Stock Market with Boundedly Rational Agents

A. De Simone

21/07/2010

1 Introduction and discussion of the model

The aim of our work is to model a single stock market with boundedly rational agents, competing in the market trying to increase their cash following a personal strategy in the path outlined by Minority Game (MG).

In the stock market, as in MG, the agents have different strategies among which to choose each turn on the basis of the efficiency in the past experience. More in depth, in MG each agent makes a binary decision with the aim of making the minoritarian choice: the agents have a set of k strategies that, given the results of the last *mem* turns, forecast the new best choice. At the end of the turn a score is assigned to each strategy according to the performances and the best one is selected for the following turn.

The greatest difference between the MG and the stock market model is that in the last case there is not a defined preference between minority and majority, but the reward is directly decided by the negotiation. Moreover the decision is not binary, sell or buy, but the agents have to decide both whether to buy or to sell and the price at what they want to realize the transaction.

We suggest two ways to cope with these problems: random prices and Artificial Neural Network (ANN). The first proposal is to use strategies considering only the sign of the variation of past prices and forecasting only the sign of the next variation, without giving any information about the amplitude of the variation itself. In this case the price will be decided randomly as a gaussian around the present price. This version of the Stock Market is then directly joining MG and the random agents stock market without introducing new complexity.

The second suggestion, that is here discussed, is to define each strategy as a field that, given the *mem* previous values of price, calculates a forecast of the new price.

The main difficulty in this case is defining the field coherently with the model and reality: ANN can be useful in affording this problem. The agents

will be equipped with k strategies as Artificial Neural Networks based on the Multi-Layer Perceptron concept, among which to choose with the usual score system: the ANNs will get as input the *mem* last price values and will calculate the forecast of the new price as output. Then the agents will decide whether to buy or sell on the basis of the difference between the present and the forecast price, making the most convenient offer considering the prices already present in the logs; it is a direct consequence that agent will not buy at a price higher than the forecast one and sell at a price lower, so that in this case they will give up to make an exchange in the turn. An handicap is assigned each turn to the strategies, according to the precision of the forecast: each strategy will receive an incremental handicap equal to the absolute value of the discrepancy between the predicted and real price, so that the best strategy will be the one with the minimum handicap.

Each ANN has been trained on real data from a real historical time series : it is worth underlining that it is not important for ANN to be really able to predict something, they should only give reasonable results compatibly with the real behaviour of traders. In this particular case a time series of the stock "Fiat" price has been used.

As in the simple stock market model [1], agents will have a counter of stocks and cash, but in this case they will renounce to buy if they have not the needed quantity of money available and to sell if they have not stocks. Agents without stocks and not able to buy will go out of the market.

Moreover a variation of the model has been studied, following another path in the creation of the ANN in order to make the forecast more realistic: before the training the raw input and output data has been scaled by the division to $2 * i_0$ where i_0 is the first input value, obtaining values near 0.5. Then the ANN has been trained on this rescaled data; the right forecast will be obtained multiplying the output to $2 * i_0$.

2 Code discussion

The ANNs have been trained on 1000 patterns made of *mem*+1 subsequent values of the input time series, the first *mem* as input and the last one as output. Each pattern has been extracted at a random point from the whole time series of the stock Fiat adjusted price from 01/01/2003 to 13/07/2010. The Multi-Layer Perceptron algorithm has been used by the Python library *bpnn.py* [2].

An Artificial Neural Network is a model elaborating data non linearly (fig.1). Each ANN receive a vector of *ci* inputs that is multiplied by a matrix W_i obtaining a vector of *ch* values; the sigmoid function, in our case an hyperbolic tangent (fig.2) is applied to all the values obtaining the hidden layer vector. The operation will be repeated with a new W_o matrix and then the sigmoid is applied another time, obtaining the output vector.

