik et. al., 1990] . (Hidden layers are processing elements that are sandwiched between the input and output layers.) These networks may consist of multiple functions and are also closely related to flexible functional forms that are sometimes used in econometric estimation [Berndt and Khaled, 1979]. The equation summarizing a network with a single hidden layer, each with a different transfer function is shown below. In applied work, it is often useful to use a linear function for f_2 and a non-linear function for f_1.

$$y=f_2(\Sigma_i a_i(f_1(\Sigma j b_{ij} x_j)))$$

EXAMPLE APPLICATIONS

Demand Functions

Consider the demand for water in a local economy where the quantity of water purchased is a function of real price, population and the season of the year. Using logs of the continuous variables, the regression equation is specified as:

$$\ln(q)=a_0+a_1\ln(x_1)+a_2\ln(x_2)+a_3s+e$$

Where q represents the demand for water in hundreds of cubic feet, x_1 represents the real price of water in dollars per 100 cubic feet, and x_2 represents the population in the given month. S is an indicator variable that takes on the value 1 in the peak months of June, July, August, September and October, but is 0 in all other months. (The data for these tests were provided by Portland Water Bureau, Portland, Oregon). The same data set was modeled by least squares regression and by a linear neural network with the results shown in Table XXI. R-square is similar for the two models, and the price and population elasticities are of similar magnitude and sign. In addition, the seasonal peaking factors computed by the alternative models are nearly identical. The models differ substantially in the weight and significance assigned to the constant term, but for this discrepancy the network model does not pay a substantial price in terms of R-square. Overall, these models appear to be comparable in the results produced, but the network model pays a substantial penalty in terms of computational efficiency. In the case reported here, the OLS solution was obtained in milliseconds, while the network solution typically took from 5 to 15 minutes to emerge when using a 33 MHZ 386 PC with a 387 math coprocessor. Additionally, the network computations were repeated several times with different starting values to verify the results.

A grid search of the error space of these variables showed a long ridge. As a result of moving along this ridge, different coefficients produce similar goodness of fit statistics. However, there is a global maximum R-square for this system of coefficients, and this is very close to the set of coefficients produced by OLS regression. But many sets of coefficients produce comparable R-square results. The network model of this linear equation finds some coefficients that produce a reasonable fit, but are not optimal. In this case, choosing between the optimal and non-optimal coefficients is not easy.

Network models become increasingly attractive as the model becomes more complex, and where data are messy. To demonstrate some of the advantages of connectionist models, the simple water demand model mentioned above was re-estimated using additional data indicating the month of the year, rather than just the season. In addition, data summarizing weather conditions were used in the estimation where they were available. In total, 26 variables were used to estimate the water demand relationship. No specific functional form was specified, but the data was scaled into the 0 <= x <= 1 range.

TABLE XXI

WATER DEMAND IN THE PORTLAND AREA 1960 - 1988
NETWORK AND LEAST SQUARES MODELS COMPARED

	NETWORK		LEAST SQUARES	
	Coefficient	t-stat.	Coefficient	t-stat
Constant	-0.04	-0.03	-3.47	-2.5
Population	1.1	10.5	1.35	13.1
Price	-0.36	-16.9	-0.36	-17.1
Season	0.34	3.9	0.38	4.5
R-square	0.64		0.65	

The data consisted of monthly observations of the real price of water, population, real per capita income, and 12 indicator variables (1 or 0) that correspond to the month that the data were observed in. The weather variables included the monthly rainfall (inches), number of rainy days, number of days over 75, 80, 85 and 90 degrees, cooling degree days, and heating degree days. However, the weather data was only available for 6 months of each year. No weather data were available in December through May, and the missing values were replaced with zeros (or alternatively, 9999).

This model would present a serious problem for OLS estimation on several counts. The model is overidentified because all the months are included explicitly in the data. Multicolinearlty is a problem because several of the weather variables are very highly correlated. The weather data is only available for six months of the year. No functional form is known or suggested from the data, especially with regard to interactions among the variables.

This model was successfully fitted with a variety of connectionist networks containing from a single hidden element, to as many as 100 hidden elements. R-square ranged from 0.75 to 0.8 for different architectures, indicating that adding large numbers of hidden elements was not especially helpful. The network models quickly learned to ignore missing data, while using all data that was present. Network models thrive on redundancy, using all data that is available, and filtering out the irrelevant components. The network models are not limited by the degrees-of-freedom problems that occur with traditional models. The model containing 100 hidden elements produced over 2700 coefficients with only 282 data observations. Furthermore, no ad-hoc functional form was specified a-priori. Instead, the network found a functional form that fit the data.

This simple application of the connectionist approach to solving economic problems shows some of the strengths as well as the weaknesses of connectionist models. The connectionist approach works well with messy data, and where no functional form is known to fit the data prior to modeling. On the negative side, connectionist models are computationally intensive, and the coefficients emerging from connectionist models may not have a satisfying theoretical interpretation. Additionally, the neural models may be no more accurate (or even less accurate), than simple alternatives such as OLS, assuming that the simple models are applicable to the data at hand.

When compared with other non-linear estimation methods, network models are found to suffer from all the problems associated with gradient descent optimization procedures. Network models are not guaranteed to reach a global optimum. Halting conditions are arbitrary, and results may depend on the starting values of the coefficients. The coefficients of network models may be difficult to interpret, and it is not clear how to perform statistical inference on the hidden layer coefficients. These tests also show that contrary to the theoretical equivalence between the models, a network model may not be as accurate as a simple linear model if the relationship is indeed linear. Many of these criticisms also apply to other non-linear models.

Cost Functions

In general neural networks can be used to construct a model of any relationship without specifying a functional form. White has shown that feedforward networks are equivalent to stochastic estimators of the relationship between inputs and outputs [White, 1989a]. These functions are also closely related to flexible functional forms that are sometimes used in econometric estimation.

For instance, production and cost functions are common in economics:

$$y = F(X)$$

$$c = G(W,y)$$

where y is the output, W is a vector of input prices, and c is the cost of production. The usual approach is to minimize cost given the constraint of the production function. The neural network provides a simple way to construct the model c=G(W,X) given data on costs, inputs quantities and input prices, without specifying the functional form. However, the assumption that the data results from a cost minimizing process remains.

Table XXII shows the error analysis for network cost functions that were trained for 100,000 iterations through the data published by Berndt and Khaled for the U.S. economy between 1947 and 1971 [Berndt and Khaled, 1979]. The network has a total of 9 inputs, including the bias, and one output. The inputs into the network are the prices and quantities of inputs into the U.S. economy: capital, labor, energy and materials. The sole output from the network is total cost.

TABLE XXII

ERROR ANALYSIS FOR NETWORK COST FUNCTIONS

Trial	Sum of Squared Errors	Direct Linear Connection	Nodes in Hidden Layer	Thousand Iterations
1	0.0085	yes	8	100
2	0.0777	no	8	100
3	0.0068	yes	0	100

Table XXII gives the sum of squared error results for three trials that used the same data and training algorithm, but with different architectures corresponding to different types of models. Trial 1 included 8 elements in the non-linear hidden layer, as well as direct linear connections between input and output layers. This is the most complex model presented here, the other two models are simplifications of it. Trial 2 included only the 8 non-linear nodes in the hidden layer, without any direct connections between the input and output layer. Trial 3 included only the direct linear connection between the input and output layer.

Using the sum of squared error results as the criterion to judge these models indicates that trial 3, with only linear connections between inputs and outputs, is superior to the networks with non-linear connections. This is counter to the results of Berndt and Khaled, which indicate that a non-linear relationship exists, thus the neural net was not able to find the non-linear relationship in this case.

Forecasting Economic Conditions

Another approach to estimation of economic systems is not based on economic theory, but on the minimization of estimation errors over time. For instance no theoretical explanation is found in the economic literature for the relationship between the values of leading economic indicators and the future state of the economy [Auerbach, 1982]. However, such a relationship is widely perceived to exist. These values are typically estimated with a multiple ARIMA process:

$$y_t = a_0 + a_j y_{t-j} + \dots + a_k y_{t-k} + b_j x_{t-j} + \dots + b_k x_{t-k} + e_t$$

where y is a dependent variable, X is a vector of independent variables, and e_t is the error at time t.

The Bureau of Economic Analysis has published a series of leading, current and lagging economic indicators since the 1930's. These indicators have been revised several times over this period, most recently in January, 1989. (These indicators have been published in the Business Conditions Digest until May, 1990 when they were consolidated into the Survey of Current Business.) The 11 leading indicators are used to form a composite leading index while 4 indicators are used to develop an index of current economic conditions. This discussion will focus on the leading and current indicators, which are listed below in Table XXIII.

TABLE XXIII

ECONOMIC INDICATORS

Coincident Indicators

Number	Title
41	Employees on nonagricultural payrolls.
47	Index of industrial production.
51	Personal income less transfer payments in 1982 dollars.
57	Manufacturing and trade sales in 1982 dollars.

Leading Economic Indicators

Number	Title
1	Average weekly hours of production or non-supervisory workers in manufacturing.
5	Average weekly initial claims for unemployment insurance, state programs.
8	Manufacturers' new orders in 1982 dollars, consumer goods and material industries.
19	Index of stock prices, 500 common stocks.
20	Contracts and orders for plant and equipment in 1982 dollars.
29	Index of new private housing units authorized by local building permits.
32	Vendor performance, percent of companies receiving slower deliveries.
83	Index of consumer expectations.
92	Change in manufacturers' unfilled orders in 1982 dollars, smoothed.
99	Change in sensitive materials prices, smoothed data.
106	Money supply, M2.

Most of the indicator series date back to 1947, and were collected monthly, however, series 83, the index of consumer expectations was not collected at all before 1952, and only quarterly until 1978. For the purpose of this study, missing values for this series were assumed to equal the last observed value. In other words the value is constant for each quarter between 1952 and 1972. This results in a monthly data set spanning the 1952-1990 period and containing 452 observations.

The leading indicators are assumed to precede the current indicators of the economy but it is unclear by how much. The time lag between a change in the leading indicators and a corresponding change in the current indicators varies. In addition, the relationship between the leading and current series is not precisely established, but is usually estimated to be 6-9 months. For this study the time lag between leading and current indicators was arbitrarily taken as 6 months. For instance, the leading indicators for January were used to predict the current economic activity in July of the same year. An interesting unanswered question is how to train networks to learn the appropriate lag for a data set.

Leading indicators can be used to forecast impending changes to the national economy, such as a recession or a recovery from recession. Such turning points are comparatively rare in recent economic history. It is my intent in this example to examine changes in the trend of the indicators and measure agreement between forecast and actual values of the indicator series with neural network methods. Since network methods do not posit a functional form they are free of the restrictions implied by such assumptions, such as constant returns to scale and homotheticity. Several different network designs were tested with variable success. The users of neural nets are burdened by large numbers of ad-hoc decisions regarding methodology. It is difficult to find a good network solution without a certain amount of groping.

The inputs into the initial network were made up of the 11 leading indicators mentioned above. In addition, all experiments use a constant input called the bias term. The selected network includes a single hidden layer, also consisting of 11 units. The number of nodes in the hidden layer and the number of hidden layers are both variables that were considered in modeling with neural nets. Preliminary results indicated that no apparent gain in sensitivity was observed by adding additional layers or nodes to this network.

The hidden layer was fully connected to all 11 inputs plus the constant term. The processing elements in the hidden layer operate in two steps. First, the inputs into the element are totaled and second, the resulting total is operated on by a transfer function, which in this case was a logistic function. Both operations can be expressed in a single equation: $o_j = \frac{1}{1+e^{-\Sigma w_i x_i}}$

Where x_i represents inputs into the processing element, and w_i represents the weights that are applied to the inputs. Many alternative functions are available for the transfer function, the sine function and hyperbolic tangent functions were also used in some of the preliminary multiple layer experiments but with poor results.

A signal is sent to the 4 elements in the output layer from each of the 11 processing elements in the hidden layer, as well as from the constant term. The nodes in the output layer are similar to the hidden nodes, but the transfer function is not necessarily the same. The choice of transfer functions is made by the analyst.

The noise of the forecast was compensated for by running 10 repetitions of each test with different random starting values of the coefficients. Random values of the coefficients were generated between +-0.1. Several tests were also done with starting values at +-0.5 and +-1.0 but with very poor results and a high tendency for the nets to diverge. Rather than running the net for a fixed number of presentations, a convergence criteria was set at RMS error of 0.017 over 100 presentations. This figure was chosen because it represented the lower end of the observed RMS error curve. RMS error rarely fell to 0.015 but reached the

0.017 level on 10 of 11 trials. The most successful architecture and network parameters are listed below:

Inputs: Constant term, 11 leading indicators, 4 current indicators.

Outputs: 4 Current indicators that are 6 months in the future.

Hidden Layer: 11 nodes that are fully connected to each element in the input and output layer.

Convergence Criteria: RMS error <= 0.017 over 100 presentations.

Learning rate: 0.05

Table XXIV shows the statistics from the network that was trained on 434 observations and tested on 440 observations. The RMS error between observed and fitted values are listed for each of the 4 series measured over the 440 observations. In both absolute and percentage terms the value of series 47, Industrial Production, is the poorest fit at 0.030 percent RMS error, while series 41, Manufacturer's Sales was the best fit at 0.013 % RMS error. This general pattern was true thorough all of the tests reported.

The value of U is known as Theil's inequality coefficient in the simulation and modeling literature. Theoretically, the value of U can range between 0 and 1.0, with 0 indicating a perfect fit and 1.0 indicating the worst possible fit between observed and computed values. The values of U shown in Table XXII indicate a good correlation between observed values and the figures estimated by the net.

TABLE XXIV

ERROR ANALYSIS OF ECONOMIC INDICATOR DATA

Series	41	47	51	57
RMS Error	0.856	2.212	0.299	0.560
RMS Error %	0.013	0.030	0.0185	0.0214
U	0.006	0.012	0.008	0.009
Um	0.003	0.002	0.005	0.001
Us	0.008	0.004	0.009	0.002
Uc	0.989	0.995	0.987	0.996

The value of Theil's U is computed as follows:

$$U = \frac{\frac{1}{n}\Sigma_i(y_i-x_i)^{1/2}}{(\frac{1}{n}\Sigma_i y_i^2)^{1/2}+(\frac{1}{n}\Sigma_i x_i^2)^{1/2}}$$

where y_i is the estimated value of the series, and x_i is the actual value for each on the n observations. The numerator of U is the rms error, while the denominator provides the scaling that insures that U remains in the 0 - 1 interval. Pindyck and Rubinfeld, [1981], show that Theil's U coefficient can be decomposed into 3 inequality proportions: Namely, Um is the bias proportion, Us is the variance proportion, and Uc is the covariance proportion. The sum of these proportions is Um + Us + Uc = 1.0

The proportions of the inequality coefficients are defined as: The bias proportion U_m is an indication of systematic error and measures how the average values of the simulated and actual series differ. The value of U_m should be near 0, and a value of U_m larger than about 0.1 is an indication of a serious systematic error or bias in the estimation. The values of U_m shown in Table XXIV do not indicate a problem with systematic bias.

The variance proportion U_S is an indication of the ability of the network to replicate the variability in the observed data. If U_S is large it indicates that one series varies considerably while the other series is relatively constant. The largest value of U_S listed in Table XXIV is 0.009 for Personal Income, series 51. This value does not indicate a problem with the variance proportion.

The covariance variation U_C, indicates the residual error after the other variabilities have been accounted for and should be close to 1.0. This is the case for the estimation reported in Table XXIV. The smallest covariance proportion shown is 0.9868 for the Personal Income series 51, which is not a cause for concern.

Although the network is able to compute reasonable outputs for each of the four coincident indicators given the historical inputs, the network is not able to accurately forecast the coincident indicators. The forecasts were biased and missed turning points in economic activity. Figure 45 shows the actual and estimated values of Industrial Production that resulted from using the network described above for forecasting the coincident indicators. The forecast is clearly biased, but captures the general trend of the data. However, this level of accuracy is not acceptable for forecasting economic activity.
Many other network architectures, data sets, and training methods were used in an attempt to produce more acceptable forecasts of coincident economic indicators, but real improvement in results was not forthcoming.

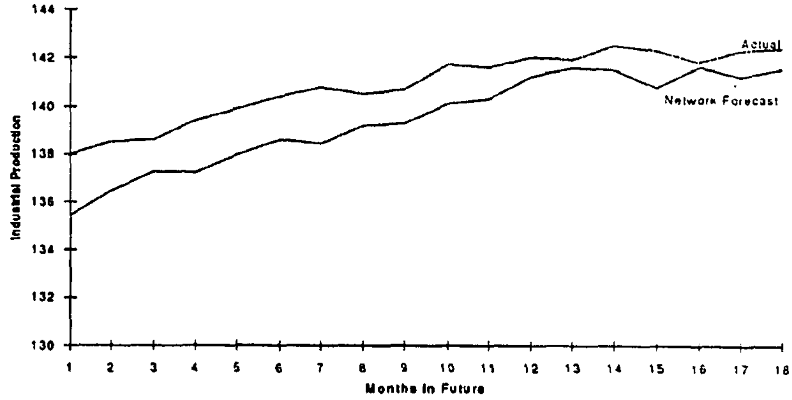

Figure 45. Actual and fitted values of Industrial Productions over 18 months.

DISCUSSION

Economic applications of neural networks are more appealing in theory than in practice. When applied to typical problems in economic analysis, neural nets trained with backpropagation are often less accurate and more complex than commonly used alternative methods. Neural networks are orders of magnitude slower that other methods when using technology that is commonly available (e.g., serial computers). However, massively parallel computer technology is evolving rapidly, and neural type computers are starting to appear in the market. Such computers should be much faster than serial computers.

Neural networks are conceptually appealing in economics because they do not posit a particular functional form. Unfortunately, the results emerging from the network are dependent on numerous ad-hoc assumptions about network architecture, learning rates, transfer functions, training methods, and so on.

Training algorithms such as backpropagation are based on gradient descent of an error function. One of the hazards of such algorithms is that they may find local optima, rather than the globally optimal solution. In practice, neural nets may not even find local optima.

Neural networks are conceptually based on the neural system that makes up the human brain. The brain is a massively parallel distributed computer that naturally solves various types of pattern recognition problems, imputes meaning to patterns, and is generally not proficient at explicit numeral computations. The brain has attributes such as the ability to learn from experience, organize information, and fault tolerance. Neurons are modeled as simple computational devices that support learning from experience.

Extending the neural model to economic systems appears to be misguided. It is true that economics systems are distributed computers, but neural networks are based on a computational model that is distinctly different from the organizing principles of an economic system. The intent in this chapter was to use neural networks as computational models of economic systems without interpreting the models as structural models of the distributed computation that exists in economic systems. Thus no claim was made that economic systems learn in the sense that the brain learns (i.e. an economic system is not a big brain, with simple processing units, but has other organizing principles).

The neural models used in the examples presented here were moderately successful as computational models of economic systems. Particularly strong features are the ability to learn from experience without choosing a functional form a-priori, and analysis of messy data, including data with missing or redundant observations. However, drawing deep theoretical conclusions from the internal structure that emerges in the network models, or using network models without reference to alternative computational models when they are available is misguided.

There is no reason to discard the simple and well known tools of econometric analysis for neural methods when the problems is an easy one. But economists may want to add neural methods to their tool kits for solving more difficult problems. Neural methods are recommended where simple methods fail to produce sound results.

CHAPTER IX

CONCLUSIONS

The goal of this research was to model economic markets as distributed computational systems. As such, the model includes agents based on different strategies that were independent of any central control. The continued success of the individual agents was dependent on their own strategies, as well as the strategies of others in the distributed computer market. Communication in the market was facilitated by a mutually accessible area of computer memory called a blackboard. The blackboard structure is very useful for sharing information among the agents in distributed systems. This structure imposes little constraint, while allowing immediate access to information by all agents. Clearly, the blackboard is only one of many conceivable structures for transferring information among agents.

It is my claim the blackboard market is a simulation that has the principal qualities of an economic system. The major difference is in the environment; real world economic systems exist in the sociosphere, but computer markets are isolated within the computer environment. Computer trading is commonplace in real world markets where some or all of the economic agents are computer programs.

