I. Introduction

The model constructed for this project involved a set of interacting agents separated into *buyers* and *sellers* of some hypothetical product or stock. The model allowed these agents, who were provided with little information about the overall preferences of other agents, attempt to maximize their profit. From the time of Adam Smith, it has been proposed that such a set of agents, all acting to maximize their own gains, actually results in profit maximization for the entire market, or a high market efficiency. The model seeks to create a similar situation with greedy agents that can "learn" to an extent from previous transaction markets.

The set of inputs to the code relate to the number of buyers and sellers, the values of the hypothetical good to these buyers and sellers, and the market run time. The function takes a structure *input*. The input designates the number of buyers and sellers (total system has *2*N* participants), the parameters for creating a vector of values representing the "personalities" for agents or the personality vectors themselves, and parameters denoting how long the market will last and how many iterations of trading will occur in a given run.

The outputs are the agent personality vectors, the record of all transactions that occurred, including transaction price, the agents involved, and the "time" at which it took place; and a set of parameters that provide some basic information about the trading that occurred, including market efficiency (the ratio of actual profit to maximum profit as determined by economic theory), the market price of the good (again determined by economic theory), and the optimal number of transactions (again from economic theory). To obtain the economically most efficient number of transactions for the market, one finds the transaction price for which there are an equal number of buyers and sellers willing to make a transaction, based on their absolute value of the product. The transaction price at this point, what has been called the *market price* for the good is the optimal economic price for the good. The maximum economic profit is calculated by finding the difference in what each buyer and seller who would benefit from making a transaction at the market price actually values the product at and the market price for the product.

II. Methods

The initial intentions for the model was to have two distinct cycles. One was the cycle of a particular market for which each buyer and seller was allowed to make a single transaction. This market followed a set of four steps (plus a zero-ith step) for a predetermined maximum number of iterations (*ceil(T/del_T*). Stage 5 and 6 outline the "big picture" code in which agents "learn". This set-up was then intended to be looped over some number of times in an attempt to allow the agents to "learn" about the system they were interacting in based on previous markets' experiences. At the time of this writing, however, only the first of these has been completed.

The four stages for the inner market loop are as follows:

Stage 0: Set-up "personalities" for buyers and sellers or obtain from input

Stage 1: Buyers and sellers decide on buy and sell prices for the good

Stage 2: Transactions take place based on the prices arrived at in Stage 1 or Stage 3 from the second iteration on

Stage 3: Buyers and sellers left in the market re-evaluate their offering prices based on basic information about the market

Stage 4: Repeat from Stage 2 until time “T” or no more transactions are possible

Stage 5: Update initial offering prices based on past market experience

Stage 6: Repeat Stages 0 - 5 for some set number of times

In Stage 0, agents assign the product a personal worth or get the parameter from the function input if the market is being repeated. Typically, the average value for buyers is made higher than the average value for sellers, though the best conditions are when there is significant overlap between the two. The case of all buyers valuing the product more than all sellers value the product results in a trivial outcome of every individual carrying out a transaction, an ill-defined market price, and maximum profit for the system. Additionally, agents are assigned a “patience”, which dictates how intent they are on obtaining a large profit as opposed to simply obtaining any profit. If a second market is started, agents retain these “personality” vectors in subsequent rounds, though a more accurate code might instead adjust agents’ patience based on past transactions.

Stage 1 and Stage 3 are similar, though the price-setting in Stage 1 is hard-coded; buyers try to obtain the good for a value significantly lower than what they value the product at, at sellers try to sell for much more than they value the product. Stage 3 involves a more refined assessment. The patience of each agent comes into play here. If the buyer or seller has not made a transaction and feels that there is still sufficient time for the opposite agent type to “come around”, they leave their offers unchanged. Else, their offers are modified in a fashion that makes them more likely to undergo a transaction. An agent’s offer is adjusted so that the new offering value is closer to the average offering prices submitted by the opposite agent type by a degree proportional to the difference between this values and the agent’s last offering value. An agent never makes an offering price that would result in a negative profit; in other words, it is better to keep the product than make an unfavorable transaction.