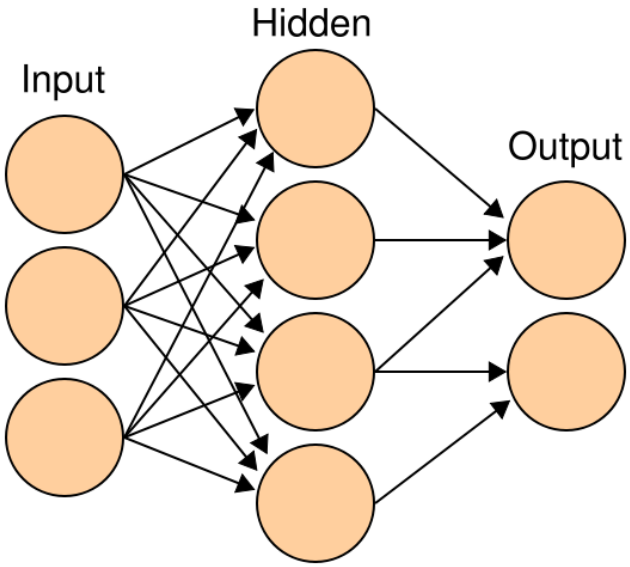

Figure 1: Schematic representation of an Artificial Neural Network (C.M.L. Burnett - Wikipedia user Cburnett).

The whole operation is

$$\mathbf{v}_o = f(f(\mathbf{v_i} \cdot W_i) \cdot W_o) \tag{1}$$

where W_i and W_o are the ANN's matrices and $f(\mathbf{x})$ is the sigmoid function.

In our case the number of input will be the last *mem* values of the historical series, the agents' memory, while the output consists of only a value: the forecast. The length of the hidden layer vector is arbitrary and in our case a 4 values vector has been used.

The ANNs creation and training has been performed with a simple Python code using in large part the already mentioned library [2]. It is worth reporting the code used to produce the patterns from the historical series:

```
def createpat(f,ndat,mem,ch,co):
   ts=readDatalist(f)

   pat=[]
   for i in range(ndat):
       pat.append([[0]*mem,[0]])
       ran=random.randint(0,len(ts)-(mem+1))
       for j in range(mem):
           pat[i][0][j]=ts[ran+j]
```

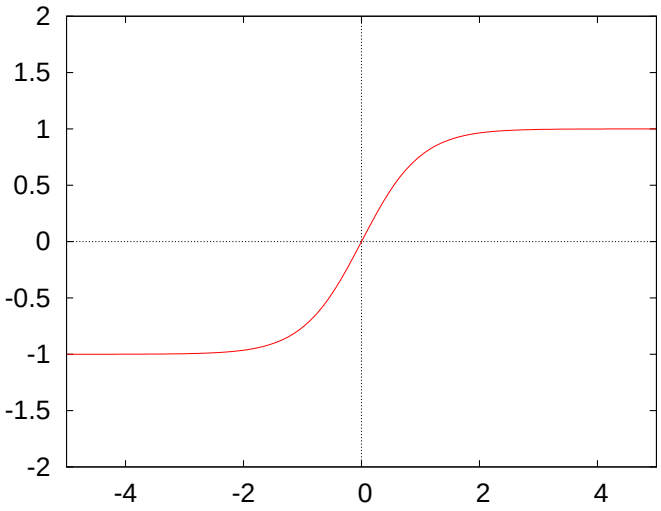

Figure 2: Example of sigmoid function realized by the hyperbolic tangent.

```
    for j in range(co):
        pat[i][1][j]=ts[ran+j+mem]

minV=findmin(ts)
maxV=findmax(ts)

for i in range(ndat):
   for j in range(co):
       pat[i][1][j]=(pat[i][1][j] - minV)/(maxV - minV)

results=[pat,minV,maxV]
return results
```

The function open the file, reads the data with the `readDataList(f)` function [1] that create a list of the prices. The `for` cycle chooses a random point in the list and create the input list with the first `mem` values and the output list with the last one. The operations are repeated `ndat` times creating a set of patterns used directly for the training.

At the end of the training of each ANN the matrices are saved in a correspondent file, ready to be read by the NetLogo code. When the `SETUP` button of the NetLogo form is pushed a set of breed of type `nns` is created and all the information about the ANNs are stored there; nevertheless this breeds will never act as real agents: the choice to use breeds to store ANNs information is due to code simplicity. Each `nn` owns as variables the matrices, the minimum and the maximum value of the output, the forecast and the handicap value.