The distributed computer market satisfies all the criteria for an economic system based on the definition of economic systems provided in Chapter II. An economy is a system that deals with scarcity through trade. The market is a recognizable unit existing within the computer environment and has the attribute that it has inputs and outputs. The market has subunits comprising producers and consumers that deal with scarcity by trading among themselves. Prices, quantities and wealth emerge naturally from the interaction of these agents in the market. The agents have meta-observer status in that they can observe the emergent properties of the system and change their behavior accordingly. The fact that there are no human beings participating in this market is immaterial to its definition as an economic system. The value of producing an economic system that can be studied in detail is large because the researcher has knowledge of the detailed working of the market. The market also becomes a repeatable event.

In a system including scarcity and trade, the agents prosper according to the success of their trading strategies, and economic agents compete with each other for the scarce resources. There is competition among the agents on three levels, among producers for market share and profits, among consumers for consumption rights for scarce resources, and among producers and consumers for the terms of trade. There are no restrictions on the strategies that are available to the agents, and the same public information is available to all.

The complexity of decision making rises as the number of agents increases because each agent must take into consideration the possible behaviors of its competitors. Perfect competition emerges when large numbers of agents interact in the market, and conversely, monopoly results when only a single producer dominates the market. The most interesting results were observed in the mid-range of complexity, where agents must consider their competitors' actions in the market. The number of agents participating in the research markets were held at a moderate level for this reason.

An alternative system structure based on neural networks was considered and rejected by following the systems approach outlined in Chapter II. A neural network may perform successfully as a computational model of an economic system as shown in Chapter VIII, but does not have an inner structure that is consistent with an economic system. An economic system comprises independent agents organized by trading for scarce resources. Scarcity and trade are attributes of economic systems that are not present in the neural network models considered here. A neural network may take all the inputs from a market and map them to the appropriate outputs after sufficient training. But this does not indicate that the internal elements represent buyers and sellers that trade for scarce resources in a market.

As mentioned above, neural networks were found to perform satisfactorily as computational models of economic systems in several applications, especially when assumptions about the functional form were not made. However, in the event that functional form could be postulated, alternative methods appeared to provide both more accuracy and more information about the underlying relationships between the variables. The ability of the neural networks to adapt to a changing environment, and freedom from assumptions about functional form were used to advantage at the economic agent level, rather than at the economic system level.

Neural networks proved to have a valuable role in modelling the learning and decision making behavior of individual agents in an economic system. In this role, the neural networks perform as a computational model of the economic system that is centered within agents of the system.

This accomplishes two purposes:

1. The agents have meta-observer status. That is, they can observe the system from a remote vantage point and observe the emergent properties of the system, just as the researcher does.

2. The agents learn from experience in the market by optimizing without assigning a particular functional form to the behavior of others.

The distributed form of the blackboard market allows the researcher wide latitude in designing and testing different types of agents and strategies for use in the market. Three general types of agents were developed and tested:

1. Satisficing

2. Stackelberg

3. Neural

Unlike many market models, these agents actually make both price and production decisions. Price, quantity and wealth are emergent variables in these markets, and the agents must use some decision making strategy to estimate them.

Evaluating the fitness of agents is a complex problem because performance of each agent depends on the behavior of the other agents in the system. The focus of this research was on the consequences of the decision making behavior of the producers in the market. The total wealth produced by the system was used as a guide to estimating the global efficiency of the market. The statistical properties of the emergent price and quantity series were also examined to determine the impact of individual agents on the market.

Several conclusions about the role of agents and their strategies are possible. Perhaps the most critical observation is that knowledge about the market greatly enhances the wealth of all participants. There was a distinct separation in the results emerging from markets where at least one agent had some knowledge about the market. The source of this knowledge was less important than having the knowledge, even if the knowledge was not perfect. A single agent with accurate knowledge profoundly changed the stability and wealth of the market, generally to the benefit of all agents.

Markets consisting only of satisficing agents without knowledge of consumer demand produced wealth that was grossly inferior to markets where at least one Stackelberg or neural agent was included. However, the universe of satisficing strategies is huge and only a few, very simple satisficing strategies were sampled. It is conceivable that other strategies could be developed that rival the knowledge based agents.

The Stackelberg agents were endowed with knowledge of the aggregate demand curve, but have no knowledge of the strategies and intentions of other producers in the market. These agents were very regular and predictable in all the markets, but were unable to learn from experience. It is difficult to imagine how the knowledge of the market could have emerged naturally from the operation of this agent in the market because these agents have no knowledge acquisition ability. When compared with other classes of agents, the Stackelberg agents do not necessarily capture more personal wealth in mixed markets.

The success of the Stackelberg agent is very dependent on the accuracy of the knowledge of the demand curve. If the Stackelberg agent is provided with inaccurate information about the average demand curve, the results may be disastrous for the agent. The agent with inaccurate information may quickly leave the market because of poor decisions.

The neural agents gather knowledge from participation in markets This knowledge is stored in the pattern of internal connection weights. The agents learn by backpropagating errors through the networks. This gives a plausible route for the emergence of knowledge by agents in economic systems. The neural agents begin as naive agents with no knowledge of the market, but learn about the market through participation, and additionally, the agents continue to evolve and respond to changes in the market.

While the presence of at least one agent with knowledge about the market is important to the market as a whole, knowledge does not guarantee higher profits to the agent that possess it. Learning appears to capture key features of the market, and suggests favorable levels for production and prices, but does not enable agents to learn the details of what is essentially a white noise process that underlies a random walk. Rule based satisficing agents often do as well or better than knowledge based agents in mixed markets. The variety in strategies and types of agents is important for the flexibility and adaptability of the market as a whole. The satisficing and neural agents tend to explore prices and quantities that are away from the mean in an effort to increase profits, but they do not always profit from such exploration. They do, however, contribute to the randomness of the market, and thus unpredictability of the market.

When some of the agents have the ability to learn, or are continually searching for a higher profit position, the resulting outcome is uncertain. This may lead to instabilities in markets which appear as irregular shifts in production and price after a period of stability. Such irregularities emerge naturally from the interaction of agents in the market even in the absence of natural variations (caused by weather conditions, for instance). These changes appear to be similar to variations that sometimes appear in natural markets. Additionally, the distributed computer markets may be unstable, and reside in a non-optimal state for extended periods.

The time series emerging from most of the market processes are white and therefore are apparently random. This is a general validation of the agents and the distributed market only because a market that is easily predictable is unlike a natural market. In the blackboard market a single agent can have a dramatic impact on the market as a whole, but no individual agent or strategy was superior in all markets.

REFERENCES

Aoki, M., 1979, "Interaction Among Economic Agents Under Imperfect Information: An Example", in New Trends in Dynamic System Theory and Economics, edited by M. Aoki, A, Marzollo, Academic Press.

Arthur, W. B., 1991. "Designing Agents that Act Like Human Agents: A Behavioral Approach to Bounded Rationality", American Economic Review, Vol. 81, No. 2, 353-359.

Auerbach, A., 1982, "The Index of Leading Indicators: "Measurement Without Theory, Thirty-Five Years Later", The Review of Economics and Statistics 64.

Baum, E., & Wilczek, F., 1988, "Supervised Learning of Probability Distributions by Neural Networks", in Neural Information Processing Systems, D. Anderson Editor, American Institute of Physics.

Berndt, E., & Wood, D., 1975, "Technology, Prices, and the Derived Demand For Energy", The Review of Economics and Statistics, No. 3 259-268.

Berndt, E., & Khaled, M., 1979, "Parametric Productivity Measurement and Choice among Flexible Functional Forms", Journal of Political Economy 87.

Boulding, K., 1970, Economics as a Science, McGraw Hill, New York.

Boulding, K., 1981 Evolutionary Economics, Sage, Beverly Hills.

Bray, M.M & Savin, N. E., 1986. "Rational Expectations Equilibrium, Learning, and Model Specification", Econometrica, Vol. 54, No 5, 1129-1160.

Brock, W., A., 1987. "Nonlinearity and Complex Dynamics in Economics and Finance", in The Economy as an Evolving Complex System, editors: P. W. Anderson, K. J. Arrow, & D. Pines, Addison-Wesley.

Cherkassky, V. & Lari-Najafi, H, 1991, "Constrained Topological Mapping for Nonparametric Regression Analysis", Neural Networks, Vol. 4, pp. 27-40.

Chicago Board of Trade, 1989. Commodities Trading Handbook.

David, A., & Pao, Y., 1989, "A Methodology for Capturing Technology via Neural Networks", in Expert Systems in Economics, Banking, and Management, L.F. Pau et al. (Editors), Elsevier Science Publishers.

Dutta, S, & Shekhar, S., 1989, "An Artificial Intelligence Approach to Predicting Bond Ratings", in Expert Systems in Economics Banking and Management, ed. L.F Pau et al, Elsevier Science Publishers.

Dutta, S., & Shekhar, S. 1988. "Bond Ratings: A Non-Conservative Application of Neural Networks", International Joint Conference on Neural Networks.

Engelmore, R., and Morgan, T. (1988) Blackboard Systems, Addison-Wesley.

Fourgeaud, C., Gourieroux, C., & Pradel, J., 1986. "Learning Procedures and Convergence to Rationality", Econometrica, Vol. 54, No. 4, 845-868.

Fox, K. & Miles, D., 1987, Systems Economics: Concepts, Models, and Multidisciplinary Perspectives, Iowa State University Press.

Friedman, D., 1991. "Evolutionary Games in Economics", Econometrica, Vol. 59, No. 3, 637-666.

Hayes-Roth, B. (1985) "A Blackboard Architecture for Control", Artificial Intelligence 26 (2), 251-321.

Hayes-Roth, B., Buchanan, B., Lichtarge, O., Hewett, M., Altman, R., Brinkley, J., Cornelius, C., Duncan, B. and Jardezky, O (1986) "PROTEAN: Deriving Protein Structure from Constraints". In: Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86), pp. 904-9.

Hays-Roth, B., Garvey, A., Johnson, M., and Hewett, M. (1986) "A Layered Environment for Reasoning about Action", Chapter 29 in Blackboard Systems, Englemore & Morgan.

Hebb, D., 1949, Organization of Behavior, Wiley.

Hinton, G., & McClelland, J., 1988, "Learning Representations by Recirculation", in Neural Information Processing Systems, D. Anderson Editor, American Institute of Physics.

Holland, J. H., & Miller, J. H., 1991. "Artificial Adaptive Agents in Economic Theory", American Economic Review, Vol. 81, No.2, 365-370.

Hopfield, J., 1982, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proceedings of the National Academy of Science.

Hoptroff, T., 1991. "Forecasting Economic Turning Points with Neural Networks", International Joint Conference on Neural Networks.

Hornik, Stinchcome, & White, 1990, "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks", Neural Networks, Vol. 3, 551-560.

Intriligator, M., 1971, Mathematical Optimization and Economic Theory, Prentice-Hall.

Irino, T., & Kawahara, H., 1990, "A Method for Designing Neural Networks Using Nonlinear Multivariate Analysis: Application to Speaker-Independent Vowel Recognition", Neural Computation 2, 386-397.

Kamijo, K., & Taniyawa, T., 1990. "Stock Price Pattern Recognition: A Recurrent Neural Network Approach", International Joint Conference on Neural Networks.

Kimoto, T., Asakawa, K., Yoda, M., & Takeoka, M., 1990. "Stock Market Prediction System with Modular Neural Networks", International Joint Conference on Neural Networks.

Kin, C., Hwee, T., 1989, "Connectionist Expert Systems for Intelligent Advisory Applications", in Expert Systems in Economics, Banking, and Management, L.F. Pau et al. (Editors), Elsevier Science Publishers.

Kiyotaki, N., & Wright, R., 1989. "On Money as a Medium of Exchange", Journal of Political Economy, Vol. 97, No. 41, 927-953.

Klimasauskas, C., Guiver, J., & Pelton, G., 1989, Neural Computing, Volume 1, NeuralWare INC.

Langton, C. G., 1991. Artificial Life II, SFI Studies in Complexity, vol. X, edited by C.G. Langton, J.D. Farmer, & S. Rasmussenm Addison-Wesley.

Lapedes, A., & Farber, R., 1988, "How Neural Nets Work", in Neural Information Processing Systems, D. Anderson Editor, American Institute of Physics.

Lendaris, G.G, 1986, "On Systemness and the Problem Solver: Tutorial Comments", IEEE Transactions on Systems Man, and Cybernetics, July / August, 1986 Volume SMC-14 no. 4. 604-610.

Lendaris G. G., 1990a, "A Proposal for Indicating Quality of Generalization when Evaluating ANNS", Submitted to International Joint Conference on Neural Networks 90- San Diego, CA.

Lendaris, G. G., 1990b, Class notes from SYSC 510I Spring 1990, Artificial Intelligence: Neural Networks, Portland State University.

Lendaris, G. G., Zwick, M., and Mathia, K., 1993, "On Matching ANN Structure to Problem Domain Structure", Submitted to World Congress on Neural Networks-93.

Leong, H., 1988, Optimization with Artificial Neural Networks Systems: A Mapping Principle and a Comparison to Gradient Methods", in Neural Information Processing Systems, D. Anderson Editor, American Institute of Physic.

Lesser, V.R., and Corkhill, D.D., (1983) "The Distributed Vehicle Monitoring Testbed: a tool for Investigating Distributed Problem Solving Networks". AI Magazine 4(3), 15-33.

Levy, S., 1992. Artificial Life, The Quest for a New Creation, Pantheon Books, New York.

Lin, K.-P. & Perry, S., 1989, "Analysis of Data for Economic Rationality: An Expert Systems Approach", in Expert Systems in Economics Banking and Management, ed. L.F Pau et al, Elsevier Science Publishers.

Lin K.-P. & Perry, S., 1988, "A Logic Programming Approach to Revealed Preference Theory", Computer Science in Economics and Management, 1.

Linsker, R., 1988a, "Self-Organization in a Perceptual Network", IEEE, March, 105-117.

Linsker, R., 1988b, "Towards an Organizing Principle for a Layered Perceptual Network", in Neural Information Processing Systems, D. Anderson Editor, American Institute of Physics.

MacKay, D., & Miller, K., 1990, "Analysis of Linsker's Simulation of Hebbian Rules", Neural Computation, 2, 173-187.

Mandelbrot, B. B., 1960. "The Pareto-Levi Law and the Distribution of Income". International Economic Review 1, 79-106.

Mandelbrot, B. B., 1963a. "The Variation of Certain Speculative Prices". Journal of Business (Chicago) 36, 394-419.

Mandelbrot, B.B., 1963b. "New Methods in Statistical Economics". Journal of Political Economy 71, 421-440.

Mandelbrot, B.B., 1983. The Fractal Geometry of Nature, W.H. Freeman and Company, New York.

Marimon, R., McGrattan, E., & Sargent, T., 1990, "Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents", Journal of Economic Dynamics and Control, 14, 465-490.

Minsky, M., & Papert, S., Perceptrons, 1969, revised edition 1988, MIT Press.

McCulloch, W. & Pitts, W., 1943, "A Logical Calculus of the Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics, 7, 115-133.

Murphy,R. E., 1965. Adaptive Processes in Economic Systems, Academic Press.

Naylor, T. H., 1971. Computer Simulation Experiments with Models of Economic Systems, Wiley.

Newell, A. (1962) "Some Problems of the Basic Organization of Problem Solving Programs". In: Proceedings of the Second Conference on Self-Organizing Systems, Yovit, M.C., and Goldstein, G.D., (eds), pp. 393-423, Spartan Books.

Nii, P., (1986a) "Blackboard Systems" (Part 1), AI Magazine 7(2), 38-53.

Nii, P., (1986b) "Blackboard Systems" (Part 2), AI Magazine 7(3), 82-106.

Nii, P., and Feigenbaum, E. A. (1978) "Rule-Based Understanding of Signals", IN: Waterman and Hayes-Roth. Pattern Directed Inference Systems, Academic Press.

Odom, M., & Sarda, R., 1990. "A Neural Network for Bank Failure Prediction", International Joint Conference on Neural Networks, 1990, San Deigo, CA..

Perlovsky, L. & McManus, M, 1991, "Maximum Likelihood Neural Networks for Sensor Fusion and Adaptive Classification", Neural Networks, Vol. 4. pp. 89-102.

Pindyck, R., & Rubinfeld, D., 1981, Economic Models & Economic Forecasts, McGraw-Hill.

Plott, C. R., 1982. "Industrial Organization Theory and Experimental Economics", Journal of Economic Literature, Vol. XX, 1485-1527.

Rich, E. & Knight, K., 1989, Artificial Intelligence, second edition, McGraw-Hill, draft.

Robbins, L., 1952, The Nature and Significance of Economic Science, MacMillan, p16.

Rosenblatt, 1962, Principles of Neurodynamics, Spartan Books.

Rumelhart & McClelland, 1986, Parallel Distributed Processing, Bradford.

Scheinkman, J.A., & Le Baron, B., 1990. "Nonlinear Dynamics and Stock Market Returns", Journal of Business.

Sietsma, J. & Dow, R., 1991, "Creating Artificial Neural Networks that Generalize", Neural Networks, Vol. 4. pp. 67-79.

Simpson, P. 1989, Artificial Neural Systems, Pergamon Press.

Simon, H., (1977) "Scientific Discovery and the Psychology of Problem Solving". In: Models of Discovery, Reidel, Boston, MA.

Simon, H., 1979, "Rational Decision Making in Business Organizations", American Economic Review, Vol. 69, no 4, Sept. pp 493 - 513.

Simon, H., 1981, The Sciences of the Artificial, MIT.

Simon, H., 1982, Models of Bounded Rationality: Economic analysis and Public Policy, Volume 1, MIT.

Smith, V., 1982, "Microeconomic Systems as an Experimental Science", American Economic Review, Dec, pp 923 - 955.

Specht, D., 1990, "Probabilistic Neural Networks", Neural Networks, Vol. 3, pp. 109-118.

Surkan, A., & Ying, X., 1991. "Training Neural Networks for Deriving Bond Rating Formulas", International Joint Conference on Neural Networks, 1991, Seattle, WA.

Terry, A. (1983) The CRYSALIS Project: Hierarchical Control of Production Systems. Technical Report HPP-83-19, Stanford University.

Tommerlein, I.D., Levitt, R. E. and Hayes-Roth, B (1987) Using Expert Systems for the layout of Temporary Facilities on Construction Sites. In: Proceedings of CIB (International Council of Building) W-65 Symposium on Organization and Management of Construction, London.

Touretzky, D., & Pomerleau, D., 1989, "What's Hidden in the Hidden Layers?", BYTE, August, 227-233.

van der Maas, H., Verschure, P., & Molenaar, P., 1990, "A Note on Chaotic Behavior in Simple Neural Networks", Neural Networks, Vol. 3, pp. 119-122.

Varian, H., 1984a, Microeconomic Analysis, Second Edition, Norton.

Varian, H., 1984b, "The Nonparametric Approach to Production Analysis", Econometrica, 52.

Weigend, A., Huberman, B., & Rumelhart, D., 1990, "Predicting the Future: A Connectionist Approach", International Journal of Neural Systems, 1 193-209.

Weigend, A., Rumelhart, D., & Huberman, B., 1991, "Generalization by Weight-Elimination with Application to Forecasting", Neural Information Processing 3 (NIPS*90), edited by Lippmann, Moody, and Touretzky, Kaufmann, April, 1991.

Weigend, A., Rumelhart, D., Huberman, B., 1991. "Generalization by Weight Elimination Applied to Currency Exchange Rate Prediction", International Joint Conference on Neural Networks.

White, H., 1989a, "Some Asymptotic Results for Learning in Single Hidden-Layer Feedforward Network Models", Journal of the American Statistical Association, December, Vol. 84, No. 408, 1003-1013.

White, H., 1989b, "Neural-Network Learning and Statistics", AI EXPERT, December, 48-52.

White, H., 1990, "Connectionist Nonparamentic Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings", Neural Networks, Vol. 3, pp. 535-549.