In Stage 2, transactions are carried out. All agents who are willing to make transactions in a round, as determined by the set of offering prices from remaining agents, are placed in a random order representing the order in which bids were placed in the market. Transactions are then attempted based on agents left in the particular transaction period. The next agent in cue’s offer gets preference over others. The first agent of the opposite type later in the cue with offers higher (or lower, if the agent is a buyer) is selected as the transaction partner, and the transaction takes place to the benefit of the agent cued up. If no such matches are left in the cue, the agent is removed from the particular round in question. A round is over when no possible matches are left in the cue.

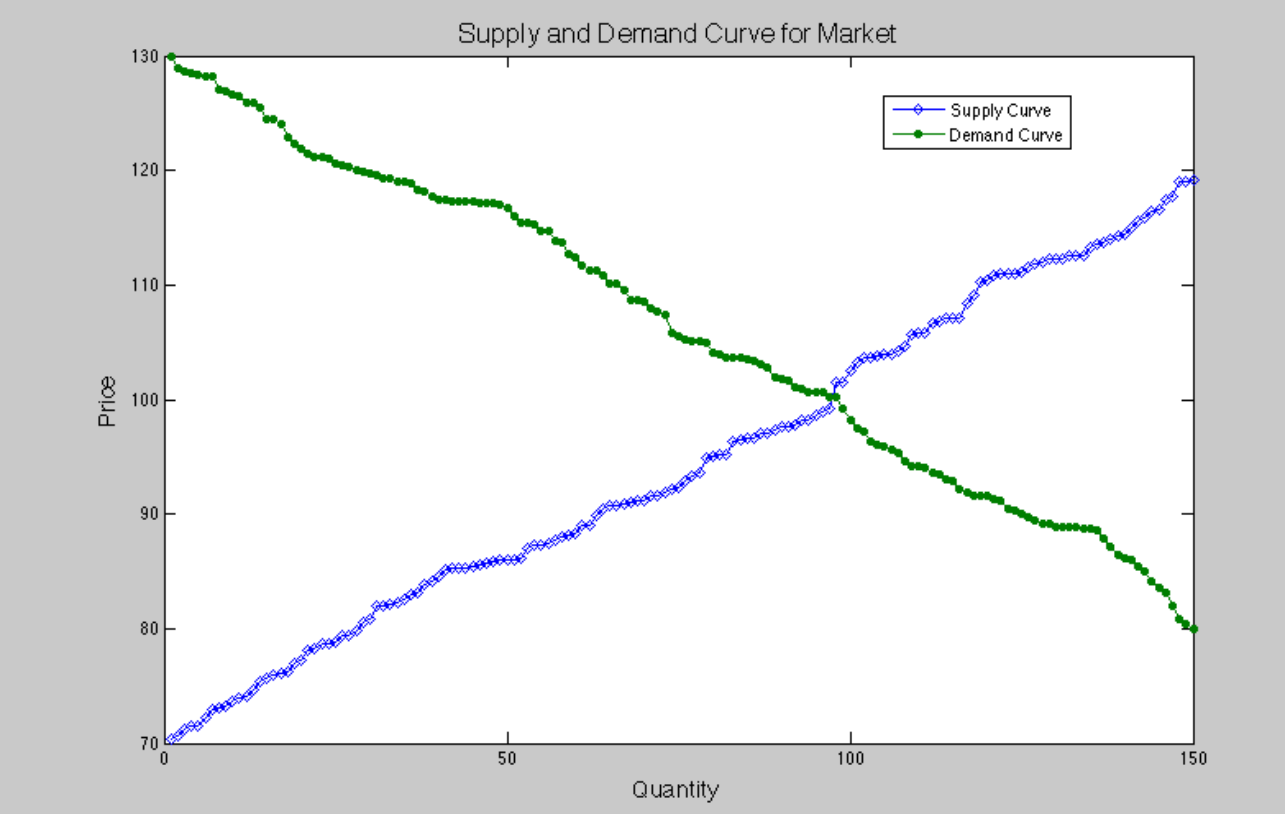
The subsequent repeating of Stage 2 - Stage 4 and expiring of the pre-set “time limit” or until no possible matches are left is one market. After a market is complete, the agents modify their starting bids based on the outcomes of past market(s) and their “personalities” (Stage 5). A new market is then started, and so on (Stage 6).