The effective agents will be the `traders` characterized by the number of stocks and cash owned, a set of variables for the state (buy/sell/pass/out-of-market) and a list where the id of the ANN knew by the trader are stored.

A technical difficulty has been the realization of the matricial product with NetLogo: the definition of a function with a return value performing this task has considerably simplified the code.

```
to-report multmatr [vi m] ;multiplies a vector vi to a matrix m
  let vo []
  let j 0
  repeat (length (item 0 m)) [
   let somm 0
   let i 0
   repeat (length vi)[
     set somm (somm + (item i vi) * (item j (item i m)))
     set i (i + 1)
   ]
   set vo lput somm vo
   set j (j + 1)
  ]
  report vo
end
```

The function receives as input the vector `vi` and the matrix `m` performing the product and writing the results summing up to the `vo` vector that then is returned. In a similar way a `sigmoid(vi)` function has been written in order to apply the sigmoid to all the elements of the list `vi`; all the operations needed to calculate the output of an ANN are performed by the following code:

```
to-report outputnn [inp0 wi0 wo0 minV0 maxV0]
  set inp0 lput 1 inp0
  let temp []
  set temp sigmoid(multmatr inp0 wi0 )
  set temp item 0 (sigmoid(multmatr temp wo0))
  report ((temp * (maxV0 - minV0)) + minV0)
end
```

It is worth mentioning that `report` line returns the value already rescaled considering the right data range (the output of the ANN is in the [0,1] range).

Once the `GO` button is pushed the simulation starts: at each turn (tick) the ANNs read the last `mem` values and forecast the new price. All the agent will read, in a randomly chosen order, the forecast from the most performing ANN they know and decide if they want to sell or buy, depending on if they

think the price will increase or decrease; then they will offer a price to buy or sell and this value will be written in the corresponding log. An important point is now defining how this agent will decide the price at which to make the offer: the path followed in the model is to propose a price in the medium point between the forecast and the first price in the ordered logs, that means the lowest for sellers and highest for buyers. Alternative choice could be to offer a price slightly different from the log price, higher in the buyers case, lower in the sellers case. An alternative rule has to be followed in case of an empty log: in this case has been chosen to use the last price `exePrice` as the reference price.

The transaction will be executed when the sell price will be lower then the buy price: in this case cash and stocks are exchanged and the correspondent values in the logs are deleted. The code performing the transaction is shown in the case the active agent is a seller:

```
if sell [
  set logS lput tmp logS
  set logS sort-by [item 0 ?1 < item 0 ?2] logS
  ;show logS

  if (not empty? logB and not empty? logS) and
  item 0 (item 0 logB) >= item 0 (item 0 logS)
  [
     set exePrice item 0 (item 0 logB)
     let agB item 1 (item 0 logB)
     let agS item 1 (item 0 logS)

     ask trader agB [set stocks stocks + 1
        set cash cash - exePrice]
     ask trader agS [set stocks stocks - 1
        set cash cash + exePrice]
     set logB but-first logB
     set logS but-first logS
     set history but-first (history)
     set history lput exePrice history
     ask nns [
         set handicap (handicap + abs ((forecast - exePrice) / 100))
         set forecast outputnn history wi wo minV maxV
     ]

   ask traders [
     set nnset sort-by [[handicap] of (nn ?1) < [handicap] of (nn ?2)] nnset
     set forecastt ([forecast] of nn (item 0 nnset))
   ]
```

```
  ]
]
```

Once the transaction has been performed, the executive price is updated, the handicap is increased on the basis of the precision of the previous forecast and all the ANNs calculate the new forecast.

The agents can also decide to pass and not to participate to the market if they cannot afford the stock or if they want to sell and have no stocks: when both the conditions are realized the agent goes out of the market.

In order to make the model more realistic the logs are cleaned after a day of transaction, represented by a `tick-for-day` number of ticks.