APPENDIX A

EXAMPLE OUTPUT FILES

/*bb.xlt*/

maxt	5
maxp	4
maxr	1
time	6
period	5
round	2
producers	4
consumers	4
total agents	8

ID	Program	Active	Type	Wealth	Coef1	Coef2	Pe	P	Qe	Q
1	pc000.exe	1	p	18109.	-0.0442	-0.2687	39.89	38.09	168.79	0.00
2	ps3.exe	1	p	5241.	-0.0726	-0.2225	44.21	39.90	39.69	0.00
3	ps7.exe	1	p	8251.	0.0953	-0.2624	44.21	37.90	93.02	0.90
4	pn2.exe	1	p	14340.	0.0958	0.1395	35.93	37.02	183.82	0.00
5	cons09.exe	1	c	8405.	-0.0878	-0.0049	27.00	27.00	19.00	19.00
6	cons09.exe	1	c	9449.	-0.0011	-0.2510	21.00	21.00	31.00	31.00
7	cons09.exe	1	c	11489.	-0.0824	-0.0184	20.00	20.00	22.00	22.00
8	cons09.exe	1	c	8059.	-0.0553	-0.2864	27.00	27.00	34.00	34.00

total wealth	83345.
ending price	485.33
ending quantity	0.00
ending round price	38.70
ending round quantity	1269.89
ending round value	49149.
ending market price	38.70
ending market quantity	1270.
ending market value	49149.

/***/

hist.xlt

r	p	t	p1	p2	p3	p4	q1	q2	q3	q4	cons	p	q	prod
1	1	1	46.30	27.29	27.29	28.00	60.88	30.00	30.00	36.82				
1	1	1	46.30	27.29	27.29	28.00	60.88	30.00	30.00	0.00	c5	28.00	36.82	p4
1	1	1	46.30	27.29	27.29	28.00	60.88	30.00	30.00	0.00				
1	1	1	46.30	27.29	27.29	28.00	60.88	0.00	30.00	0.00	c7	27.29	30.00	p2
1	1	1	46.30	27.29	27.29	28.00	60.88	0.00	30.00	0.00				
1	1	1	46.30	27.29	27.29	28.00	60.88	0.00	0.00	0.00	c6	27.29	30.00	p3
1	1	1	46.30	27.29	27.29	28.00	60.88	0.00	0.00	0.00				
1	1	1	46.30	27.29	27.29	28.00	51.20	0.00	0.00	0.00	c8	46.30	9.67	p1
1	1	2	48.40	28.66	28.66	28.63	51.20	0.00	0.00	0.00				
1	1	2	48.40	28.66	28.66	28.63	48.20	0.00	0.00	0.00	c6	48.40	3.00	p1

r	p	t	p1	p2	p3	p4	q1	q2	q3	q4	cons	p	q	prod
1	1	2	48.40	28.66	28.66	28.63	48.20	0.00	0.00	0.00				
1	1	2	48.40	28.66	28.66	28.63	41.03	0.00	0.00	0.00	c7	48.40	7.17	p1
1	1	2	48.40	28.66	28.66	28.63	41.03	0.00	0.00	0.00				
1	1	2	48.40	28.66	28.66	28.63	33.59	0.00	0.00	0.00	c5	48.40	7.44	p1
1	1	2	48.40	28.66	28.66	28.63	33.59	0.00	0.00	0.00				
1	1	2	48.40	28.66	28.66	28.63	28.00	0.00	0.00	0.00	c8	48.40	5.59	p1
1	1	3	48.40	28.66	28.66	30.34	28.00	0.00	0.00	0.00				
1	1	3	48.40	28.66	28.66	30.34	20.56	0.00	0.00	0.00	c5	48.40	7.44	p1
1	1	3	48.40	28.66	28.66	30.34	20.56	0.00	0.00	0.00				
1	1	3	48.40	28.66	28.66	30.34	17.56	0.00	0.00	0.00	c6	48.40	3.00	p1
1	1	3	48.40	28.66	28.66	30.34	17.56	0.00	0.00	0.00				
1	1	3	48.40	28.66	28.66	30.34	11.97	0.00	0.00	0.00	c8	48.40	5.59	p1
1	1	3	48.40	28.66	28.66	30.34	11.97	0.00	0.00	0.00				
1	1	3	48.40	28.66	28.66	30.34	4.80	0.00	0.00	0.00	c7	48.40	7.17	p1
1	1	4	48.40	28.66	28.66	31.54	4.80	0.00	0.00	0.00				
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00	c5	48.40	4.80	p1
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00				
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00	c6	31.54	0.00	p4
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00				
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00	c8	28.66	0.00	p3
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00				
1	1	4	48.40	28.66	28.66	31.54	0.00	0.00	0.00	0.00	c7	48.40	0.00	p1
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00				
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00	c5	28.66	0.00	p3
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00				
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00	c6	31.74	0.00	p4
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00				
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00	c7	31.74	0.00	p4
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00				
1	1	5	48.40	28.66	28.66	31.74	0.00	0.00	0.00	0.00	c8	28.66	0.00	p3
1	2	1	45.07	39.02	39.02	31.74	100.41	30.00	39.42	85.82				
1	2	1	45.07	39.02	39.02	31.74	100.41	30.00	39.42	47.83	c8	31.74	37.99	p4
1	2	1	45.07	39.02	39.02	31.74	100.41	30.00	39.42	47.83				
1	2	1	45.07	39.02	39.02	31.74	100.41	30.00	39.42	8.71	c7	31.74	39.12	p4

r	p	t	p1	p2	p3	p4	q1	q2	q3	q4	cons	p	q	prod
1	2	1	45.07	39.02	39.02	31.74	100.41	30.00	39.42	8.71				
1	2	1	45.07	39.02	39.02	31.74	100.41	8.26	39.42	8.71	c6	39.02	21.74	p2
1	2	1	45.07	39.02	39.02	31.74	100.41	8.26	39.42	8.71				
1	2	1	45.07	39.02	39.02	31.74	100.41	8.26	14.05	8.71	c5	39.02	25.37	p3
1	2	2	45.89	40.98	40.98	35.58	100.41	8.26	14.05	8.71				
1	2	2	45.89	40.98	40.98	35.58	100.41	8.26	0.00	8.71	c7	40.98	14.05	p3
1	2	2	45.89	40.98	40.98	35.58	100.41	8.26	0.00	8.71				
1	2	2	45.89	40.98	40.98	35.58	89.94	8.26	0.00	8.71	c8	45.89	10.47	p1
1	2	2	45.89	40.98	40.98	35.58	89.94	8.26	0.00	8.71				
1	2	2	45.89	40.98	40.98	35.58	77.70	8.26	0.00	8.71	c5	45.89	12.24	p1
1	2	2	45.89	40.98	40.98	35.58	77.70	8.26	0.00	8.71				
1	2	2	45.89	40.98	40.98	35.58	77.70	8.26	0.00	0.00	c6	35.58	8.71	p4
1	2	3	45.89	40.98	40.98	36.64	77.70	8.26	0.00	0.00				
1	2	3	45.89	40.98	40.98	36.64	67.23	8.26	0.00	0.00	c8	45.89	10.47	p1
1	2	3	45.89	40.98	40.98	36.64	67.23	8.26	0.00	0.00				
1	2	3	45.89	40.98	40.98	36.64	55.25	8.26	0.00	0.00	c7	45.89	11.98	p1
1	2	3	45.89	40.98	40.98	36.64	55.25	8.26	0.00	0.00				
1	2	3	45.89	40.98	40.98	36.64	43.00	8.26	0.00	0.00	c5	45.89	12.24	p1
1	2	3	45.89	40.98	40.98	36.64	43.00	8.26	0.00	0.00				
1	2	3	45.89	40.98	40.98	36.64	43.00	0.00	0.00	0.00	c6	40.98	8.26	p2
1	2	4	45.89	40.98	40.98	37.48	43.00	0.00	0.00	0.00				
1	2	4	45.89	40.98	40.98	37.48	31.02	0.00	0.00	0.00	c7	45.89	11.98	p1
1	2	4	45.89	40.98	40.98	37.48	31.02	0.00	0.00	0.00				
1	2	4	45.89	40.98	40.98	37.48	23.00	0.00	0.00	0.00	c6	45.89	8.02	p1
1	2	4	45.89	40.98	40.98	37.48	23.00	0.00	0.00	0.00				
1	2	4	45.89	40.98	40.98	37.48	10.76	0.00	0.00	0.00	c5	45.89	12.24	p1
1	2	4	45.89	40.98	40.98	37.48	10.76	0.00	0.00	0.00				
1	2	4	45.89	40.98	40.98	37.48	0.29	0.00	0.00	0.00	c8	45.89	10.47	p1
1	2	5	45.89	40.98	40.98	38.11	0.29	0.00	0.00	0.00				
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00	c6	45.89	0.29	p1
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00				
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00	c5	40.98	0.00	p3
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00				
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00	c7	38.11	0.00	p4

r	p	t	p1	p2	p3	p4	q1	q2	q3	q4	cons	p	q	prod
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00				
1	2	5	45.89	40.98	40.98	38.11	0.00	0.00	0.00	0.00	c8	38.11	0.00	p4
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	134.82				
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	107.71	c5	38.11	27.12	p4
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	107.71				
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	84.14	c6	38.11	23.57	p4
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	84.14				
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	58.54	c8	38.11	25.60	p4
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	58.54				
1	3	1	42.71	43.53	43.53	38.11	136.51	36.85	63.91	31.65	c7	38.11	26.90	p4
1	3	2	41.60	43.53	43.53	36.94	136.51	36.85	63.91	31.65				
1	3	2	41.60	43.53	43.53	36.94	136.51	36.85	63.91	5.75	c6	36.94	25.90	p4
1	3	2	41.60	43.53	43.53	36.94	136.51	36.85	63.91	5.75				
1	3	2	41.60	43.53	43.53	36.94	116.30	36.85	63.91	5.75	c7	41.60	20.21	p1
1	3	2	41.60	43.53	43.53	36.94	116.30	36.85	63.91	5.75				
1	3	2	41.60	43.53	43.53	36.94	97.49	36.85	63.91	5.75	c8	41.60	18.81	p1
1	3	2	41.60	43.53	43.53	36.94	97.49	36.85	63.91	5.75				
1	3	2	41.60	43.53	43.53	36.94	77.04	36.85	63.91	5.75	c5	41.60	20.45	p1
1	3	3	41.60	41.36	41.36	37.41	77.04	36.85	63.91	5.75				
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	63.91	5.75	c7	41.36	20.67	p2
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	63.91	5.75				
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	43.00	5.75	c5	41.36	20.91	p3
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	43.00	5.75				
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	23.72	5.75	c8	41.36	19.29	p3
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	23.72	5.75				
1	3	3	41.60	41.36	41.36	37.41	77.04	16.18	6.64	5.75	c6	41.36	17.08	p3
1	3	4	41.60	41.36	41.36	37.75	77.04	16.18	6.64	5.75				
1	3	4	41.60	41.36	41.36	37.75	56.83	16.18	6.64	5.75	c7	41.60	20.21	p1
1	3	4	41.60	41.36	41.36	37.75	56.83	16.18	6.64	5.75				
1	3	4	41.60	41.36	41.36	37.75	38.02	16.18	6.64	5.75	c8	41.60	18.81	p1
1	3	4	41.60	41.36	41.36	37.75	38.02	16.18	6.64	5.75				
1	3	4	41.60	41.36	41.36	37.75	21.43	16.18	6.64	5.75	c6	41.60	16.59	p1
1	3	4	41.60	41.36	41.36	37.75	21.43	16.18	6.64	5.75				
1	3	4	41.60	41.36	41.36	37.75	0.98	16.18	6.64	5.75	c5	41.60	20.45	p1

r	p	t	p1	p2	p3	p4	q1	q2	q3	q4	cons	p	q	prod
1	3	5	41.60	41.36	41.36	37.96	0.98	16.18	6.64	5.75				
1	3	5	41.60	41.36	41.36	37.96	0.98	0.00	6.64	5.75	c7	41.36	16.18	p2
1	3	5	41.60	41.36	41.36	37.96	0.98	0.00	6.64	5.75				
1	3	5	41.60	41.36	41.36	37.96	0.98	0.00	0.00	5.75	c8	41.36	6.64	p3
1	3	5	41.60	41.36	41.36	37.96	0.98	0.00	0.00	5.75				
1	3	5	41.60	41.36	41.36	37.96	0.98	0.00	0.00	0.00	c6	37.96	5.75	p4
1	3	5	41.60	41.36	41.36	37.96	0.98	0.00	0.00	0.00				
1	3	5	41.60	41.36	41.36	37.96	0.00	0.00	0.00	0.00	c5	41.60	0.98	p1
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	183.82				
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	156.63	c7	37.96	27.19	p4
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	156.63				
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	132.76	c6	37.96	23.87	p4
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	132.76				
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	106.87	c8	37.96	25.89	p4
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	106.87				
1	4	1	39.89	44.21	44.21	37.96	168.79	39.69	93.02	79.46	c5	37.96	27.41	p4
1	4	2	38.09	44.21	44.21	36.94	168.79	39.69	93.02	79.46				
1	4	2	38.09	44.21	44.21	36.94	168.79	39.69	93.02	53.55	c6	36.94	25.90	p4
1	4	2	38.09	44.21	44.21	36.94	168.79	39.69	93.02	53.55				
1	4	2	38.09	44.21	44.21	36.94	168.79	39.69	93.02	25.68	c8	36.94	27.87	p4
1	4	2	38.09	44.21	44.21	36.94	168.79	39.69	93.02	25.68				
1	4	2	38.09	44.21	44.21	36.94	141.64	39.69	93.02	25.68	c5	38.09	27.16	p1
1	4	2	38.09	44.21	44.21	36.94	141.64	39.69	93.02	25.68				
1	4	2	38.09	44.21	44.21	36.94	114.70	39.69	93.02	25.68	c7	38.09	26.94	p1
1	4	3	38.09	42.00	42.00	36.83	114.70	39.69	93.02	25.68				
1	4	3	38.09	42.00	42.00	36.83	87.76	39.69	93.02	25.68	c7	38.09	26.94	p1
1	4	3	38.09	42.00	42.00	36.83	87.76	39.69	93.02	25.68				
1	4	3	38.09	42.00	42.00	36.83	87.76	39.69	93.02	0.00	c	36.83	25.68	p4
1	4	3	38.09	42.00	42.00	36.83	87.76	39.69	93.02	0.00				
1	4	3	38.09	42.00	42.00	36.83	60.60	39.69	93.02	0.00	c5	38.09	27.16	p1
1	4	3	38.09	42.00	42.00	36.83	60.60	39.69	93.02	0.00				
1	4	3	38.09	42.00	42.00	36.83	34.96	39.69	93.02	0.00	c8	38.09	25.64	p1
1	4	4	38.09	39.90	39.90	36.84	34.96	39.69	93.02	0.00				
1	4	4	38.09	39.90	39.90	36.84	8.02	39.69	93.02	0.00	c7	38.09	26.94	p1

r	p	t	p1	p2	p3	p4	q1	q2	q3	q4	cons	p	q	prod
1	4	4	38.09	39.90	39.90	36.84	8.02	39.69	93.02	0.00				
1	4	4	38.09	39.90	39.90	36.84	8.02	15.98	93.02	0.00	c5	39.90	23.71	p2
1	4	4	38.09	39.90	39.90	36.84	8.02	15.98	93.02	0.00				
1	4	4	38.09	39.90	39.90	36.84	8.02	15.98	73.02	0.00	c6	39.90	20.00	p3
1	4	4	38.09	39.90	39.90	36.84	8.02	15.98	73.02	0.00				
1	4	4	38.09	39.90	39.90	36.84	8.02	15.98	50.90	0.00	c8	39.90	22.13	p3
1	4	5	38.09	39.90	37.90	37.02	8.02	15.98	50.90	0.00				
1	4	5	38.09	39.90	37.90	37.02	8.02	15.98	26.91	0.00	c6	37.90	23.99	p3
1	4	5	38.09	39.90	37.90	37.02	8.02	15.98	26.91	0.00				
1	4	5	38.09	39.90	37.90	37.02	8.02	15.98	0.90	0.00	c8	37.90	26.01	p3
1	4	5	38.09	39.90	37.90	37.02	8.02	15.98	0.90	0.00				
1	4	5	38.09	39.90	37.90	37.02	8.02	0.00	0.90	0.00	c7	39.90	15.98	p2
1	4	5	38.09	39.90	37.90	37.02	8.02	0.00	0.90	0.00				
1	4	5	38.09	39.90	37.90	37.02	0.00	0.00	0.90	0.00	c5	38.09	8.02	p1

/**/

trade.xlt

tr	tp	tt	np	nc	r	p	t	pd	c	Q	P	q
1	4	5	4	4	1	1	1	4	5	157.70	28.00	36.82
1	4	5	4	4	1	1	1	2	7	120.88	27.29	30.00
1	4	5	4	4	1	1	1	3	6	90.88	27.29	30.00
1	4	5	4	4	1	1	1	1	8	60.88	46.30	9.67
1	4	5	4	4	1	1	2	1	6	51.20	48.40	3.00
1	4	5	4	4	1	1	2	1	7	48.20	48.40	7.17
1	4	5	4	4	1	1	2	1	5	41.03	48.40	7.44
1	4	5	4	4	1	1	2	1	8	33.59	48.40	5.59
1	4	5	4	4	1	1	3	1	5	28.00	48.40	7.44
1	4	5	4	4	1	1	3	1	6	20.56	48.40	3.00
1	4	5	4	4	1	1	3	1	8	17.56	48.40	5.59
1	4	5	4	4	1	1	3	1	7	11.97	48.40	7.17
1	4	5	4	4	1	1	4	1	5	4.80	48.40	4.80
1	4	5	4	4	1	2	1	4	8	255.66	31.74	37.99
1	4	5	4	4	1	2	1	4	7	217.67	31.74	39.12
1	4	5	4	4	1	2	1	2	6	178.55	39.02	21.74
1	4	5	4	4	1	2	1	3	5	156.81	39.02	25.37
1	4	5	4	4	1	2	2	3	7	131.43	40.98	14.05
1	4	5	4	4	1	2	2	1	8	117.38	45.89	10.47
1	4	5	4	4	1	2	2	1	5	106.91	45.89	12.24
1	4	5	4	4	1	2	2	4	6	94.67	35.58	8.71
1	4	5	4	4	1	2	3	1	8	85.96	45.89	10.47
1	4	5	4	4	1	2	3	1	7	75.49	45.89	11.98
1	4	5	4	4	1	2	3	1	5	63.50	45.89	12.24
1	4	5	4	4	1	2	3	2	6	51.26	40.98	8.26
1	4	5	4	4	1	2	4	1	7	43.00	45.89	11.98

tr	tp	tt	np	nc	r	p	t	pd	c	Q	p	q
1	4	5	4	4	1	2	4	1	6	31.02	45.89	8.02
1	4	5	4	4	1	2	4	1	5	23.00	45.89	12.24
1	4	5	4	4	1	2	4	1	8	10.76	45.89	10.47
1	4	5	4	4	1	2	5	1	6	0.29	45.89	0.29
1	4	5	4	4	1	3	1	4	5	372.10	38.11	27.12
1	4	5	4	4	1	3	1	4	6	344.98	38.11	23.57
1	4	5	4	4	1	3	1	4	8	321.42	38.11	25.60
1	4	5	4	4	1	3	1	4	7	295.82	38.11	26.90
1	4	5	4	4	1	3	2	4	6	268.92	36.94	25.90
1	4	5	4	4	1	3	2	1	7	243.02	41.60	20.21
1	4	5	4	4	1	3	2	1	8	222.81	41.60	18.81
1	4	5	4	4	1	3	2	1	5	204.00	41.60	20.45
1	4	5	4	4	1	3	3	2	7	183.55	41.36	20.67
1	4	5	4	4	1	3	3	3	5	162.88	41.36	20.91
1	4	5	4	4	1	3	3	3	8	141.97	41.36	19.29
1	4	5	4	4	1	3	3	3	6	122.68	41.36	17.08
1	4	5	4	4	1	3	4	1	7	105.60	41.60	20.21
1	4	5	4	4	1	3	4	1	8	85.40	41.60	18.81
1	4	5	4	4	1	3	4	1	6	66.58	41.60	16.59
1	4	5	4	4	1	3	4	1	5	49.99	41.60	20.45
1	4	5	4	4	1	3	5	2	7	29.54	41.36	16.18
1	4	5	4	4	1	3	5	3	8	13.36	41.36	6.64
1	4	5	4	4	1	3	5	4	6	6.72	37.96	5.75
1	4	5	4	4	1	3	5	1	5	0.98	41.60	0.98
1	4	5	4	4	1	4	1	4	7	485.33	37.96	27.19
1	4	5	4	4	1	4	1	4	6	458.14	37.96	23.87
1	4	5	4	4	1	4	1	4	8	434.27	37.96	25.89
1	4	5	4	4	1	4	1	4	5	408.37	37.96	27.41
1	4	5	4	4	1	4	2	4	6	380.96	36.94	25.90
1	4	5	4	4	1	4	2	4	8	355.06	36.94	27.87
1	4	5	4	4	1	4	2	1	5	327.19	38.09	27.16
1	4	5	4	4	1	4	2	1	7	300.03	38.09	26.94
1	4	5	4	4	1	4	3	1	7	273.09	38.09	26.94
1	4	5	4	4	1	4	3	4	6	246.15	36.83	25.68
1	4	5	4	4	1	4	3	1	5	220.47	38.09	27.16
1	4	5	4	4	1	4	3	1	8	193.31	38.09	25.64
1	4	5	4	4	1	4	4	1	7	167.67	38.09	26.94
1	4	5	4	4	1	4	4	2	5	140.73	39.90	23.71
1	4	5	4	4	1	4	4	3	6	117.03	39.90	20.00
1	4	5	4	4	1	4	4	3	8	97.03	39.90	22.13
1	4	5	4	4	1	4	5	3	6	74.90	37.90	23.99
1	4	5	4	4	1	4	5	3	8	50.91	37.90	26.01
1	4	5	4	4	1	4	5	2	7	24.90	39.90	15.98
1	4	5	4	4	1	4	5	1	5	8.92	38.09	8.02