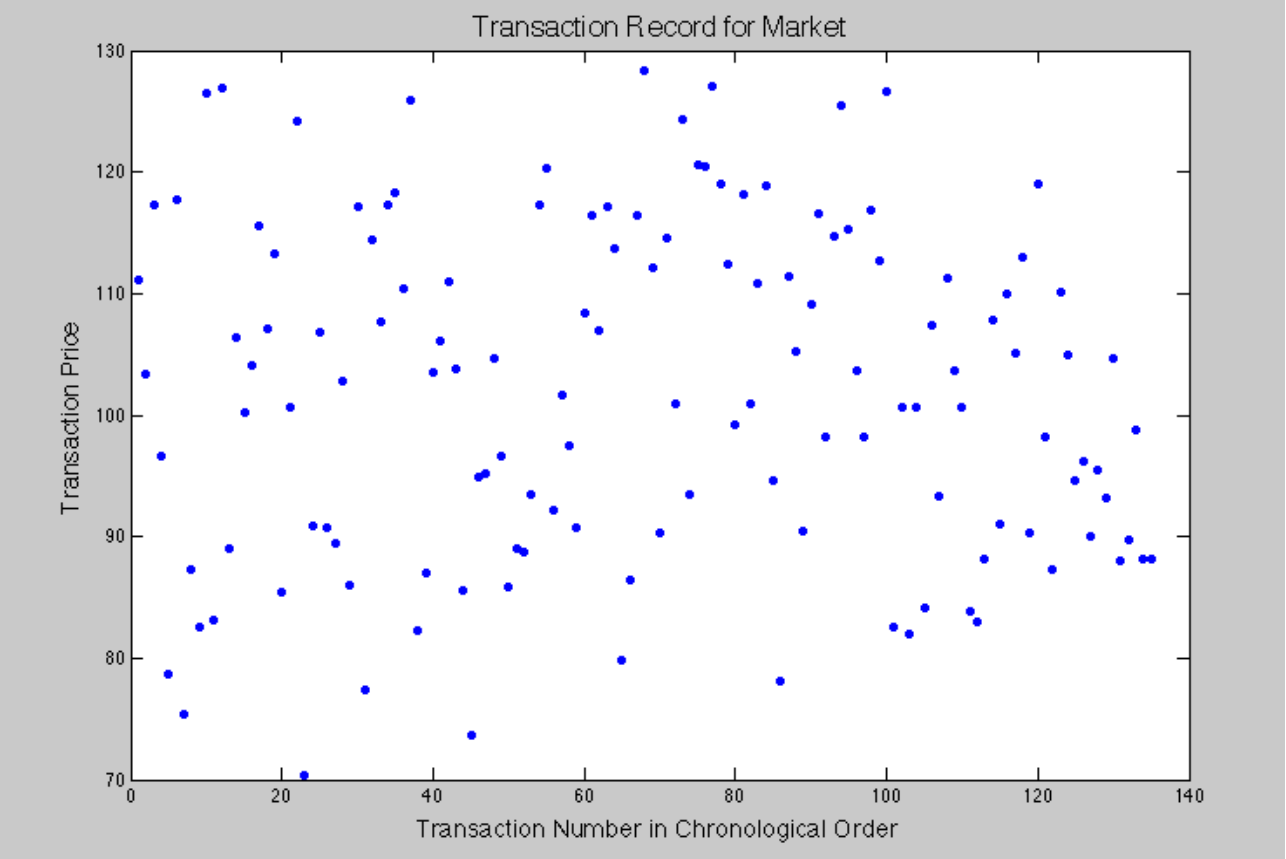
III. Analysis

While a single market with inputs operated smoothly, the repetition of markets in an attempt to have agents “learn” and have a more efficient market arise did not occur. The two general plots from the code are shown below. The first consists of the supply and demand curve for the system, given how agents value the product (demand is downward sloping, and supply upward). The second is the plot of transactions that occurred in the order they occurred and their respective transaction prices. The total number of agents in

the system was 300 (N=150). From the supply / demand curve plot, it can be seen that the optimum number of transactions is about 100 and the market price of the good is about $100 (random coincidence). For the particular transaction plot shown, the average going rate for the good was around $100 ($102) and 135 transactions took place. The efficiency of this market was 76%, even though the mean value was close to the market price for the good. This is because the distribution of transaction prices is too varied. A more efficient market would have most transactions taking place in a narrow band around the market price; a perfect market would have all transactions taking place at the market value.