/***/

wealth1.xlt

r	p	t	prod	wealth
1	1	1	1	1000.000000
1	1	2	1	1093.470337
1	1	3	1	2216.454590
1	1	4	1	3339.438965
1	1	5	1	3571.662842
1	2	1	1	3571.662842
1	2	2	1	3019.609375
1	2	3	1	4061.912109
1	2	4	1	5654.062012
1	2	5	1	7614.190918
1	3	1	1	7627.455078
1	3	2	1	6894.910156
1	3	3	1	9368.773438
1	3	4	1	9368.773438
1	3	5	1	12532.950195
1	4	1	1	12573.683594
1	4	2	1	11679.714844
1	4	3	1	13740.297852
1	4	4	1	16777.466797
1	4	5	1	17803.585938
2	5	1	1	18109.072266

/***/

wealth2.xlt

r	p	t	prod	wealth
1	1	1	2	1000.000000
1	1	2	2	1621.174316
1	1	3	2	1621.174316
1	1	4	2	1621.174316
1	1	5	2	1621.174316
1	2	1	2	1621.174316
1	2	2	2	2272.054688
1	2	3	2	2272.054688
1	2	4	2	2610.424072
1	2	5	2	2610.424072
1	3	1	2	2610.424072
1	3	2	2	2379.059326
1	3	3	2	2379.059326
1	3	4	2	3234.083740
1	3	5	2	3234.083740
1	4	1	2	3903.177002
1	4	2	2	3657.838867
1	4	3	2	3657.838867
1	4	4	2	3657.838867
1	4	5	2	4603.603027
2	5	1	2	5241.225586

/***/

wealth3.xlt

r	p	t	prod	wealth
1	1	1	3	1000.000000
1	1	2	3	1616.177612
1	1	3	3	1616.177612
1	1	4	3	1616.177612
1	1	5	3	1616.177612
1	2	1	3	1616.177612
1	2	2	3	2355.666992
1	2	3	3	2931.504883
1	2	4	3	2931.504883
1	2	5	3	2931.504883
1	3	1	3	2931.504883
1	3	2	3	2556.104980
1	3	3	3	2556.104980
1	3	4	3	4924.860352
1	3	5	3	4924.860352
1	4	1	3	5199.427734
1	4	2	3	4675.703125
1	4	3	3	4675.703125
1	4	4	3	4675.703125
1	4	5	3	6356.470703
2	5	1	3	8251.351562

/**/

wealth4.xlt

r	p	t	prod	wealth
1	1	1	4	1000.000000
1	1	2	4	1793.260254
1	1	3	4	1793.260254
1	1	4	4	1793.260254
1	1	5	4	1793.260254
1	2	1	4	1793.260254
1	2	2	4	3753.137939
1	2	3	4	4063.124023
1	2	4	4	4063.124023
1	2	5	4	4063.124023
1	3	1	4	4063.124023
1	3	2	4	7258.182617
1	3	3	4	8215.084961
1	3	4	4	8215.084961
1	3	5	4	8215.084961
1	4	1	4	8433.182617
1	4	2	4	11407.961914
1	4	3	4	13394.458008
1	4	4	4	14340.435547
1	4	5	4	14340.435547
2	5	1	4	14340.435547

APPENDIX B

PROGRAMS

```
//t.c top level program 3/6/92
#include <stdio.h>
#include <process.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include <alloc.h>
#include "shuffle.c"
#include "bb.h"
#include <time.h>
#include <bios.h>


#define SCREEN


int ReadMarketFile(){

    FILE *lpMarketFile;
    char *temp[80];
    int tmp;

    if ((lpMarketFile = fopen("market.","r")) == NULL){
        fprintf(stderr,"Cannot open market file\n");
        exit(1);
        }
    fscanf(lpMarketFile,"%d", &b->maxt);
    fgets(temp,80,lpMarketFile);
    fscanf(lpMarketFile,"%d",&b->maxp);
    fgets(temp,80,lpMarketFile);
    fscanf(lpMarketFile,"%d",&b->maxr);
    fgets(temp,80,lpMarketFile);
    fscanf(lpMarketFile, "%d",&InitialWealth);
    fclose(lpMarketFile);
    return 0;
    }

int ReadAgentFile(){
     int i;
     FILE *lpAgentFile;
     char *line,*tempstring;

     if ((lpAgentFile = fopen("agents.","r")) == NULL){
           fprintf(stderr,"Cannot open  agents file\n");
           exit(1);
           }
    b->producers=b->consumers=0;
    for (i=1;i<MAXAGENTS;i++)
```

```
            {
            if( (fgets(line,80,lpAgentFile)) != NULL)
                {
                TotalAgents++;
                tempstring= strtok(line," ");
                strcpy(a[i].name,tempstring);
                tempstring=strtok(NULL," ");
                if(( strchr(tempstring,'c')) != NULL){
                     a[i].type='c';
                     b->consumers ++ ;
                     }
                else{
                    if( (strchr(tempstring,'p')) != NULL){
                        a[i].type='p';
                        b->producers++;
                        }
                    else
                        a[i].type='e';
                    }
                tempstring=strtok(NULL," ");
                strcpy(a[i].id,tempstring);
            }
        }
        fclose(lpAgentFile);
        if((TotalAgents)<2){
          printf("Error in Agent File, not enough agents\n");
          exit (1) ;
          }
          b->agents=TotalAgents;
          return 0;
}

int *shuffle(int , int *);
/************************/
int main(void){
    int  *pid,*cid,*id,*id1,r,x,y;
    register i;
    char *p[12];
    FILE *lpRandomFile;
    time_t start, stop,t;

    start = time(NULL);
    mpointer = MK_FP(0x0000,0x01e0);
    clrscr();
    d = malloc( sizeof(struct history ));
    if(!d){
            printf("error in malloc d\n");
            exit(1);
            }
    c = farmalloc((unsigned long) MAXAGENTS * sizeof(struct trade ));
    if(!c){
```

```
        printf("error in malloc c\n");
        exit(1);
        }

    b = malloc( sizeof(struct data_b ));
    if(!b){
            printf("error in malloc b\n");
            exit(1);
            }
    a = farmalloc((unsigned long) MAXAGENTS * sizeof(struct agent ));
    if(!a){
        printf("error in malloc a\n");
        exit(1);
        }
    ReadAgentFile();
    ReadMarketFile();
    a = farrealloc(a,(unsigned long) (1 + b->agents) * sizeof(struct agent ));
    if(!a){
        printf("error in realloc a\n");
        exit(1);
        }
    c = farrealloc(c,(unsigned long) (1 + b->agents) * sizeof(struct trade ));
    if(!c){
        printf("error in realloc c\n");
        exit(1);
        }
/* set up pointers to shared memory */
    mpointer[0]= FP_SEG(a);
    mpointer[1]= FP_OFF(a);
    mpointer[2]= FP_SEG(b);
    mpointer[3]= FP_OFF(b);
    mpointer[4]= FP_SEG(c);
    mpointer[5]= FP_OFF(c);
    mpointer[6]= FP_SEG(d);
    mpointer[7]= FP_OFF(d);

    itoa(InitialWealth,p,10);
    r= spawnl(P_WAIT,"t1.exe",p,p,NULL);
        if(r==-1){
            printf("error spawning t1 (setup program)\n");
            getch();
            exit(1);
            } // if

    pid= (int *) malloc((1 + b->producers) * sizeof( int));
    if(!pid){
        printf("memory allocation error\n");
        exit(1);
        } // if
    cid= (int *) malloc((1 + b->consumers) * sizeof( int));
    if(!cid){
```

```
        printf("memory allocation error\n");
        exit(1);
        } // if
    id= (int *) malloc((1 + TotalAgents) * sizeof( int));
    if(!id){
        printf("memory allocation error\n");
        exit(1);
        } // if
    id1= (int *) malloc((1 + TotalAgents) * sizeof( int));
    if(!id1){
        printf("memory allocation error\n");
        exit(1);
        } // if
    for(i=1;i<= b->producers;i++) pid[i]=i;
    for(i=1;i<= b->consumers;i++) cid[i]=i;
    for(i=1;i<= b->agents ;i++) id[i]=id1[i]=i;
    if ((lpTrades = fopen("trade.xlt","a+t")) == NULL){
        fprintf(stderr,"Cannot open trade file\n");
        exit(1);
        }
    mpointer[8]= FP_SEG(lpTrades);
    mpointer[9]= FP_OFF(lpTrades);
    if ((lpAve = fopen("ave.xlt","a+t")) == NULL){
        fprintf(stderr,"Cannot open average file\n");
        exit(1);
        }
    mpointer[10]= FP_SEG(lpAve);
    mpointer[11]= FP_OFF(lpAve);
    if ((lpHist = fopen("hist.xlt","a+t")) == NULL){
        fprintf(stderr,"Cannot open HISTORY file\n");
        exit(1);
        }
    mpointer[12]= FP_SEG(lpHist);
    mpointer[13]= FP_OFF(lpHist);
    mpointer[14]= FP_SEG(id1);
    mpointer[15]= FP_OFF(id1);

    #ifdef SCREEN
    textmode(C4350);
    highvideo();
    for (i=50;i>=1;i--)
        printf("%d\n",i);
    #endif

gotoxy(3,50);
srand(biostime(0,0L));
for(i=0;i<1000;i++) random(100);
    for(b->round=1;b->round<=b->maxr;b->round++){
        for(b->period=1;b->period<=b->maxp;b->period++){
            for(b->time=1; b->time <= b->maxt; b->time++ ){
                shuffle(TotalAgents,id);
```

```
            for(i=1;i<=TotalAgents;i++){
            if( a[id[i]].type=='p' && a[id[i]].active==1){
                itoa(id[i],p,10);
                r=spawnl(P_WAIT,a[id[i]].name,p,p,NULL);
                if(r== 1){
                    printf("error spawning producer %d\n",i);
                    exit(3);
                    } // if r
            } // ifa[i]
        }//for i
    shuffle(TotalAgents,id);
    for(i=1;i<=TotalAgents;i++){
        if(a[ id[i] ].type=='c'&& a[ id[i]].active==1){
            shuffle(TotalAgents,id1);
            itoa( id[i] ,p,10);
            r=spawnl(P_WAIT,a[ id[i] ].name,p,p,NULL);
            if(r==1){
                printf("error spawning agent %d\n",i);
                exit(3);
                } // if r
        } // ifa[i]
    } // for i

#ifdef SCREEN
    x=wherex();
    y=wherey();
#endif
gotoxy(70,2);
cprintf("%d %d %d ",b->round,b->period,b->time);
#ifdef SCREEN
    gotoxy(x,y);
#endif

r= spawnl(P_WAIT,"update.exe",NULL);
if(r==-1){
    printf("error spawning update program\n");
    exit(1);
    } //if r

    }   // for b->time
} //  for b->period
} // for b->round

r= spawnl(P_WAIT,"record.exe",NULL);
if(r==-1){
    printf("error spawning record program\n");
    exit(1);
    }
for(i=1;i<=TotalAgents;i++){
    if( a[id[i]].type=='p' && a[id[i]].active==1){
        itoa(id[i],p,10);
```

```
            b->time=1;
            r=spawnl(P_WAIT,a[id[i]].name,p,p,NULL);
            if(r== 1){
                printf("error spawning producer %d\n",i);
                exit(3);
                } // if r
            } // ifa[i]
        }//for i

    farfree(a);
    farfree(b);
    farfree(c);
    farfree(d);
    fclose(lpTrades);
    fclose(lpAve);
    fclose(lpHist);
    free(id);

    stop=time(NULL);
    textmode(C80);
    gotoxy(1,24);
    printf("Elapsed time = %d seconds \n", stop-start);
    return 0;
} //end t.c
/***************************************************************************/

//t1.c blackboard setup program 3/17/92
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "bb.h"
#include <dos.h>

int MakeBB(){
    int i,j;
    unsigned far *mpointer;

    mpointer = MK_FP(0x0000,0x01e0);
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);

    randomize();
    b->time=b->period=b->round=1;
    for(i=1; i <= b->agents; i++){
        c[i].qe=c[i].q=(random(1000) > 500)? 25 + (float) random(11) : 25 -(float) random(11);
        c[i].p=c[i].pe=(random(1000) > 500)? 25 + (float) random(6) : 25 -(float) random(6);
        }
```

```
    for(j=1;j<= b->agents;j++){
        a[j].coef[0]= //(float) i;
        ( ((float) random(RAND_MAX)/RAND_MAX) >= 0.5) ?
        ((float) random(RAND_MAX)/RAND_MAX)*(0.1) :
            -((float) random(RAND_MAX)/RAND_MAX)*(0.1) ;
        a[j].coef[1]= //(float) i;
        ( ((float) random(RAND_MAX)/RAND_MAX) >= 0.5) ?
            ((float) random(RAND_MAX)/RAND_MAX)*(0.3) :
            -((float) random(RAND_MAX)/RAND_MAX)*(0.3) ;
    }

    for(i=1; i <= b->agents; i++) {
        a[i].wealth=(float) InitialWealth ;
        a[i].active=1;
    }


    d->cprice=0;
    d->cquantity=0;
    d->periodp=0;
    d->periodq=0;
    d->periodv=0;
    d->roundp=0;
    d->roundq=0;
    d->roundv=0;
    d->marketp=0;
    d->marketq=0;
    d->marketv=0;

for(i=1;i<= b->agents;i++){
    if(a[i].type=='p'){
        d->periodq += c[i].q;
        d->periodv += c[i].p*c[i].q;
        }
    }
    d->periodp1 = d->periodv/d->periodq;
    d->periodq1=d->periodq;
    d->roundp1= d->periodp1;
    d->roundq1= d->periodq1;
    d->periodq=0;
    d->periodv=0;
    return 0;
    }
/**************************/

int main(int argc, char *argv[])
    {
    InitialWealth= atoi(argv[1]);
    MakeBB();
    return 0;
    }
```

```
/*********************************************************************************/
//update.c 7/21/92
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <dos.h>
#include "bb.h"
/*************************/
int main(int argc, char *argv[]){
    int i;
    unsigned far *mpointer;

    mpointer = MK_FP(0x0000,0x01e0);
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    lpAve=MK_FP(mpointer[10], mpointer[11]);
    if(d->periodq >0){
        fprintf(lpAve,"%#8.2f\t %#8.2f\t %#8.2f\n",d->periodv, d->periodv / d->periodq, d-
>periodq);
    }
    if(b->time >= b->maxt && b->round <= b->maxr){
        d->cprice=0;
        for(i=0;i<b->agents;i++)
            if(a[i].type=='p')
            d->cprice += c[i].qe;
            d->periodp1 = d->periodp;
            d->periodq1 = d->periodq;
            d->cquantity=0;
            d->periodp=0;
            d->periodq=0;
            d->periodv=0;
            fprintf(lpAve,"\t\t\t\t%#8.2f\t %#8.2f\n",d->periodp1, d->periodq1);
            if(b->period >= b->maxp && b->round < b->maxr){
                d->roundp1=d->roundp;
                d->roundq1=d->roundq;
                d->cquantity=0;
                d->periodp=0;
                d->periodq=0;
                d->periodv=0;
                d->roundp=0;
                d->roundq=0;
                d->roundv=0;
                }
            }
    return 0;
    }
/******************************************************************************/
```

```
//record.c 7-21-92
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>
#include "bb.h"

FILE *lpBB;

    int main (){
        int i,j;
        float TotalWealth=0;
        unsigned far *mpointer;
    mpointer = MK_FP(0x0000,0x01e0);
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    if ((lpBB = fopen("bb.xlt","w+t")) == NULL){
        fprintf(stderr,"Cannot open blackboard file\n");
        exit(1);
        }
    fprintf(lpBB,"maxt \t%d\n", b->maxt);
    fprintf(lpBB,"maxp \t%d\n", b->maxp);
    fprintf(lpBB,"maxr \t%d\n", b->maxr);
    fprintf(lpBB,"time \t%d\n", b->time);
    fprintf(lpBB,"period \t%d\n", b->period);
    fprintf(lpBB,"round \t%d\n", b->round);
    fprintf(lpBB,"producers \t%d\n", b->producers);
    fprintf(lpBB,"consumers \t%d\n", b->consumers);
    fprintf(lpBB,"total agents \t%d\n", b->agents);
    fprintf(lpBB,"\nID Program    Act Type Wealth Coef1   Coef2  Pc     P    Qc     Q\n");
    for(i=1;i<=b->agents;i++) {
        fprintf(lpBB,"%d  %s  %d    %c %#6.0f %#8.4f %#8.4f  %#6.2f  %#6.2f  %#6.2f  %#6.2f\n",
i,          a[i].name, a[i].active, a[i].type, a[i].wealth, a[i].coef[0], a[i].coef[1], c[i].pc, c[i].p, c[i].qc,
c[i].q);
        TotalWealth += a[i].wealth;
        }
     fprintf(lpBB,"\ntotal wealth       \t%#8.0f\n",TotalWealth);
     fprintf(lpBB,"ending price        \t%#8.2f\n", d->cprice) ;
     fprintf(lpBB,"ending quantity      \t%#8.2f\n", d->cquantity) ;
     fprintf(lpBB,"ending period price   \t%#8.2f\n", d->periodp) ;
     fprintf(lpBB,"ending period quantity\t%#8.2f\n", d->periodq) ;
     fprintf(lpBB,"ending period value   \t%#8.0f\n", d->periodv) ;
     fprintf(lpBB,"ending round price    \t%#8.2f\n", d->roundp) ;
     fprintf(lpBB,"ending round quantity \t%#8.2f\n", d->roundq) ;
     fprintf(lpBB,"ending round value    \t%#8.0f\n", d->roundv) ;
     fprintf(lpBB,"ending market price   \t%#8.2f\n", d->marketp) ;
     fprintf(lpBB,"ending market quantity\t%#8.0f\n", d->marketq) ;
     fprintf(lpBB,"ending market value   \t%#8.0f\n", d->marketv) ;
```

```
    fclose(lpBB);
    return 0;
    } //end record.c

/***************************************************************/
//shuffle.c 7/21/92
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <bios.h>

    int *shuffle(int n,int *p){
    int k,i, temp,index;
    FILE *lpRandomFile;
    long bios_time;

    for (i=1;i<=n;i++) {
        index=random(n)+1;
        temp=p[i];
        p[i]=p[index];
        p[index]=temp;
    }
    return p;
    }
/*******************************************************************/
/*dim2.c*/
#include <stdio.h>
#include <alloc.h>

char  **dim2(int row, int col,unsigned size);
void free2(char **pa);

    void free2(char **pa){
        free(*pa);
        free(pa);
        }

    char  **dim2(int row, int col,unsigned size){
        int i;
        char **prow;
        char *pdata;
        pdata = (char *) malloc((unsigned ) (row) * (col) * size);
        if(! pdata ) {
            fprintf(stderr,"error 1 in allocation of 2 dimensional array \n");
            exit(1);
        }
        prow =(char **) malloc((row) * sizeof(char *));
        if(prow == (char   **) NULL){
            fprintf(stderr,"error 2 in allocation of 2 dimensional array\n");
            exit(1);
        }
```

```
    for(i=0;i< row;i++){
        prow[i]=pdata;
        pdata += size * col;
    }
    return prow;
    }
/*************************************************************/
//util.c
// utilities for agent programs

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
#include <float.h>
#include <string.h>

int id;
char buffer [4096];

void Catcher(int sig, int type){
    int i,j;

    gettext(1,1,80,8,buffer);
    window(1,1,80,8);
    textcolor(BLACK);
    textbackground(WHITE);
    for(j=1;j<9;j++){
        for(i=1;i<80;i++)
            putch(' ');
            putch('\r');
            putch('\n');
        }
    gotoxy(1,1);
    cprintf("Caught Floating Point Error in %s \r\n",a->name);
    cprintf("id= %d \r\n",id);
    cprintf("c[id].p = %f c[id].pe = %f \r\n",c[id].p, c[id].pe);
    cprintf("c[id].q = %f c[id].qe = %f \r\n",c[id].q, c[id].qe);
    cprintf("FPE = %d \r\n", sig);
    cprintf("FPE status before clear = %X \r\n", _status87());
    _clear87();
    cprintf("FPE status after clear = %X \r\n", _status87());
    exit (1);
    }
    FILE *makeFile(char root[4],char id[3],char extension[3]){
    FILE *lpFile;
    char fid[4],fname[12];
```

```
    strcpy(fname,root);
    strcpy(fid,id);
    strcat(fname,fid);
    strcat(fname,extension);
     if ((lpFile = fopen(fname,"a+t")) == NULL){
        fprintf(stderr,"Cannot open %s\n",fname);
        exit(1);
        }
     return lpFile;
     }

char *makeFileName(char root[4],char id[3],char extension[3]){
    char fid[4],fname[12],*name;

    strcpy(fname,root);
    strcpy(fid,id);
    strcat(fname,fid);
    strcat(fname,extension);
    name=&fname;
    return name;
 }
//end of util.c
/*************************************/
//bb.h
#define MAXAGENTS 1100
#define UNITCOST 10
#define UNITUTIL 50

struct agent{
    char  name[13];
    char  type;
    char   id[20];
    float coef[2];
    float wealth;
    int active;
    };

struct data_b{
    int maxt; // maximum number of time steps
    int maxr;
    int maxp;
    int time;
    int period;
    int round;
    int agents;
    int producers;
    int consumers;
    };

struct trade{
    float qe;
```

```
    float q;
    float pe;
    float p;
    };

struct history{
    float cprice;
    float cquantity;
    float periodp;  // average price for the period so far
    float periodq;
    float periodv;
    float roundp;  // average price for the round so far
    float roundq;
    float roundv;
    float marketp;  // average price for the market so far
    float marketq;
    float marketv;
    float periodp1;  //average price in last period
    float periodq1;
    float roundp1;
    float roundq1;
    };

struct agent far *a;
struct data_b far *b;
struct trade far *c;
struct history far *d;
FILE *lpTrades;
FILE  *lpAve;
FILE *lpBB;
FILE *lpHist;
int InitialWealth;
int TotalAgents=0;
unsigned far *mpointer;
double (* func) (double);
//end bb.h
/*************************************************************/

// cons09.c 7/21/92
#include <stdio.h>
#include <math.h>
#include "bb.h"
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include <time.h>
#include <signal.h>
#include <float.h>

#define CUTOFF 0.1
#define SLOPE0 2.0
```

```
// #define HIST //write to history file if defined
// #define TRADES
#define SCREEN

int id;

float demand(float p, int id){
    float q,slope;

    q=100+a[id].coef[1] - (SLOPE0 + a[id].coef[0]) * p;
    q=(q>=0)? q : 0;
    return q;
    }

float demandPrice(float q){
    float p;
    p=(float) (100+ a[id].coef[1] - q )/ (SLOPE0 + a[id].coef[0]);
    p=(p>=0)? p : 0;
    return p;
    }

int eval(int id_p,int id){
    float xd;

    xd= (float) demand( c[id_p].p,id ) - (float) c[id_p].q ;
    xd=(fabs(xd)>= CUTOFF)? ((xd>0)? 1:-1) : 0 ;
    return (int) xd;
    }

void Catcher(int sig, int type){
    char buffer [4096];
    int i,j;

    gettext(1,1,80,8,buffer);
    window(1,1,80,8);
    textcolor(BLACK);
    textbackground(RED);
    for(j=1;j<9;j++){
        for(i=1;i<80;i++)
            putch(' ');
        putch('\r');
        putch('\n');
        }
    gotoxy(1,1);
    cprintf("Caught Floating point Bounds Problem in %s \r\n",a[i].name);
    cprintf("id= %d \r\n",id);
    cprintf("c[id].p = %f c[id].pe = %f \r\n",c[id].p, c[id].pe);
    cprintf("c[id].q = %f c[id].qe = %f \r\n",c[id].q, c[id].qe);
    cprintf("FPE = %d \r\n", sig);
    cprintf("FPE status before clear = %X \r\n", _status87());
```

```
    _clear87();
    cprintf("FPE status after clear = %X \r\n", _status87());
    exit(1);
    }

int main(int argc,char *argv[]){
    int i,j,index,x,p,q;
    int *pid;
    float pmin=1000,qBought=-1, xsmax=-1000,totalQuantity=0,qmax=-1;
    unsigned far *mpointer;
    time_t t;
    char fname[15],fid[4];
    FILE *lpTemp;

    mpointer = MK_FP(0x0000,0x01e0);
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);
    lpHist=MK_FP(mpointer[12], mpointer[13]);
    pid=MK_FP(mpointer[14], mpointer[15]);
    signal(SIGFPE,Catcher);
    if(argc > 0)
        id=atoi(argv[1]);
    if(a[id].wealth <= 0) return 0;
    # ifdef HIST
        fprintf(lpHist,"%#d\t%#d\t%#d\t",b->round,b->period,b->time);
        for(i=1;i<=b->agents;i++) if(a[i].type =='p') fprintf(lpHist,"%#6.2f\t",c[i].p);
        for(i=1;i<=b->agents;i++) if(a[i].type =='p') fprintf(lpHist,"%#6.2f\t",c[i].q);
        fprintf(lpHist,"\n");
    #endif
    /*find total quantity in the market */
    for(i=1;i<=b->agents;i++){
        if(a[pid[i]].type=='p'){
            totalQuantity += c[pid[i]].q;
        }
    }
    /* search for best price - quantity combination
    first compare quantity available and quantity demanded at the given price
    choose min(q_available, q_demanded) for each producer
    second, choose the highest quantity, thus filling the greatest demand at
    the lowest price*/

    index =5000;
    for(i=1;i<=b->agents;i++){
        if(a[pid[i]].type=='p'){
            qBought = max (qmax,  min (c[pid[i]].q, demand (c[pid[i]].p,id ) ) );
            index=(qmax==qBought)? index : pid[i];
            qmax=qBought;
          }
```

```
        }
    if(index != 5000)
        c[index].q -= qBought;
    if(index != 5000 && qBought>0){
        d->cquantity=qBought;
        d->periodq += qBought;
        d->periodv += qBought * c[index].p;
        d->periodp = ( d->periodq )? d->periodv / d->periodq : 0 ;
        d->roundq += qBought;
        d->roundv += qBought * c[index].p;
        d->roundp = ( d->roundq )? d->roundv / d->roundq : 0 ;
        d->marketq += qBought;
        d->marketv += qBought * c[index].p;
        d->marketp = ( d->marketq )? d->marketv / d->marketq : 0 ;
    }
    if(index != 5000 && qBought > 0){
        a[index].wealth += qBought*c[index].p;
        a[id].wealth += qBought*(UNITUTIL - c[id].p);
        }
    #ifdef SCREEN
        if (index != 5000 && qBought > 0){
            highvideo();
            p= (int) 50 - c[index].p;
            if(wherex()>=79){
                clrscr();
                gotoxy(0,0);
                for (i=50;i>=1;i--)
                    printf("%d\n",i);
                    x=3;
            } //if where
        else x=wherex();
        textcolor(index); // make color equal to producer id
        gotoxy(x, p);
        putch('p');
/*************/
    #ifdef TRADES
    if(qBought > 0){
    fprintf(lpTrades,"%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\t%#6.2f\t%#6.2f\t%#6.
2f\n",
        b->maxr,b->maxp,b->maxt,b->producers,b->consumers,
        b->round,b->period,b->time,index,id,totalQuantity,c[index].p,qBought);
    } // if qBought
    #endif
    } // if index != 5000
    #endif

    #ifdef HIST
        fprintf(lpHist,"%#d\t%#d\t%#d\t",b->round,b->period,b->time);