Ideally, no agents to the right of the supply-demand curve intercept should make transaction. For such a case, the profit of the market would be the area between the supply and demand curves to the left of the intercept point.





The most likely reason for an inability of the agents to "learn" the optimal transaction price is the lack of memory about (a) how much profit they've made and (b) overall market trends. This is a result of rules for setting the new starting bid prices that are too simple, as well as constant patience on the part of agents, though to a lesser extent. In the experiment this is based on, individuals were granted information about all transactions and bids that took place and sufficient mental capacities to recall previous bids and the general overall appearance of the market outcome.

IV. Conclusions

Though the core *Market* code performs well at its task, the desired "learning" feature is not realized by the *meta_Market* portion of the code. Such shortcoming is due to inadequacies in the update price algorithm. With additional time, a reliable algorithm could be constructed that would result in an optimizing of market efficiency due to a converging of starting prices to near the market value for the good based only on the information already available to the agents, with the addition of a "memory" of multiple previous experiences.

Additionally, it would be interesting to allow agents multiple transactions in a particular market. Also, allowing agents to switch between buying and selling based on a moving market value and a semi-fixed personal worth / profit maximization attempt would also be interesting to implement. Such code would have a single, continuous market. An algorithm would update the going price based on the number of transactions taking place and how agents valued the stocks.