        for(i=1;i<=b->agents;i++) if(a[i].type =='p') fprintf(lpHist,"%#6.2f\t",c[i].p);
        for(i=1;i<=b->agents;i++) if(a[i].type =='p') fprintf(lpHist,"%#6.2f\t", c[i].q);
        fprintf(lpHist,"c \t %#d\t%#6.2f\t%#6.2f\t",id,c[index].p,qBought);
```

```
        fprintf(lpHist,"p\t %#d\n\n",index);
    #endif
    return 0;
    } //end cons09.c
/***********************************************************/
//pc000.c Stackelberg agent CV == 0.0 7/21/92
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include "bb.h"
#include <signal.h>
#include <float.h>
#include <string.h>
#include "util.c"

#define RATE  1.0
#define CV    0.0 // COMPILE WITH DIFFERENT CV FOR EACH STACKELGERG AGENT

int id;
float w=0,b0=50,b1=5;
char buffer [4096];

float dpdq(float qx, float qe, float qa){
    float a0=50, a1=0.5;

    a1 /= (float) (b->maxt * b->consumers);
    return (a0-b1 - a1*(qx + qe ) - a1*CV*qe );
    }
/* cost curve */
float cost(float q){
    return  b0 + b1 * q;
    }
/**************/
float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
    }
/****round*******/
float round(float fptest, int decimals){
    asm{
        fild word ptr decimals
        fld st (0)
        fld dword ptr fptest
        fmul
        frndint
        fdivr
        fstp dword ptr fptest
        }
    return fptest;
    }
```

```
/* base on price */
float expectedPrice(float p, float pe){
    float newPrice;

    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }


float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q;
    }

int main(int argc,char *argv[]){
    int i,j, adjust, decimals=100;
    float timeCriteria, qCriteria, q, sold=0, newSales=0,totalQuantity,fptest;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpTemp, *lpCoef;
    float a1, a2, a3, profit=0, periodProfit, roundProfit, dq, adjustment=1,lastprice;
    char fname[12],fid[4];
    long pos;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);

    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth <= 0){
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
        a[id].active=0;
        return 0;
    }
    lpProfit=makeFile("p00",argv[1],".xlt");
    if(b->time==1 && b->period==1 && b->round==1); //initialization
    else {
        fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
        fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
        fseek(lpProfit,SEEK_CUR,SEEK_END);
        if(b->time==2) newSales=sold=periodProfit=0;
        newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
        sold += newSales;
```

```
        profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )? 0 : newSales * (c[id].p -
unitCost());
        periodProfit += profit;
        roundProfit += profit;
        }
    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",((b->time        ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit
,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
    fclose(lpTemp);
    if(b->time == 2 && b->time < b->maxt){
        totalQuantity=0;
        for(i=1;i<=b->agents;i++){
            if(a[i].type=='p') totalQuantity += c[i].q;
            }
        lastprice=c[id].p;
        c[id].p  = 50  - ( totalQuantity   ) / ( 2 * (b->maxt- b->time+1 ) * b->consumers ) ;
        c[id].p=(c[id].p>b1)? c[id].p : b1;  // use marginal cost rather that average cost
        c[id].p = round(c[id].p,decimals);
        }
    if(b->time==1) {
        dq= (float) dpdq(d->periodq1,c[id].qe,c[id].q)*1.0 ;
        lpTemp=makeFile("q00",argv[1],".xlt");
        dq= (float) dpdq(d->periodq1,c[id].qe,c[id].q)*1.0 ;
        fprintf(lpTemp,"%d\t%d\t%d\t%f\t%f\t%f\t%f\t\n",
            b->round,b->period,b->time,d->periodq1,c[id].qe,c[id].q,dq);
    fclose(lpTemp);
// set q for next period
    q = c[id].qe + dq;
    c[id].q = (cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q : a[id].wealth/unitCost();
    a[id].wealth -=   cost(c[id].q);
    c[id].qe=c[id].q;
    c[id].pe  = 50  - ( d->periodq1 + (dq)  ) / ( 2*(b->maxt)*b->consumers ) ;
/* set price at marginal cost or above (not average cost) */
    c[id].pe = (c[id].pe>b1)? c[id].pe : b1;  //use marginal rather than average cost
    c[id].p = c[id].pe = round(c[id].pe,decimals);
    }
    fclose(lpProfit);
    return 0;
    } // end pc000.c
/*******************************************/
// ps1.c 8/12 based on ps3 but tries to capture
// market share by cutting opening price
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
```

```
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=50,b1=5;

/* cost curve */
float cost(float q){
    return b0 + b1 * q ;
    }

float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
    }

/* base on price */
float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16], ch;
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;
    float lastProfit, deltaProfit;
    long endPosition;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
```

```
b=MK_FP(mpointer[2], mpointer[3]);
c=MK_FP(mpointer[4], mpointer[5]);
d=MK_FP(mpointer[6], mpointer[7]);
lpTrades=MK_FP(mpointer[8], mpointer[9]);
if(argc > 0)
    id=atoi(argv[1]);
if( a[id].wealth < 0) {
    c[id].pe=c[id].p=c[id].qe=c[id].q=0;
    b->producers--;
    a[id].active=0;
    return 0;
    }
/*** noise values are constant throughout the simulation **/
b0 += a[id].coef[1] ;
b1 += a[id].coef[0] ;
lpProfit=makeFile("p00",argv[1],".xlt");
if(b->time==1 && b->period==1 && b->round==1); //initialization
else {
    fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
    fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
    fseek(lpProfit,SEEK_CUR,SEEK_END);
if(b->time==1){
    periodProfit1=periodProfit;
    }
if(b->time==2) {
    newSales=sold=periodProfit=0;
    }
newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
sold += newSales;
profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p - unitCost());
periodProfit += profit;
roundProfit += profit;
}
fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit
,periodProfit,roundProfit);
fclose(lpProfit);
lpTemp=makeFile("wealth",argv[1],".xlt");
fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
fclose(lpTemp);
if(b->time != 1){
    timeCriteria=(float) (b->maxt-b->time)/b->maxt;
    qCriteria= (float) (c[id].qe - c[id].q)/c[id].qe;
    // qCriteria is 1 if no goods have been purchased since the start of the period
    if(timeCriteria >= 0.5 && qCriteria >= 0.5)
        c[id].p *= 1.05;  // increase price if all sold early
    if(timeCriteria < 0.5 && qCriteria < 0.5)
        c[id].p *=0.95;
    c[id].p = (c[id].p > b1)? c[id].p : b1; // use marginal cost
    return 0;
```

```
    } // if price != 1 price adjustment
    if (b->time == 1 && b->period ==1 && b->round==1);
    else {
        if(b->time == 1 && b->period ==2 && b->round ==1 ){
            deltaProfit=0;
            lpProfit=makeFile("datp0",argv[1],".xlt");
            fprintf(lpProfit,"%10.2f\t%10.2f\n",deltaProfit,periodProfit1);
            fclose(lpProfit);
            lastProfit=periodProfit1;
            }
        else{
            lpProfit=makeFile("datp0",argv[1],".xlt");
            endPosition=ftell(lpProfit) -12;
            fseek(lpProfit,endPosition , SEEK_END);//
            fscanf(lpProfit,"%f",&lastProfit);
            fseek(lpProfit,SEEK_CUR,SEEK_END);
            deltaProfit=(periodProfit1 - lastProfit);
            fprintf(lpProfit,"%10.2f\t %10.2f\n", deltaProfit, periodProfit1);
            fclose(lpProfit);
        }
    if(deltaProfit >=0)
        q = c[id].qe + RATE*deltaProfit/((lastProfit+periodProfit1+1)/2)*c[id].qe -c[id].q;
    if(deltaProfit < 0 && c[id].q>0)
        q = c[id].qe  -c[id].q;
    if(deltaProfit < 0 && c[id].q==0)
        q = c[id].qe - RATE*RATE*deltaProfit/((lastProfit+periodProfit1+1)/2)*c[id].qe ;
    q =( q > 0) ? q : 100;
    } // end of else

    if(b->time==1) {
        c[id].pe=d->periodp1 * 0.99;
        c[id].pe=( c[id].pe > unitCost()) ? c[id].pe : unitCost();
        c[id].p = c[id].pe;
        if(b->time== b->period== b->round == 1) q=30;
        if(cost(q) > a[id].wealth) printf("*");
        c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q :
            (a[id].wealth >0)? a[id].wealth/unitCost() : 0;
        a[id].wealth -=  cost(c[id].q);
        c[id].qe=c[id].q;
        }//time==1
        return 0;
}// end ps1.c
/***************************************************************************/

// ps2.c 8/12 based on ps3
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
```

```
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=50,b1=5;

/* cost curve */
float cost(float q){
    return b0 + b1 * q ;
    }
float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
        }

float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16], ch;
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;
    float lastProfit, deltaProfit;
    long endPosition;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
```

```
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);

    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth < 0) {
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
        a[id].active=0;
        return 0;
        }
    /*** noise values are constant throughout the simulation **/
    b0 += a[id].coef[1] ;
    b1 += a[id].coef[0] ;
    lpProfit=makeFile("p00",argv[1],".xlt");
    if(b->time==1 && b->period==1 && b->round==1); //initialization
    else {
        fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
        fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
        fseek(lpProfit,SEEK_CUR,SEEK_END);
        if(b->time==1){
            periodProfit1=periodProfit;
            }
        if(b->time==2) {
            newSales=sold=periodProfit=0;
            }
        newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
        sold += newSales;
        profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p -
unitCost());
        periodProfit += profit;
        roundProfit += profit;
        }
    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
        ((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit
,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
    fclose(lpTemp);
    if(b->time != 1){
        timeCriteria=(float) (b->maxt-b->time)/b->maxt;
        qCriteria= (float) (c[id].qe - c[id].q)/c[id].qe;
        // qCriteria is 1 if no goods have been purchased since the start of the period
        if(timeCriteria >= 0.5 && qCriteria >= 0.5)
            c[id].p *= 1.05;  // increase price if all sold early
        if(timeCriteria < 0.5 && qCriteria < 0.5)
            c[id].p *=0.95;
        c[id].p = (c[id].p > b1)? c[id].p : b1; // use marginal cost
        return 0;
```

```
        } // if price != 1 price adjustment
    if (b->time == 1 && b->period ==1 && b->round==1);
    else {
        if(b->time == 1 && b->period ==2 && b->round ==1 ){
            deltaProfit=0;
            lpProfit=makeFile("datp0",argv[1],".xlt");
            fprintf(lpProfit,"%10.2f\t%10.2f\n",deltaProfit,periodProfit1);
            fclose(lpProfit);
            lastProfit=periodProfit1;
            }
        else{
            lpProfit=makeFile("datp0",argv[1],".xlt");
            endPosition=ftell(lpProfit) -12;
            fseek(lpProfit,endPosition , SEEK_END);//
            fscanf(lpProfit,"%f",&lastProfit);
            fseek(lpProfit,SEEK_CUR,SEEK_END);
            deltaProfit=(periodProfit1 - lastProfit);
            fprintf(lpProfit,"%10.2f\t %10.2f\n", deltaProfit, periodProfit1);
            fclose(lpProfit);
            }
        if(deltaProfit >=0)
            q = c[id].qe + RATE*deltaProfit/((lastProfit+periodProfit1+1)/2)*c[id].qe -c[id].q;
        if(deltaProfit < 0 && c[id].q>0)
            q = c[id].qe   -c[id].q;
    if(deltaProfit < 0 && c[id].q==0)
        q = c[id].qe - RATE*RATE*deltaProfit/((lastProfit+periodProfit1+1)/2)*c[id].qe ;
    q =( q > 0) ? q : 100;
    } // end of else
    if(b->time==1) {
        c[id].pe=d->periodp1;
        c[id].pe=( c[id].pe > unitCost()) ? c[id].pe : unitCost();
        c[id].p = c[id].pe;
    if(b->time== b->period== b->round == 1) q=30;
    if(cost(q) > a[id].wealth) printf("*");
    c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q : (a[id].wealth >0)?
a[id].wealth/unitCost() : 0;
    a[id].wealth -=  cost(c[id].q);
    c[id].qe=c[id].q;
    }//time==1
    return 0;
} // end ps2.c
/*****************************************************************************/

// ps3.c 7/21/92
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
```

```
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=50,b1=5;

/* cost curve */
float cost(float q){
    return b0 + b1 * q ;
    }

float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
        }

float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16], ch;
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;
    float lastProfit, deltaProfit;
    long endPosition;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
```

```
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);

    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth < 0) {
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
        a[id].active=0;
        return 0;
        }
/*** noise values are constant throughout the simulation **/
    b0 += a[id].coef[1] ;
    b1 += a[id].coef[0] ;
    lpProfit=makeFile("p00",argv[1],".xlt");
    fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
    fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
    fseek(lpProfit,SEEK_CUR,SEEK_END);
    periodProfit=(b->time==1 && b->round==1 && b->period ==1)? 0 : periodProfit;
    if(b->time==1 ){
        periodProfit1=periodProfit;
        sold =  newSales= 0 ;
        periodProfit =0;
        roundProfit=(b->period==1)? 0 : roundProfit;

    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
            b->time,c[id].pe, c[id].p, c[id].qe, c[id].q, sold,
            newSales * (c[id].p - unitCost()) ,periodProfit,roundProfit);
        } // end if b->time==1
    else {
        newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
        sold += newSales;
        profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.1 )?  0 :  newSales * (c[id].p - unitCost());
        periodProfit += profit;
        roundProfit += profit;

    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
            b->time,c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit ,periodProfit,roundProfit);
        }  // end else i.e. (time !=1)
    fclose(lpProfit);
    lpProfit=makeFile("p00",argv[1],".xlt");
    if(b->time==1 && b->period==1 && b->round==1); //initialization
    else {
        fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
        fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
        fseek(lpProfit,SEEK_CUR,SEEK_END);
        if(b->time==1){
            periodProfit1=periodProfit;
        }
```

```
        if(b->time==2) {
            newSales=sold=periodProfit=0;
        }
        newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
        sold += newSales;
        profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p -
unitCost());
        periodProfit += profit;
        roundProfit += profit;
        }
    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
        ((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit
,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
    fclose(lpTemp);
    if(b->time != 1){
        timeCriteria=(float) (b->maxt-b->time)/b->maxt;
        qCriteria= (float) (c[id].qe - c[id].q)/c[id].qe;
    if(timeCriteria >= 0.5 && qCriteria >= 0.5)
        c[id].p *= 1.05;  // increase price if all sold early
    if(timeCriteria < 0.5 && qCriteria < 0.5)
        c[id].p *=0.95;
    c[id].p = (c[id].p > b1)? c[id].p : b1; // use marginal cost
     return 0;
    } // if price != 1 price adjustment
    if (b->time == 1 && b->period ==1 && b->round==1);
    else {
        if(b->time == 1 && b->period ==2 && b->round ==1 ){
            deltaProfit=0;
            lpProfit=makeFile("datp0",argv[1],".xlt");
            fprintf(lpProfit,"%10.2f\t%10.2f\n",deltaProfit,periodProfit1);
            fclose(lpProfit);
            lastProfit=periodProfit1;
            }
        else{
            lpProfit=makeFile("datp0",argv[1],".xlt");
            endPosition=ftell(lpProfit) -12;
            fseek(lpProfit,endPosition , SEEK_END);//
            fscanf(lpProfit,"%f",&lastProfit);
            fseek(lpProfit,SEEK_CUR,SEEK_END);
            deltaProfit=(periodProfit1 - lastProfit);
            fprintf(lpProfit,"%10.2f\t %10.2f\n", deltaProfit, periodProfit1);
            fclose(lpProfit);
            }
    if(deltaProfit >=0)
        q = c[id].qe + RATE*deltaProfit/((lastProfit+periodProfit1+1)/2)*c[id].qe -c[id].q;
    if(deltaProfit < 0 && c[id].q>0)
        q = c[id].qe   -c[id].q;
```

```
    if(deltaProfit < 0 && c[id].q==0)
        q = c[id].qe - RATE*RATE*deltaProfit/((lastProfit+periodProfit1+1)/2)*c[id].qe ;
    q =( q > 0) ? q : 100;
    } // end of else
    if(b->time==1) {
        c[id].pe=c[id].p=d->periodp1*1.1;
    if(b->time== b->period== b->round == 1) q=30;
    if(cost(q) > a[id].wealth) printf("*");
    c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q : (a[id].wealth >0)?
a[id].wealth/unitCost() : 0;
    a[id].wealth -=  cost(c[id].q);
    c[id].qe=c[id].q;
    }//time==1
    return 0;
    } // end ps3.c
/*************************************************************************/
// ps7.c 8/12/92
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=50,b1=5;

/* cost curve */
float cost(float q){
    return b0 + b1 * q ;
    }
float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
        }
float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }
float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }
```

```
int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16], ch;
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;
    float lastProfit, deltaProfit;
    long endPosition;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);

    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth < 0) {
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
        a[id].active=0;
        return 0;
        }
/*** noise values are constant throughout the simulation **/
    b0 += a[id].coef[1] ;
    b1 += a[id].coef[0] ;
    lpProfit=makeFile("p00",argv[1],".xlt");
    fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
    fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
    fseek(lpProfit,SEEK_CUR,SEEK_END);
    periodProfit=(b->time==1 && b->round==1 && b->period ==1)? 0 : periodProfit;
    if(b->time==1 ){
        periodProfit1=periodProfit;
        sold = newSales= 0 ;
        periodProfit =0;
        roundProfit=(b->period==1)? 0 : roundProfit;

    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
            b->time,c[id].pe, c[id].p, c[id].qe, c[id].q, sold,
            newSales * (c[id].p - unitCost()) , periodProfit,roundProfit);
        } // end if b->time==1
```

```
else {
    newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
    sold += newSales;
    profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.1 )?  0 :  newSales * (c[id].p - unitCost());
    periodProfit += profit;
    roundProfit += profit;

  fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
        b->time,c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit ,periodProfit,roundProfit);
    }  // end else i.e. (time !=1)
  fclose(lpProfit);
  lpProfit=makeFile("p00",argv[1],".xlt");
  if(b->time==1 && b->period==1 && b->round==1); //initialization
  else {
    fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
    fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
    fseek(lpProfit,SEEK_CUR,SEEK_END);
    if(b->time==1){
        periodProfit1=periodProfit;
    }
    if(b->time==2) {
        newSales=sold=periodProfit=0;
    }
    newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
    sold += newSales;
    profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p -
unitCost());
    periodProfit += profit;
    roundProfit += profit;
    }

  fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
        ((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold,
        profit ,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id,
a[id].wealth);
    fclose(lpTemp);
    if(b->time != 1){
        timeCriteria=(float) (b->maxt-b->time)/b->maxt;
        qCriteria= (float) (c[id].qe - c[id].q)/c[id].qe;
        // qCriteria is 1 if no goods have been purchased since the start of the period
    if(timeCriteria >= 0.5 && qCriteria >= 0.5)
        c[id].p *= 1.05;  // increase price if all sold early
    if(timeCriteria < 0.5 && qCriteria < 0.5)
        c[id].p *=0.95;
    c[id].p = (c[id].p > b1)? c[id].p : b1; // use marginal cost
    return 0;
```

```
    } // if price != 1 price adjustment
  if (b->time == 1 && b->period ==1 && b->round==1);
  else {
    if(b->time == 1 && b->period ==2 && b->round ==1 ){
      deltaProfit=0;
      lpProfit=makeFile("datp0",argv[1],".xlt");
      fprintf(lpProfit,"%10.2f\t%10.2f\n",deltaProfit,periodProfit1);
      fclose(lpProfit);
      lastProfit=periodProfit1;
      }
    else{
      lpProfit=makeFile("datp0",argv[1],".xlt");
      endPosition=ftell(lpProfit) -12;
      fseek(lpProfit,endPosition , SEEK_END);//
      fscanf(lpProfit,"%f",&lastProfit);
      fseek(lpProfit,SEEK_CUR,SEEK_END);
      deltaProfit=(periodProfit1 - lastProfit);
      fprintf(lpProfit,"%10.2f\t %10.2f\n", deltaProfit, periodProfit1);
      fclose(lpProfit);
      }
    q= d->periodq1 / b->producers;
    q =( q > 0) ? q : 100;
    } // end of else
  if(b->time==1) {
    c[id].pe=c[id].p=d->periodp1*1.1;
  if(b->time== b->period== b->round == 1) q=30;
  if(cost(q) > a[id].wealth) printf("*");
    c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q :
    (a[id].wealth >0)? a[id].wealth/unitCost() : 0;
  a[id].wealth -=  cost(c[id].q);
  c[id].qe=c[id].q;
  }//time==1
  return 0;
  } //end ps7.c
/*****************************************************************************/
// ps8.c 8/12/92
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5
```

```
float w=0,b0=50,b1=5;

/* cost curve */
float cost(float q){
    return b0 + b1 * q ;
    }

float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
    }

float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16], ch;
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;
    float lastProfit, deltaProfit;
    long endPosition;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);

    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth < 0) {
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
```

```
        a[id].active=0;
        return 0;
        }
    /*** noise values are constant throughout the simulation **/
    b0 += a[id].coef[1];
    b1 += a[id].coef[0];
    lpProfit=makeFile("p00",argv[1],".xlt");
    if(b->time==1 && b->period==1 && b->round==1); //initialization
    else {
        fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
        fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
        fseek(lpProfit,SEEK_CUR,SEEK_END);
        if(b->time==1){
            periodProfit1=periodProfit;
            }
        if(b->time==2) {
            newSales=sold=periodProfit=0;
        }
        newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
        sold += newSales;
        profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p -
unitCost());
        periodProfit += profit;
        roundProfit += profit;
        }
    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
        ((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold,
        profit ,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
    fclose(lpTemp);
    /*************************/
    if(b->time != 1){
    timeCriteria=(float) (b->maxt-b->time)/b->maxt;
    qCriteria= (float) (c[id].qe - c[id].q)/c[id].qe;
    // qCriteria is 1 if no goods have been purchased since the start of the period
    if(timeCriteria >= 0.5 && qCriteria >= 0.5)
        c[id].p *= 1.05;  // increase price if all sold early
    if(timeCriteria < 0.5 && qCriteria < 0.5)
        c[id].p *=0.95;
    c[id].p = (c[id].p > b1)? c[id].p : b1; // use marginal cost
     return 0;
    } // if price != 1 price adjustment
    if (b->time == 1 && b->period ==1 && b->round==1);
    else {
       if(b->time == 1 && b->period ==2 && b->round ==1 ){
           deltaProfit=0;
           lpProfit=makeFile("datp0",argv[1],".xlt");
           fprintf(lpProfit,"%10.2f\t%10.2f\n",deltaProfit,periodProfit1);
```

```
            fclose(lpProfit);
            lastProfit=periodProfit1;
            }
        else{
            lpProfit=makeFile("datp0",argv[1],".xlt");
            endPosition=ftell(lpProfit) -12;
            fseek(lpProfit,endPosition , SEEK_END);//
            fscanf(lpProfit,"%f",&lastProfit);
            fseek(lpProfit,SEEK_CUR,SEEK_END);
            deltaProfit=(periodProfit1 - lastProfit);
            fprintf(lpProfit,"%10.2f\t %10.2f\n", deltaProfit, periodProfit1);
            fclose(lpProfit);
            }
    q = (c[id].q < c[id].qe * 0.02 ) ? c[id].qe*1.08 : ( c[id].q > c[id].qe * 0.05 ) ? c[id].qe * 0.95 :
c[id].qe;
    q =( q > 0) ? q : 100;
    } // end of else
    if(b->time==1) {
        c[id].pe=c[id].p=d->periodp1 * 1.1;
    if(b->time== b->period== b->round == 1) q=30;
    if(cost(q) > a[id].wealth) printf("*");
    c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q : (a[id].wealth >0)?
a[id].wealth/unitCost() : 0;
    a[id].wealth -=  cost(c[id].q);
    c[id].qe=c[id].q;
    }//time==1
    return 0;
    }
//end ps8.c
/**************************************************************************/

//nett.c
// 7/3/92 see line 18 before compling
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <stdarg.h>
#include <dir.h>
#include "dim2.c"
#include "sig.c"
#include "scale.c"
#include "shuffle.c"

#define LEARNING_RATE 0.10
#define RAND_RANGE 0.1
#define MAX_ROWS 500   // use 500 to compile nett3 and 100 for agents
```

```
#define ran() ( (random(20000) > 10000)? (float)
RAND_RANGE*random(RAND_MAX)/RAND_MAX :(float) -
RAND_RANGE*random(RAND_MAX)/RAND_MAX )

double sigmoid(double);
double hTan(double);
double ihTan(double);
float **dataIn, **dataOut, **w12, **w23;
float *sum,*sumOut,*dOut,*dh;
FILE *lpWeights,*lpOut;
int *who;
float *dataInMax, *dataInMin, *dataOutMax, *dataOutMin, *scaleIn, *scaleOut;

void netset(float **w12, float **w23,int in,int hidden,int out){
    int i, j;
    randomize();
    for(i=1;i<1000;i++) rand(); //exercize random number generator
    for(i=0;i<=in;i++){
        for(j=0; j<hidden ; j++) w12[i][j] = (float) ran();
        }
    for(i=0;i<=hidden;i++){
        for(j=0; j<out ; j++) w23[i][j] = (float) ran();
        }
    }

void printm(float **x,int rows,int cols){
    int i, j;
    for(i=0;i<rows;i++){
        for(j=0;j<cols;j++) printf("%f\t", x[i][j]);
        printf("\n");
     }
     }

int countRows(FILE *fp){
    int rows=0,r1;
    char line[256];
    rewind(fp);
    while(fgets(line,256,fp)) rows++; //count the lines in the file
    rewind(fp);
    if(rows>MAX_ROWS){ //add 6/3/92 to read only most recent MAX_ROW lines
        r1=rows-MAX_ROWS;
        while (r1-- >0) fgets(line,256,fp);
        rows=MAX_ROWS;
        }
    return rows;
    }

int readData(FILE *fp, int rows,int inputs,int outputs){
     int i,j;
// Dimension the input and out matrices
    dataIn =dim2(rows,inputs,sizeof(float));
```

```
    dataOut=dim2(rows,outputs,sizeof(float));
// Read the training and recall data
    for(i=0;i<rows;i++){
        for(j=0;j<inputs;j++) fscanf(fp,"%f", &dataIn[i][j]);
        for(j=0;j<outputs;j++) fscanf(fp,"%f", &dataOut[i][j]);
        }
    fseek(fp,SEEK_CUR,SEEK_END);
    return 0;
}

int readWeights(FILE *lpWeights,float **w12, float **w23, int inputs, int outputs, int hidden){
        int i ,j;
// Read Historical minimum and maximum for all inputs and outputs
    for(j=0;j<inputs;j++) fscanf(lpWeights,"%f ", &dataInMin[j]);
    for(j=0;j<inputs;j++) fscanf(lpWeights,"%f ", &dataInMax[j]);
    for(j=0;j<outputs;j++) fscanf(lpWeights,"%f ", &dataOutMin[j]);
    for(j=0;j<outputs;j++) fscanf(lpWeights,"%f ", &dataOutMax[j]);
    for(i=0;i<=inputs;i++){
        for(j=0; j<hidden ; j++) fscanf(lpWeights,"%f ",&w12[i][j]);
        }
    for(i=0;i<=hidden;i++){
        for(j=0; j<outputs ; j++) fscanf(lpWeights,"%f ", &w23[i][j]);
        }
    }

int allocateMemory(int hidden,int outputs,int inputs){
/*** Set up memory for the net *************/
    sum= (float*) calloc(hidden,sizeof(float));
    if(!sum){
        fprintf(stderr,"error allocating memory for hidden layer\n");
        exit(1);
        }
    sumOut= (float*) calloc(outputs,sizeof(float));
    if(!sumOut){
        fprintf(stderr,"error allocating output layer\n");
        exit(1);
        }
    dh= (float*) calloc(hidden,sizeof(float));
    if(!dh){
        fprintf(stderr,"error allocating memory for hidden layer dh\n");
        exit(1);
        }
    dOut= (float*) calloc(outputs,sizeof(float));
    if(!dOut){
        fprintf(stderr,"error allocating output layer dOut\n");
        exit(1);
        }
    dataInMax= (float*) calloc(inputs,sizeof(float));
    if(!dataInMax){
        fprintf(stderr,"error allocating dataInMax\n");
        exit(1);
```

```
        }
    dataInMin= (float*) calloc(inputs,sizeof(float));
    if(!dataInMin){
        fprintf(stderr,"error allocating dataInMin\n");
        exit(1);
        }
    dataOutMax= (float*) calloc(outputs,sizeof(float));
    if(!dataOutMax){
        fprintf(stderr,"error allocating dataOutMax\n");
        exit(1);
        }
    dataOutMin= (float*) calloc(outputs,sizeof(float));
    if(!dataOutMin){
        fprintf(stderr,"error allocating dataOutMin\n");
        exit(1);
        }
    scaleIn= (float*) calloc(inputs,sizeof(float));
    if(!scaleIn){
        fprintf(stderr,"error allocating scaleIn\n");
        exit(1);
        }
    scaleOut= (float*) calloc(outputs,sizeof(float));
    if(!scaleOut){
        fprintf(stderr,"error allocating scaleOut\n");
        exit(1);
        }
    return 0;
    }

int forwardPass(int hidden,int outputs, int inputs, float *in,float **w12, float **w23,double (*
func) (double) ){
    int i,j;
    for(i=0;i<hidden;i++){
        sum[i]=w12[0][i];
        for(j=0;j<inputs;j++) sum[i] += w12[j+1][i]*in[j];
        sum[i]=(*func) (sum[i]);
        }
    for(i=0;i<outputs;i++){
        sumOut[i]=w23[0][i];
        for(j=0;j<hidden;j++) sumOut[i] += w23[j+1][i]*sum[j];
        sumOut[i]=(*func) (sumOut[i]);
        }
     return 0;
    }

int adjustWeights(int hidden,int outputs,int inputs,int k,float **w12,float **w23, double (* dfunc)
(double))
{
    int i,j;
    float rate=LEARNING_RATE,dHid;
    for(i=0;i<outputs;i++){
```

```
        dOut[i]= ( dataOut[k][i]-sumOut[i] ) * (( *dfunc) (sumOut[i])) ; //general
        w23[0][i] += dOut[i]*rate ;
        for(j=1;j<hidden+1;j++)
            w23[j][i] += dOut[i] * sum[j-1] *rate;
            }
    // adjust weights from input layer to hidden layer
        for(i=0;i<hidden;i++){
            dHid=0;
            for(j=0;j<outputs;j++)
                dHid += dOut[j]*w23[i+1][j];
            dh[i]= ((* dfunc) (sum[i])) * dHid * rate;
            w12[0][i] += dh[i] ;
            for(j=1;j<inputs+1;j++)
                w12[j][i] += dh[i] * dataIn[k][j-1];
            }
            return 0;
        }

int restoreScale(FILE *lpOut,int outputs,int k,double (* func) (double) ){
    float spread;
    int i,j;
    for(j=0;j<outputs;j++){
        spread= dataOutMax[j]-dataOutMin[j];
        if(spread==0)
            fprintf(lpOut,"%#8.4f\t%#8.4f\t", dataOutMax, sumOut[0]);
        else
            fprintf(lpOut,"%#8.4f\t%#8.4f\n",
                (dataOut[k][j]+1)* spread/2+ dataOutMin[j], (sumOut[j]+1)* spread/2+
dataOutMin[j] );
    }
    return 0;
}

saveWeights(FILE *lpWeights,int inputs,int outputs,int hidden,float **w12, float **w23){
    int i,j;
    rewind(lpWeights);
// Save Historical minimum and maximum for all inputs and outputs
    for(j=0;j<inputs;j++) fprintf(lpWeights,"%f\t", dataInMin[j]);
    fprintf(lpWeights,"\n");
    for(j=0;j<inputs;j++) fprintf(lpWeights,"%f\t", dataInMax[j]);
    fprintf(lpWeights,"\n");
    for(j=0;j<outputs;j++) fprintf(lpWeights,"%f\t", dataOutMin[j]);
    fprintf(lpWeights,"\n");
    for(j=0;j<outputs;j++) fprintf(lpWeights,"%f\t", dataOutMax[j]);
    fprintf(lpWeights,"\n");
// Save Weights
    for(i=0;i<=inputs;i++){
        for(j=0; j<hidden ; j++) fprintf(lpWeights,"%f\t",w12[i][j]);
        fprintf(lpWeights,"\n");
        }
    for(i=0;i<=hidden;i++){
```

```
        for(j=0; j<outputs  ; j++) fprintf(lpWeights,"%f\t", w23[i][j]);
        fprintf(lpWeights,"\n");
        }
    fclose(lpWeights);
    return 0;
    }

getWeights(char weightFile[16],int hidden,int inputs,int outputs,float **w12,float **w23){
    int initialize,j;
    struct ffblk ffblk;
    initialize=( findfirst(weightFile,&ffblk,0) )? 0 : 1;
    if(initialize == 0){
        netset(w12, w23, inputs, hidden, outputs);
        if(( lpWeights = fopen(weightFile, "w+t")) == NULL){
            fprintf(stderr, "Can't open weight file 1 %s \n",weightFile);
            exit(1);
            }
        for(j=0;j<inputs;j++)
            dataInMax[j]=dataInMin[j]=dataIn[0][j];
        for(j=0;j<outputs;j++)
            dataOutMax[j]=dataOutMin[j]=dataOut[0][j];
    }
// Open existing weight file.
    else {
        if(( lpWeights = fopen(weightFile, "r+t")) == NULL){
            fprintf(stderr, "Can't open weight file 2 %s \n",weightFile);
            exit(1);
            }
        readWeights(lpWeights, w12, w23, inputs, outputs, hidden);
        } // end of else, done setting / getting  weights
        return 0;
    }

int train(int maxIter,int inputs, int outputs, int rows, int hidden, float **w12, float **w23, float
**in, double  (* func) (double), double (* dfunc) (double) ){
    int i,k,l,iter;
    for(iter=0; iter<maxIter; iter++){
        shuffle(rows,who);
        for(k=0;k<rows;k++){  /* for each row of data */
            l=who[k];
            forwardPass(hidden,outputs, inputs, in[l], w12, w23, (* func) );
            adjustWeights(hidden, outputs, inputs, l, w12, w23, (* dfunc));
        } /***end of k loop  ***/
    } /* end of iter loop */
    return 0;
    }

int trainNet(int inputs,int hidden,int outputs,int iterations,int rows,FILE *lpData,char
weightFile[16]){
    int i,j,k;
    allocateMemory(hidden, outputs,inputs);
```

```
    w12=dim2(inputs+1,hidden,sizeof(float));
    w23=dim2(hidden+1,outputs,sizeof(float));
    readData(lpData,rows,inputs,outputs); //reads data
    getWeights(weightFile,hidden, inputs, outputs, w12, w23);
    for(j=0;j<inputs;j++){
        dataInMin[j]=(dataInMin[j] <= dataIn[rows-1][j])? dataInMin[j] : dataIn[rows-1][j];
        dataInMax[j]=(dataInMax[j] >= dataIn[rows-1][j])? dataInMax[j] : dataIn[rows-1][j];
        }
    for(j=0;j<outputs;j++){
        dataOutMin[j]=(dataOutMin[j] < dataOut[rows-1][j])? dataOutMin[j] : dataOut[rows-1][j];
        dataOutMax[j]=(dataOutMax[j] > dataOut[rows-1][j])? dataOutMax[j] : dataOut[rows-
1][j];
        }
    for(i=0;i<rows;i++){
        for(j=0;j<inputs;j++)  dataIn[i][j]= scale(dataIn[i][j]);
        for(j=0;j<outputs;j++) dataOut[i][j]= scale(dataOut[i][j]);
        }
    who= (float*) calloc(rows+1,sizeof(int));
    if(!who){
        fprintf(stderr,"error allocating memory for hidden layer\n");
        exit(1);
        }
    for(i=0;i<rows;i++) who[i]=i;
    train(iterations, inputs,  outputs, rows, hidden, w12, w23, dataIn, sigmoid, dhSig );
    saveWeights(lpWeights,inputs,outputs,hidden, w12, w23);
    return 0;
    } // end of trainNet

int main(int argc,char *argv[]){
    int i,j,k, maxi, inputs, outputs, rows, hidden_elements,iterations,initialize,hidden;
    FILE *fp,*lpNetProfit;
    char weightFile[16],fid[4],fname[16];
    float *inDatPointer, inDat1[6];
    if( argc<2 ){
        printf("syntax:  net2  data.dat initialize ID# iteration hidden_elements\n");
        exit(1);
        }
    if(( fp= fopen(argv[1], "r")) == NULL){
        fprintf(stderr, "Can't open file %s\n",argv[1]);
        exit(1);
        }
    initialize=atoi(argv[2]);
    maxi=atoi(argv[3]);
    iterations=atoi(argv[4]);
    hidden=hidden_elements=atoi(argv[5]);
    strcpy(weightFile,"w00");
    strcpy(fid,argv[3]);
    strcat(weightFile,fid);
    strcat(weightFile,".txt");
    inputs=6;
    outputs=1;
```

```
    rows=0;

    rows = countRows(fp);
    readData(fp,rows,inputs,outputs); //reads data and scales into 0<x<=1 range
    w12=dim2(inputs+1,hidden,sizeof(float));
    w23=dim2(hidden+1,outputs,sizeof(float));
    getWeights(weightFile,hidden, inputs, outputs, w12, w23, initialize);
    allocateMemory(hidden, outputs, inputs);
    who= (float*) calloc(rows+1,sizeof(int));
    if(!who){
        fprintf(stderr,"error allocating memory for hidden layer\n");
        exit(1);
        }
    for(i=0;i<rows;i++) who[i]=i;
        train(iterations, inputs, outputs, rows, hidden, w12, w23, dataIn,hTan));
     if(( lpOut = fopen("netout.xlt", "a+t")) == NULL){
        fprintf(stderr, "Can't open netout file \n");
        exit(1);
        }
    for(k=0;k<rows;k++){  /* for each row of data */
        forwardPass(hidden,outputs, inputs, dataIn[k], w12, w23,hTan); //compute net output
        restoreScale(lpOut, outputs, k, hTan);
        }/*** end of k loop **/
    saveWeights(lpWeights,inputs,outputs,hidden, w12, w23);
    free2(w12);
    free2(w23);
    return 0;
    } //end nett.c

//sig.c
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <stdlib.h>

double sigmoid(double x){
    int oldstatus,newstatus;
    asm{
        fldl2e;
        fmul qword ptr x;
        fchs;
        fstcw oldstatus;
        fstcw newstatus;
        and newstatus,0f3ffh;
        or newstatus,00c00h;
        fwait;
        fldcw newstatus;
        fld st;
        frndint;
        fldcw oldstatus;
        fsubr st,st(1);
```

```
        f2xm1;
        fld1;
        faddp st(1),st;
        fscale;
        ffree st(1);
        fld1;
        fadd;
        fld1;
        fdiv st,st(1);
        fstp qword ptr x;
        ffree st;
        }
    return x;
    }

double hTanScale(double x){
    int oldstatus,newstatus;
    double scale=0.00003;
    asm{
        fstcw oldstatus;
        fstcw newstatus;
        and newstatus,0f3ffh;
        or  newstatus,00c00h;
        fwait;
        fldcw newstatus;
        fldl2e;
        fmul qword ptr x;
        fmul qword ptr scale;
        fld st;
        frndint;
        fsubr st,st(1);
        fld st
        f2xm1;
        fld1;
        fadd
        fxch st(1)
        fincstp
        fscale;
        fxch
        fchs
        fdecstp
        fchs
        f2xm1;
        fld1;
        fadd
        fscale;
        fxch
        ffree st
        fincstp
        fld  st
        fxch st(2)
```

```
        fadd st(1),st
        fsub st,st(2)
        fdivr
        fstp qword ptr x;
        ffree st
         fldcwoldstatus;
        }
    return x;
    }
double hTan(double x){
    int oldstatus,newstatus;
    asm{
        fstcw oldstatus;
        fstcw newstatus;
        and newstatus,0f3ffh;
        or  newstatus,00c00h;
        fwait;
        fldcw newstatus;
        fldl2e;
        fmul qword ptr x;
        fld st;
        frndint;
        fsubr st,st(1);
        fld st
        f2xm1;
        fld1;
        fadd
        fxch st(1)
        fincstp
        fscale;
        fxch
        fchs
        fdecstp
        fchs
        f2xm1;
        fld1;
        fadd
        fscale;
        fxch
        ffree st
        fincstp
        fld  st
        fxch st(2)
        fadd st(1),st
        fsub st,st(2)
        fdivr
        fstp qword ptr x;
        ffree st
         fldcwoldstatus;
        }
    return x;
```

```
   }

double dhTan(double x){
   asm{
      fld1
      fld st
      fld qword ptr x
      fadd st(1),st
      fsubr st,st(2)
      fmul
      fstp qword ptr x
      ffree st
      }
   return x;
   }

double dhSig(double x){
   asm{
      fld1
      fld qword ptr x
      fsub  st(1),st
      fmul
      fstp qword ptr x
      ffree st
      }
   return x;
   }

float   hTans(float x){
      float ep,em;
      float scale=0.00003;
      ep=exp(x*scale);
      em=exp(-x*scale);
      x=((ep-em))/(ep+em);
      return x;
      }

double ihTan(double x){
   double unscale=16666.667;  // unscale=0.5/0.00003
   if (x==1) return 100000;
   asm{
   fld1
   fld st
   fld qword ptr x
   fadd st(1),st
   fsubr st,st(2)
   fdiv
   fldln2
   fxch st(1)
   fyl2x
   fmul qword ptr unscale
```

```
    fstp qword ptr x
    ffree st
    }
return x;
} //end sig.c
/**********************************************************************/

/* scale.c
* scales the input into the range -1 < scale(x) < 1
* unscale reverses the scaling
* The amount of scaling is controled by the factor SCALE_FACTOR
*/
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <stdlib.h>

#define SCALE_FACTOR 20.0
#define LINEAR_SCALE_FACTOR  0.0001
#define LINEAR_UNSCALE_FACTOR 10000
#define SHIFT_FACTOR 10000

extern float *dataInMax, *dataInMin, *dataOutMax, *dataOutMin, *scaleIn, *scaleOut;
double linearScale(double x){
   double scale = LINEAR_SCALE_FACTOR;
   asm{
      fld    scale
      fmul  x
      fstp   x
      }
   return x;
   }

double linearUnscale(double x){
   double scale = LINEAR_UNSCALE_FACTOR;
   asm{
      fld    scale
      fmul  x
      fstp   x
      }
   return x;
   }

double scale(double x){
   double sx,shift;
   sx=1/SCALE_FACTOR;
   if(x==0) return -1.0;
   asm fld sx
   asm fld x
   asm fyl2x
   asm fstp sx
```

```
    return sx;
    }

double unScale(double x){
    double sx=SCALE_FACTOR,shift=SHIFT_FACTOR;
    unsigned int cw1,cw;
     if(x==0) return 1;
    if(x==-1.0) return ;
    asm fstcw cw
    asm fstcw cw1
    asm and cw,0xf3ff
    asm or  cw,0x0400
    asm fldcw cw

    asm fld x
    asm fld sx
    asm fmul
    asm fld st(0)
    asm fld st(0)
    asm frndint
    asm fldcw cw1
    asm fsub
    asm f2xm1
    asm fld1
    asm fadd
    asm fscale
    asm fstp sx
    asm ffree st(0)
    sx=(x >= 0)? sx : sx*0.5;
    return sx;
    }

int scale2(int rows,int inputs,int outputs,float **dataIn,float **dataOut){
    int i,j;
    float spread,shift;
    for(j=0;j<inputs;j++){
        spread=dataInMax[j]-dataInMin[j];
        if( spread==0)
            for(i=0;i<rows;i++) dataIn[i][j]=0;
        else{
            shift= -3 - (6/spread)*dataInMin[j];
            for(i=0;i<rows;i++) dataIn[i][j] = shift + 6.0/spread * dataIn[i][j];
            }
        }
    for(j=0;j<outputs;j++){
        spread=dataOutMax[j]-dataOutMin[j];
        if( spread==0)
            for(i=0;i<rows;i++) dataOut[i][j]=0;
        else{
            shift= -1 - (2/spread)*dataOutMin[j];
            for(i=0;i<rows;i++) dataOut[i][j]= shift + 2.0/spread * dataOut[i][j];
```

```
        }
    }
    return 0;
}//end scale

int scale1(int inputs, float *now){
    int j;
    float spread,shift;
    for(j=0;j<inputs;j++){
    spread=dataInMax[j]-dataInMin[j];
    if( spread==0)
        now[j]=0;
    else{
        shift= -3 - (6/spread)*dataInMin[j];
        now[j] = shift + 6.0/spread * now[j];
        }
    }
return 0;
} /*end scale1*/

float scale0(float x,int j){
    float spread,shift;
    spread=dataInMax[j]-dataInMin[j];
    if( spread==0)   x=0;
    else{
        shift= -3 - (6/spread)*dataInMin[j];
        x = shift + 6.0/spread * x;