V. MATLAB Code

```
function output = Market_2(input)
% [market_eff market_price trans_record]

%% Set variables for market based on inputs
T=input.T(1); del_T=input.T(2); %% Time, iteration information
if isempty(input.N),N=length(input.buy_value); %% Number of agents
else,N=input.N;end
%% Setting or getting personality parameters
if isempty(input.buy_value)
    base=input.price_info(1); range=input.price_info(2);
spread=input.price_info(3);

buy_value=ones(1,N).*base+(rand(1,N)-.4*ones(1,N)).*range+.5*spread.*ones(1,N);
%% Assign value of product to buyers

sell_value=ones(1,N).*base+(rand(1,N)-.6*ones(1,N)).*range-.5*spread.*ones(1,N);
%% Assign value of product to sellers
else,buy_value=input.buy_value; sell_value=input.sell_value;end
if isempty(input.buy_patience)
    %% Set buyer and seller patiences, 'normalized' to max market run time
    buy_patience = rand(1,N)*T; sell_patience = rand(1,N)*T;
else, buy_patience=input.buy_patience; sell_patience=input.sell_patience;end

%% This bit find the optimum number of buyers, sellers in the market based
%% on how everyone values the product being exchanged.  If the sorted
%% vectors were plotted, 'keep' would be the quantity of the good at the
%% intersection of the two lines.  The price at this intersection would be
%% the market price for the good.  Since this is a discrete system, the
%% market price is defined as the mean of the buy and sell value for the
%% 'last' buyer and seller.
dumm_sell=sort(sell_value,'ascend'); dumm_buy=sort(buy_value,'descend');
%% Plot of supply and demand curves
% plot(1:N,dumm_sell,1:N,dumm_buy)
%% Determining optimum ## transactions
for ii=N:-1:1
j=length(find(dumm_sell<dumm_buy(ii)));
if j>=ii,keep=j;,break;end
end
%% Calculate maximum profit from optimum number of buyers and sellers
keep=max(length(dumm_sell(dumm_sell<=dumm_buy(keep))),
length(dumm_buy(dumm_buy>=dumm_sell(keep))));
profit_max=sum(dumm_buy(1:keep))-sum(dumm_sell(1:keep));
%% Calculate market price of good based on how people value good
market_price=mean([dumm_buy(keep),dumm_sell(keep)]);


itr=ceil(T/del_T); %% Set number of iterations given T, del_T
buy_indx=1:N; %% Buyers index
sell_indx=1:N; %% Sellers index
buy_price=buy_value*.5; %% Each wants to make a lot of profit...
sell_price=sell_value*2;
profit=0;T=0;trans_record=[];current=[];current_2=[];buy=0;sell=0; %% Initialize
variables
count=0;
%fig=figure;
for i=1:itr
    T=T+del_T; %% Update time step
```

```
    max_buy=max(buy_price(buy_indx)); %% Max buying price
    min_sell=min(sell_price(sell_indx)); %% Min selling price
    buyers=find(buy_price(buy_indx)>=min_sell); buyers=buy_indx(buyers); %% If
not offering more than min, remove
    sellers=find(sell_price(sell_indx)<=max_buy); sellers=sell_indx(sellers); %%
If asking more than max, remove

    temp_indx=[buyers,sellers+N]; %% All making transaction in one indx, maps
onto indx
    temp_indx(randperm(length(temp_indx)))=temp_indx; %% Set bidding order & put
temp_indx in bidding order
    if ~isempty(temp_indx),count=count+1;end %% Records the number of successful
transaction periods
    temp_trans_record=[]; %% Initiating temporary transaction record for updat-
ing market bids

    while length(temp_indx)>1 & max(temp_indx)>N & min(temp_indx)<=N

        current=temp_indx(1); %% Since temp_indx is in the bidding order, take
first entry

        if current<=N %% Buyer condition, buyer advantage
            temp=temp_indx(temp_indx>N); %% Take away buyers in temp_indx
            %% Find first instance in temp_indx where sell price is less
            %% than or equal to what current buyer is willing to pay
            current_2=temp(find(sell_price(temp-N)<=buy_price(current),1))-N;
            if ~isempty(current_2)
                sell=sell+1;
                record=[sell_price(current_2),current,current_2 T]; %% Set
transaction price to whatever seller is willing to sell for
                temp_indx(temp_indx==current_2+N)=[]; %% Remove nodes from
temp_indx so they are not reused
                buyers(find(buyers==current))=[]; buy_indx(find(buy_indx==cur-
rent))=[]; %% Clear buyer from both indexes
                sellers(find(sellers==current_2))=[];
sell_indx(find(sell_indx==current_2))=[]; %% Clear seller from both indexes
                %% Update market profit
                add_profit=abs(buy_value(current)-record(1))+...
                           abs(sell_value(current_2)-record(1));
            end

        else %% Seller condition, seller advantage
            current=current-N; %% Adjust variable value from temp_indx list
            temp=temp_indx(temp_indx<=N); %% Take away other sellers from
temp_indx
            %% Find first instance in temp_indx where buy price is greater
            %% than or equal to what current seller is willing to pay
            current_2=temp(find(buy_price(temp)>=sell_price(current),1));
            if ~isempty(current_2);
                buy=buy+1;
                record=[buy_price(current_2),current_2,current T]; %% Set trans-
action price to whatver buyer is willing to pay
                temp_indx(temp_indx==current_2)=[]; %% Remove nodes from
temp_indx so they are not reused