        }
    return x;
} //end scale.c
/******************************************************************************/// nett3.c for training
network without participating in market
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=10,b1=5;

float cost(float q){
    return b1 * q ;
```

```
    }

float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q =  1.0*(p-b1);
    return c[id].qe+q;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16];
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;

    signal(SIGFPE,Catcher);
    if(argc > 0)
        id=atoi(argv[1]);
        iterations=atoi(argv[2]);
        iterations=(iterations>1)? iterations : 100;
    lpProfit=makeFile("p00",argv[1],".xlt");
    strcpy(weightFile,makeFileName("w00",argv[1],".txt"));
    fclose(lpProfit);
    // note that d->cprice is actually total Qe
    lpData=makeFile("dat00",argv[1],".xlt");
    rewind(lpData);
    inputs=7;
    outputs=1;
    hidden=5;
    rows = countRows(lpData)-1;
    if(rows >0)
        trainNet(inputs,hidden,outputs,iterations,rows,lpData, weightFile);
    free2(w12);
    free2(w23);
    free2(dataIn);
    free2(dataOut);
    free(sum);
    free(sumOut);
    free(dOut);
    free(dh);
```

```
free(who);
lpProfitData=makeFile("datp0",argv[1],".xlt");
rewind(lpProfitData);
inputs=5;
outputs=1;
hidden=3;
rows = countRows(lpProfitData);
if( rows >0){
allocateMemory(hidden, outputs,inputs);
readData(lpProfitData,rows,inputs,outputs); //reads data into dataIn and dataOut
w12=dim2(inputs+1,hidden,sizeof(float));
w23=dim2(hidden+1,outputs,sizeof(float));
strcpy(weightFile,makeFileName("wp0",argv[1],".txt"));
getWeights(weightFile,hidden, inputs, outputs, w12, w23);
for(j=0;j<inputs;j++){
    dataInMin[j]=(dataInMin[j] <= dataIn[rows-1][j])? dataInMin[j] : dataIn[rows-1][j];
    dataInMax[j]=(dataInMax[j] >= dataIn[rows-1][j])? dataInMax[j] : dataIn[rows-1][j];
    }
for(j=0;j<outputs;j++){
    dataOutMin[j]=(dataOutMin[j] < dataOut[rows-1][j])? dataOutMin[j] : dataOut[rows-1][j];
    dataOutMax[j]=(dataOutMax[j] > dataOut[rows-1][j])? dataOutMax[j] : dataOut[rows-
1][j];
    }
scale2(rows,inputs,outputs,dataIn,dataOut);
who= (float*) calloc(rows+1,sizeof(int));
if(!who){
    fprintf(stderr,"error allocating memory for hidden layer\n");
    exit(1);
    }
for(i=0;i<rows;i++) who[i]=i;
train(iterations, inputs, outputs, rows, hidden, w12, w23, dataIn,hTan,dhTan);
if(( lpOut = fopen("netp.xlt", "a+t")) == NULL){
    fprintf(stderr, "Can't open netp file \n");
    exit(1);
    }
for(i=0;i<rows;i++){  /* for each row of data */
    forwardPass(hidden,outputs, inputs, dataIn[i], w12, w23, hTan); //compute net output
    restoreScale(lpOut, outputs, i, ihTan);
    }/*** end of i loop **/
fclose(lpOut);
saveWeights(lpWeights,inputs,outputs,hidden, w12, w23);
free2(w12);
free2(w23);
free2(dataIn);
free2(dataOut);
free(sum);
free(sumOut);
free(dOut);
free(dh);
free(who);
} // if time ==1
```

```
lpData=makeFile("dat$0",argv[1],".xlt");
rewind(lpData);

    inputs=6;
    outputs=1;
    hidden=4;
    rows = countRows(lpData);
    if(rows >0)
        strcpy(weightFile,makeFileName("w$0",argv[1],".txt"));
    trainNet(inputs,hidden,outputs,iterations,rows,lpData, weightFile);
    free2(w12);
    free2(w23);
    free2(dataIn);
    free2(dataOut);
    free(sum);
    free(sumOut);
    free(dOut);
    free(dh);
    free(who);
    } // if rows
    return 0;
} // end nett3.c
/****************************************************************************/
// pn1.c #include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=50,b1=5;

float cost(float q){
    return b0 + b1 * q ;
    }

float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
    }

float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
```

```
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit,periodProfit1=0, roundProfit,totalQuantity;
    char fname[12],fid[4],*weightFile[16];
    long pos;
    float now[6],now1[6];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;

    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);

    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth <= 0) {
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
        a[id].active=0;
        return 0;
        }
    /*** noise values are constant throughout the simulation **/
    b0 += a[id].coef[1] ;
    b1 += a[id].coef[0] ;
    lpProfit=makeFile("p00",argv[1],".xlt");
    if(b->time==1 && b->period==1 && b->round==1); //initialization
    else {
        fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
        fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
        fseek(lpProfit,SEEK_CUR,SEEK_END);
        if(b->time==1){
            periodProfit1=periodProfit;
        }
        if(b->time==2) {
```

```
            newSales=sold=periodProfit=0;
        }
        newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
        sold += newSales;
        profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p -
unitCost());
        periodProfit += profit;
        roundProfit += profit;
        }
    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
        ((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit
,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
    fclose(lpTemp);
    // note that d->cprice is actually total Qe
    qxe= d->cprice - c[id].qe;
    qxa=d->cprice -  d->periodq1 - c[id].q;
    qxa=(qxa>=0)? qxa : 0;
    if (b->time == 1 && b->period ==1 && b->round==1);
    else {
        if(b->time == 1 ){
            lpData=makeFile("dat00",argv[1],".xlt");
        if(b->round > b->maxr) {
            fprintf(lpData,"%f\n",qxe);
        fclose(lpData);
        return 0;
        }
        if (b->time == 1 && b->period ==2 && b->round==1)
            fprintf(lpData,"%d\t%d\t%d\t%f\t%f\t%f\t%f\t",
                b->producers,b->consumers,b->maxt,c[id].qe,c[id].q,qxe,qxa);
        else
            fprintf(lpData,"%f\n%d\t%d\t%d\t%f\t%f\t%f\t%f\t",
                qxe,b->producers,b->consumers,b->maxt,c[id].qe,c[id].q,qxe,qxa);
        rewind(lpData);
        inputs=7;
        outputs=1;
        hidden=5;
        iterations=100;
        rows = countRows(lpData)-1;
        if(rows >0){
            strcpy(weightFile,makeFileName("w00",argv[1],".txt"));
            allocateMemory(hidden, outputs,inputs);
            w12=dim2(inputs+1,hidden,sizeof(float));
            w23=dim2(hidden+1,outputs,sizeof(float));
            readData(lpData,rows,inputs,outputs); //reads data
            getWeights(weightFile,hidden, inputs, outputs, w12, w23);
            for(j=0;j<inputs;j++){
                dataInMin[j]=(dataInMin[j] <= dataIn[rows-1][j])? dataInMin[j] : dataIn[rows-1][j];
```

```
            dataInMax[j]=(dataInMax[j] >= dataIn[rows-1][j])? dataInMax[j] : dataIn[rows-1][j];
            }
        for(j=0;j<outputs;j++){
            dataOutMin[j]=(dataOutMin[j] < dataOut[rows-1][j])? dataOutMin[j] :
dataOut[rows-1][j];
            dataOutMax[j]=(dataOutMax[j] > dataOut[rows-1][j])? dataOutMax[j] :
dataOut[rows-1][j];
            }
    // Scale the data into 0 < x <= 1 range
        for(i=0;i<rows;i++){
            for(j=0;j<inputs;j++)  dataIn[i][j]= scale(dataIn[i][j]);
            for(j=0;j<outputs;j++) dataOut[i][j]= scale(dataOut[i][j]);
            }
        who= (float*) calloc(rows+1,sizeof(int));
        if(!who){
            fprintf(stderr,"error allocating memory for hidden layer\n");
            exit(1);
            }
        for(i=0;i<rows;i++) who[i]=i;
            train(iterations, inputs,  outputs, rows, hidden, w12, w23, dataIn, sigmoid, dhSig );
            saveWeights(lpWeights,inputs,outputs,hidden, w12, w23);
            } // end of rows > 0
        if(b->time==1 && rows>0){
            now[0]=scale(b->producers);
            now[1]=scale(b->consumers);
            now[2]=scale(b->maxt);
            now[3]=scale(c[id].qe);
            now[4]=scale(c[id].q);
            now[5]=scale(qxe);
            now[6]=scale(qxa);
            forwardPass(hidden,outputs, inputs, now, w12, w23,sigmoid); //compute net output
            lpNetProfit=makeFile("q00",argv[1],".xlt");
            fprintf(lpNetProfit,"%f\t%f\n",unScale(now[5]),unScale(sumOut[0]));
            fclose(lpNetProfit);
            qxe=unScale(sumOut[0]);  // save the scaled value of projected output by others
            free2(w12);
            free2(w23);
            free2(dataIn);
            free2(dataOut);
            free(sum);
            free(sumOut);
            free(dOut);
            free(dh);
            free(who);
        } // end of if time==1 and rows >0
    if(b->time==1&&b->period==1&&b->round==1){
        printf("error, this should not happen...\n");
        printf("press any key to exit\n");
        getch();
        exit (-1);
        }
```

```
    lpProfitData=makeFile("datp0",argv[1],".xlt");
    fprintf(lpProfitData,"%d\t%d\t%d\t%f\t%f\t%f\n",
        b->producers,b->consumers,b->maxt,c[id].qe,qxe,periodProfit1);
    rewind(lpProfitData);
    inputs=5;
    outputs=1;
    hidden=3;
    iterations=100;
    rows = countRows(lpProfitData);
    if( rows >0){
        allocateMemory(hidden, outputs,inputs);
    readData(lpProfitData,rows,inputs,outputs); //reads data into dataIn and dataOut
    w12=dim2(inputs+1,hidden,sizeof(float));
    w23=dim2(hidden+1,outputs,sizeof(float));
    strcpy(weightFile,makeFileName("wp0",argv[1],".txt"));
    getWeights(weightFile,hidden, inputs, outputs, w12, w23);
    for(j=0;j<inputs;j++){
        dataInMin[j]=(dataInMin[j] <= dataIn[rows-1][j])? dataInMin[j] : dataIn[rows-1][j];
        dataInMax[j]=(dataInMax[j] >= dataIn[rows-1][j])? dataInMax[j] : dataIn[rows-1][j];
        }
    for(j=0;j<outputs;j++){
        dataOutMin[j]=(dataOutMin[j] < dataOut[rows-1][j])? dataOutMin[j] : dataOut[rows-1][j];
        dataOutMax[j]=(dataOutMax[j] > dataOut[rows-1][j])? dataOutMax[j] : dataOut[rows-
1][j];
        }
    scale2(rows,inputs,outputs,dataIn,dataOut);
    who= (float*) calloc(rows+1,sizeof(int));
    if(!who){
        fprintf(stderr,"error allocating memory for hidden layer\n");
        exit(1);
        }
    for(i=0;i<rows;i++) who[i]=i;
        train(iterations, inputs, outputs, rows, hidden, w12, w23, dataIn,hTan,dhTan);
        /* this compares actual and estimated output
        if(( lpOut = fopen("netp.xlt", "a+t")) == NULL){
            fprintf(stderr, "Can't open netp file \n");
            exit(1);
            }
        for(i=0;i<rows;i++){   /* for each row of data */
            forwardPass(hidden,outputs, inputs, dataIn[i], w12, w23, hTan); //compute net
output
            restoreScale(lpOut, outputs, i, ihTan);
            }/*** end of i loop  **/
        fclose(lpOut);
        saveWeights(lpWeights,inputs,outputs,hidden, w12, w23);
    } // end of else ie rows > 0
    if(b->time==1 && rows>0){
        now[0]=b->producers;
        now[1]=b->consumers;
        now[2]=b->maxt;
        now[3]=c[id].qe;
```

```
now[4]=qxe ; // qxe is the projected quantity produced by others in the next period
  scale1(inputs,now);
forwardPass(hidden,outputs, inputs, now, w12, w23, hTan); //compute net output
lpNetProfit=makeFile("p10",argv[1],".xlt");
spread= dataOutMax[0]-dataOutMin[0];
maxProfit=(sumOut[0]+1)* spread/2+ dataOutMin[0];
optimumQ=c[id].qe;
for (i=0;i<100;i++){
    tempQ=c[id].qe - 50+i;
    if(tempQ>0){ //insist on positive quantities
        now[3]=(float) scale0(tempQ,3);
        forwardPass(hidden,outputs, inputs, now, w12, w23, hTan); //compute net output
        spread= dataOutMax[0]-dataOutMin[0];
        tempProfit=(sumOut[0]+1)* spread/2+ dataOutMin[0];
        if(tempProfit >= maxProfit){
            maxProfit=tempProfit;
            optimumQ=tempQ;
        }
    }// if tempQ>0
} // for i<200
fprintf(lpNetProfit,"%f\t%f\n",maxProfit,optimumQ);
fclose(lpNetProfit);
free2(w12);
free2(w23);
free2(dataIn);
free2(dataOut);
free(sum);
free(sumOut);
free(dOut);
free(dh);
free(who);
} // if time ==1
} // if time == 1
} // end else
if(b->time != 1){
    c[id].p= (d->periodp+c[id].pe)/2; //test 11-23-92
    c[id].p = (c[id].p > b1)? c[id].p : b1; //price at marginal cost or above
    }//time !=1
if(b->time==1) {
    c[id].pe=expectedPrice(c[id].p,c[id].pe);
    q=optimumQ; // for feeding in expecd quantity
    if(b->time== b->period== b->round == 1) q=supply(c[id].qe,id);
    if(cost(q) > a[id].wealth) printf("*");
    c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )?
        q : (a[id].wealth >0)? a[id].wealth/unitCost() : 0;
    a[id].wealth -=  cost(c[id].q);
    c[id].qe=c[id].q;
    }//time==1
return 0;
} //end pn1.c
/*******************************************************************************///
```

```
//pn2.c 7/21/92
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <dos.h>
#include <dir.h>
#include <signal.h>
#include <float.h>
#include <string.h>
#include "bb.h"
#include "nett.c"
#include "util.c"

#define RATE 0.5

float w=0,b0=50,b1=5;

float cost(float q){
    return b0 + b1 * q ;
    }
float unitCost(void){
    return(b0+b1*c[id].qe)/c[id].qe;
        }
    float expectedPrice(float p, float pe){
    float newPrice;
    newPrice= RATE*p +(1-RATE)* pe;
    return newPrice;
    }

float supply(float p, int id){
    float q;
    q=20+a[id].coef[1] + (0.5+a[id].coef[0]) * p ;
    return q ;
    }

int main(int argc,char *argv[]){
    int i,j, adjust;
    float timeCriteria, qCriteria, q, sold=0, newSales=0;
    unsigned far *mpointer;
    FILE *lpA, *lpProfit, *lpData, *lpNetProfit, *fp, *lpProfitData, *lpTemp;
    float a1, a2, a3, profit, periodProfit=0,periodProfit1=0, roundProfit=0,totalQuantity=0;
    char fname[12],fid[4],*weightFile[16];
    long pos;
    float now[8],now1[8];
    int inputs,outputs,hidden,rows,iterations;
    float qxe,qxa,maxProfit, optimumQ, tempProfit, tempQ,spread,shift;
    mpointer = MK_FP(0x0000,0x01e0);
    signal(SIGFPE,Catcher);
    a1=a2=a3=0;
    a=MK_FP(mpointer[0], mpointer[1]);
```

```
    b=MK_FP(mpointer[2], mpointer[3]);
    c=MK_FP(mpointer[4], mpointer[5]);
    d=MK_FP(mpointer[6], mpointer[7]);
    lpTrades=MK_FP(mpointer[8], mpointer[9]);
    if(argc > 0)
        id=atoi(argv[1]);
    if( a[id].wealth <= 0) {
        c[id].pe=c[id].p=c[id].qe=c[id].q=0;
        b->producers--;
        a[id].active=0;
        return 0;
        }
/*** noise values are constant throughout the simulation **/
    b0 += a[id].coef[1] ;
    b1 += a[id].coef[0] ;
    lpProfit=makeFile("p00",argv[1],".xlt");
    if(b->time==1 && b->period==1 && b->round==1); //initialization
    else {
        fseek(lpProfit,ftell(lpProfit)-(10+10+10+10+4),SEEK_END);//
        fscanf(lpProfit,"%f %f %f %f",&sold,&profit,&periodProfit,&roundProfit);
        fseek(lpProfit,SEEK_CUR,SEEK_END);
        if(b->time==1){
            periodProfit1=periodProfit;
        }
    if(b->time==2) {
        newSales=sold=periodProfit=0;
        }
    newSales = (c[id].qe == c[id].q )? 0 : c[id].qe-c[id].q-sold;
    sold += newSales;
    profit= ( fabs(newSales * (c[id].p - unitCost())) < 0.01 )?  0 :  newSales * (c[id].p - unitCost());
    periodProfit += profit;
    roundProfit += profit;
    }
    fprintf(lpProfit,"%d\t%#6.4f\t%#6.4f\t%#6.4f\t%#6.4f\t%#8.4f\t%#10.4f\t%#10.2f\t%#10.2f
\n",
    ((b->time ==1)? 5 : b->time-1), c[id].pe, c[id].p, c[id].qe, c[id].q, sold, profit
,periodProfit,roundProfit);
    fclose(lpProfit);
    lpTemp=makeFile("wealth",argv[1],".xlt");
    fprintf(lpTemp,"%d\t%d\t%d\t%d\t%f\n",b->round,b->period,b->time, id, a[id].wealth);
    fclose(lpTemp);
  // note that d->cprice is actually total Qe
    qxe= d->cprice - c[id].qe;
    qxa=d->cprice -  d->periodq1 - c[id].q;
    qxa=(qxa>=0)? qxa : 0;
    if (b->time == 1 && b->period ==1 && b->round==1);  //skip opening
    else {
        if(b->time == 1 ){
            lpData=makeFile("dat00",argv[1],".xlt");
            if(b->round > b->maxr) {
            fprintf(lpData,"%f\n",qxe);
```

```
        fclose(lpData);
        return 0;
        }
if (b->time == 1 && b->period ==2 && b->round==1)
    fprintf(lpData,"%d\t%d\t%d\t%f\t%f\t%f\t%f\t",
    b->producers,b->consumers,b->maxt,c[id].qe,c[id].q,qxe,qxa);
else
    fprintf(lpData,"%f\n%d\t%d\t%d\t%f\t%f\t%f\t%f\t",
        qxe,b->producers,b->consumers,b->maxt,c[id].qe,c[id].q,qxe,qxa);
rewind(lpData);
inputs=7;
outputs=1;
hidden=5;
iterations=100;
rows = countRows(lpData)-1;
if(rows >0){
    strcpy(weightFile,makeFileName("w00",argv[1],".txt"));
    trainNet(inputs,hidden,outputs,iterations,rows,lpData, weightFile);
    }
if(b->time==1 && rows>0){
    now[0]=scale(b->producers);
    now[1]=scale(b->consumers);
    now[2]=scale(b->maxt);
    now[3]=scale(c[id].qe);
    now[4]=scale(c[id].q);
    now[5]=scale(qxe);
    now[6]=scale(qxa);
    forwardPass(hidden,outputs, inputs, now, w12, w23,sigmoid); //compute net output
    qxe=unScale(sumOut[0]); // save the scaled value of projected output by others
    free2(w12);
    free2(w23);
    free2(dataIn);
    free2(dataOut);
    free(sum);
    free(sumOut);
    free(dOut);
    free(dh);
    free(who);
} // end of if time==1 and rows >0
if(b->time==1&&b->period==1&&b->round==1){
    printf("error, this should not happen...\n");
    printf("press any key to exit\n");
    getch();
    exit (-1);
    }
if(b->time==1){
    lpProfitData=makeFile("datp0",argv[1],".xlt");
    fprintf(lpProfitData,"%d\t%d\t%d\t%f\t%f\t%f\n",
        b->producers,b->consumers,b->maxt,c[id].qe,qxe,periodProfit1);
    rewind(lpProfitData);
inputs=5;
```

```
outputs=1;
hidden=3;
iterations=100;
rows = countRows(lpProfitData);
if( rows >0){
    allocateMemory(hidden, outputs,inputs);
    readData(lpProfitData,rows,inputs,outputs); //reads data into dataIn and dataOut
    w12=dim2(inputs+1,hidden,sizeof(float));
    w23=dim2(hidden+1,outputs,sizeof(float));
    strcpy(weightFile,makeFileName("wp0",argv[1],".txt"));
    getWeights(weightFile,hidden, inputs, outputs, w12, w23);
    for(j=0;j<inputs;j++){
        dataInMin[j]=(dataInMin[j] <= dataIn[rows-1][j])? dataInMin[j] : dataIn[rows-1][j];
        dataInMax[j]=(dataInMax[j] >= dataIn[rows-1][j])? dataInMax[j] : dataIn[rows-1][j];
        }
    for(j=0;j<outputs;j++){
        dataOutMin[j]=(dataOutMin[j] < dataOut[rows-1][j])? dataOutMin[j] : dataOut[rows-
1][j];
        dataOutMax[j]=(dataOutMax[j] > dataOut[rows-1][j])? dataOutMax[j] : dataOut[rows-
1][j];
        }
scale2(rows,inputs,outputs,dataIn,dataOut);
who= (float*) calloc(rows+1,sizeof(int));
if(!who){
    fprintf(stderr,"error allocating memory for hidden layer\n");
    exit(1);
    }
for(i=0;i<rows;i++) who[i]=i;
    train(iterations, inputs, outputs, rows, hidden, w12, w23, dataIn,hTan,dhTan);
saveWeights(lpWeights,inputs,outputs,hidden, w12, w23);
} // end of else ie rows > 0
}
if(b->time==1 && rows>0){
    now[0]=b->producers;
    now[1]=b->consumers;
    now[2]=b->maxt;
    now[3]=c[id].qe;
    now[4]=qxe ; // qxe is the projected quantity produced by others in the next period
    scale1(inputs,now);
    forwardPass(hidden,outputs, inputs, now, w12, w23, hTan); //compute net output
    lpNetProfit=makeFile("p10",argv[1],".xlt");
    spread= dataOutMax[0]-dataOutMin[0];
    maxProfit=(sumOut[0]+1)* spread/2+ dataOutMin[0];
    optimumQ=c[id].qe;
    for (i=0;i<100;i++){
        tempQ=c[id].qe - 50+i;
        if(tempQ>0){ //insist on positive quantities
            now[3]=(float) scale0(tempQ,3);
        forwardPass(hidden,outputs, inputs, now, w12, w23, hTan); //compute net output
        spread= dataOutMax[0]-dataOutMin[0];
        tempProfit=(sumOut[0]+1)* spread/2+ dataOutMin[0];
```

```
            if(tempProfit >= maxProfit){
                maxProfit=tempProfit;
                optimumQ=tempQ;
                }
        }// if tempQ>0
    } // for i<200
    fprintf(lpNetProfit,"%f\t%f\n",maxProfit,optimumQ);
    fclose(lpNetProfit);
    free2(w12);
    free2(w23);
    free2(dataIn);
    free2(dataOut);
    free(sum);
    free(sumOut);
    free(dOut);
    free(dh);
    free(who);
} // if time ==1
}  // if time == 1
} // end else
if(b->time != 1){
    c[id].p= (d->periodp+c[id].pe)/2; //test 11/23/92
    c[id].p = (c[id].p > b1)? c[id].p : b1; //using marginal cost
    }//time !=1
if(b->time==1) {
if( ! (b->period == 1 && b->round == 1) ) {
// note that d->cprice is actually total Qe
    lpData=makeFile("dat$0",argv[1],".xlt");
    fprintf(lpData,"%d\t%d\t%d\t%d\t%d\t%f\t%f\n",
        b->producers,b->consumers,b->maxt,b->period,b->round,d->cprice,d->periodp1);
    rewind(lpData);
    inputs=6;
    outputs=1;
    hidden=4;
    iterations=100;
    rows = countRows(lpData);
    if(rows >0)
        strcpy(weightFile,makeFileName("w$0",argv[1],".txt"));
        trainNet(inputs,hidden,outputs,iterations,rows,lpData, weightFile);
        if(b->time==1 && rows>0){
            now[0]=scale(b->producers);
            now[1]=scale(b->consumers);
            now[2]=scale(b->maxt);
            now[3]=scale(b->period);
            now[4]=scale(b->round);
            now[5]=scale(d->cprice);
            forwardPass(hidden,outputs, inputs, now, w12, w23,sigmoid); //compute net output
            lpNetProfit=makeFile("$00",argv[1],".xlt");
            fprintf(lpNetProfit,"%f\t%f\n",d->periodp1, unScale(sumOut[0]) );
            fclose(lpNetProfit);
        }
```

```
c[id].pe=unScale(sumOut[0]);
free2(w12);
free2(w23);
free2(dataIn);
free2(dataOut);
free(sum);
free(sumOut);
free(dOut);
free(dh);
free(who);
} // if rows
q=optimumQ;  // for feeding in expecd quantity
if(b->time== b->period== b->round == 1) q=supply(c[id].pe,id);
if(cost(q) > a[id].wealth) printf("*");
c[id].q=(cost(q) <= a[id].wealth && a[id].wealth >= 0 )? q : (a[id].wealth >0)?
a[id].wealth/cost(q) : 0;
a[id].wealth -=  cost(c[id].q);
c[id].qe=c[id].q;
}//time==1
return 0;
} //end pn2.c
```