                buyers(find(buyers==current_2))=[]; buy_indx(find(buy_indx==cur-
rent_2))=[]; %% Clear buyer from both indexes
                sellers(find(sellers==current))=[];
sell_indx(find(sell_indx==current))=[]; %% Clear seller from both indexes
                %% Update market profit
```

```
                add_profit=abs(buy_value(current_2)-record(1))+...
                           abs(sell_value(current)-record(1));
            end
        end
        if ~isempty(current_2) %% Transaction takes place
            temp_trans_record=[temp_trans_record; record];  %% Update transac-
tion price record
            profit=profit+add_profit; %% Update profit
        end
        current=[];current_2=[];temp_indx(1)=[]; %% Clear 'current' variables,
temp_indx ref
    end
    buy_energy=T./buy_patience; sell_energy=T./sell_patience;
    %% To update, or not to update?
    buy_logic=(rand(size(buy_patience))>=exp(-buy_energy));
    sell_logic=(rand(size(sell_patience))>=exp(-sell_energy));
    %% Update buyers, sellers offering prices differently if no
    %% transactions took place in the last bidding session
    buy_price=min([...
                   buy_value',...

(buy_price+buy_logic.*(mean(sell_price(sell_price<9999))-buy_price)...
                    .*abs(rand(1,N)-.12))'... %% This compensates for discrepan-
cies in i'm not
                    %% sure what, but something resulting in a tendency
                    %% for buyers to shift their offering prices to
                    %% what they value the product at faster than
                    %% sellers
                   ]');

    sell_price=max([...
                    sell_value',...

(sell_price-sell_logic.*(sell_price-mean(buy_price(buy_price>0)))...
                    .*rand(1,N))'...
                   ]');
    %% Null / void buyers, sellers that have already make transactions
    buy_price(setxor(1:N,buy_indx))=-9999;
    sell_price(setxor(1:N,sell_indx))=9999;
    %% Update transaction record
    trans_record=[trans_record; temp_trans_record];
    %% End the market run if no more buyers or no more sellers
    if isempty(buy_indx) | isempty(sell_indx) |
~(min(sell_value(sell_indx))<=max(buy_value(buy_indx))),break,end
end
market_eff=profit/profit_max; %% Calculate market efficiency
%% Plot prices of all transactions that occured in the order they occured
 plot(1:length(trans_record),trans_record(:,1), 'LineStyle', 'none',
'Marker','.','MarkerSize',14)
%% Plot supply, demand curves of those agents who did not make
%% transactions
dumm_buy=buy_value(buy_indx); dumm_sell=sell_value(sell_indx);
dumm_buy=sort(dumm_buy,'descend'); dumm_sell=sort(dumm_sell);
% fig=figure;
% plot(1:length(dumm_sell),dumm_sell,1:length(dumm_buy),dumm_buy);

%% Setting output variable
output.stuff=[min(sell_price(sell_indx)) max(buy_price(buy_indx)) keep count buy
sell market_eff];
```

```
output.N=N; output.buy_value=buy_value; output.sell_value=sell_value;
output.buy_patience=buy_patience; output.sell_patience=sell_patience;
output.trans_record=trans_record;


function output = meta_Market(input)

%% input.trans
%% input.buy / sell_value

trans_record=input.trans_record;
avg_trans=mean(trans_record(:,1));
buy_value=input.buy_value; sell_value=input.sell_value;
N=length(sell_value);
buy_trans=zeros(1,N);sell_trans=zeros(1,N);
for ii=1:length(trans_record)
    buy_trans(trans_record(ii,2))=...
        trans_record(ii,1);
    sell_trans(trans_record(ii,3))=...
        trans_record(ii,1);
end

output.buy_price=min(abs([...
                    (avg_trans+(buy_value-avg_trans).*(rand(1,N)-0.7));...
                    (avg_trans+(buy_trans-avg_trans).*(rand(1,N)-0.7));...
                    (buy_trans+(buy_value-buy_trans).*(rand(1,N)-0.7))...
                    ]))';
output.sell_price=max(abs([...
                    (avg_trans+(avg_trans-sell_value).*(rand(1,N)-0.3));...
                    (avg_trans+(avg_trans-sell_trans).*(rand(1,N)-0.3));...
                    (sell_trans+(sell_value-sell_trans).*(rand(1,N)-0.3))...
                    ]))';
nob=setxor(1:N,trans_record(:,2)); nos=setxor(1:N,trans_record(:,3));
output.buy_price(nob)=max([...
                    (buy_value(nob)<=avg_trans).*...

(buy_value(nob).*(ones(1,length(nob))-0.15*rand(1,length(nob))));...
                    (buy_value(nob)>avg_trans).*...

(avg_trans+(buy_value(nob)-avg_trans).*(rand(1,length(nob))-0.7))]);
output.sell_price(nos)=min([...
                    (sell_value(nos)>=avg_trans).*...

(sell_value(nos).*(ones(1,length(nos))+0.15*rand(1,length(nos))));...
                    (sell_value(nos)<avg_trans).*...

(avg_trans+(avg_trans-sell_value(nos)).*(rand(1,length(nos))+0.3))]);
buy_price=output.buy_price'; sell_price=output.sell_price';
wrong_buy=find(buy_price>buy_value);
wrong_sell=find(sell_price<sell_value);
buy_price(wrong_buy)=buy_value(wrong_buy);
sell_price(wrong_sell)=sell_value(wrong_sell);
output.buy_price=buy_price'; output.sell_price=sell_price';

%% Transition between the two codes:

for i=1:n
    output = meta_Market(output);
    input.buy_price=output.buy_price;
```

```
    input.sell_price=output.sell_price
    output = Market(input)
end
```