As discussed before, a variation of the model has been discussed, using resized value divided to $2 * i_0$ where i_0 is the first input value.The results in the different cases will be compared. The NetLogo code implementing the variant is the following:

```
to-report outputnn [inp0 wi0 wo0]
   let inpscaled []
   foreach inp0 [set inpscaled lput (? / (2 * (item 0 inp0))) inpscaled ]
   set inpscaled lput 1 inpscaled
   let temp []
   set temp sigmoid(multmatr inpscaled wi0 )
   set temp item 0 (sigmoid(multmatr temp wo0))
   report (temp * 2 * (item 0 inp0))
end
```

3 Results

3.1 Data not rescaled

A simulation has been performed using raw input data and rescaling the output in order to obtain a value between 0 and 1. We created 100 traders each one knowing 4 strategies from a set of 90 ANNs (fig.3). In this case the forecast of ANNs is not completely realistic because the values are strongly oscillating and sometimes strongly different from the last price. In figure 4 the strong oscillation of the price is shown. It is worth stressing that the scoring system favorites strategies forecasting prices largely different from the present price: for example an high price will win the negotiation and the forecast of a strong price increase will become true.

It has been observed in all the simulation that, after a period of transaction, more and more agents come to the pass phase until all the agents renounce to negotiate. This "death" of the market is shown in figure 5.

3.2 Data rescaled

In the case of rescaled data the forecasts are less oscillating and more coherent with the previous pattern. A simulation with 100 traders knowing 2 strategies from a set of 90 has been performed: a general preference for buying respect to selling has been observed. This preference generates an asimmetry between seller and buyers and a strong growth of the price until the buyers finish their cash, they go in the pass phase, the market has a brief crisis and then the negotiations stop (fig.6).

Another simulation has been performed with traders knowing 4 strategies: in this case the general preference for optimistic strategies is even stronger, since with 4 strategies is more likely that the agents know at least one of the most optimistic strategy, that in this case means the strongest. We observe also in this case a growth of the market: in the case showed in fig.7 the growth is so rapid that the sellers finish the stocks and no negotiation is then possible.

4 Conclusions

An ABM model simulating the negotiation of boundedly rational traders acting in a stock market has been performed. In our model each agent knew a limited number of ANN forecasting the new price on the basis of a short memory; the agents choose the ANN to be used with a scoring system based on the goodness of the forecasts in the past. The greatest difficulty in the model has been the training of the ANN, performed with a Python code on raw input data: a more sophisticated approach is necessary to

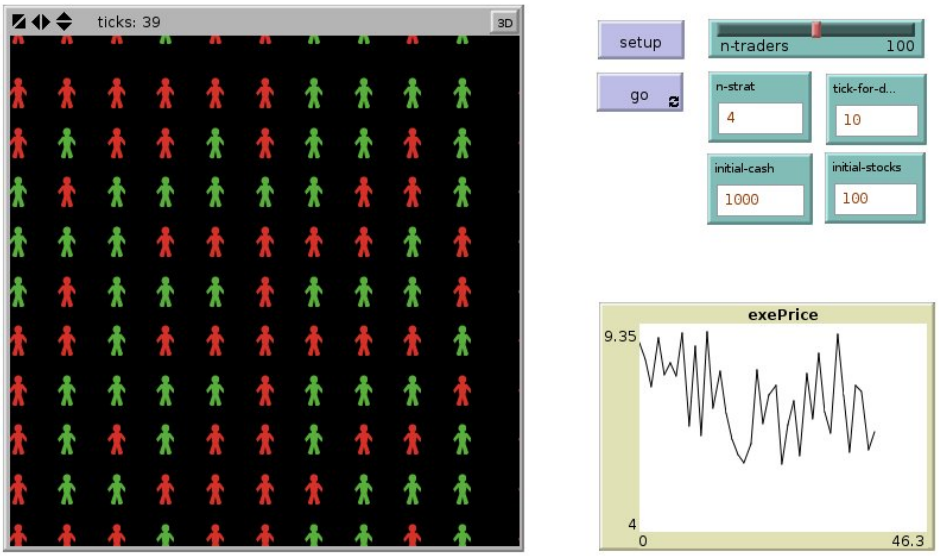

Figure 3: Example of the NetLogo window during the simulation with not rescaled data (buyers are red, sellers are green).

obtain forecasts compatible with experiments. A first proposal is to focus the attention on the returns instead of on the absolute value of the price.

We performed successfully the simulation with the NetLogo software, implementing both the ANN algorithm and the stock market mechanism, observing an effective negotiation and the transition of the system toward a steady state with no active agents. The features on which to focus the attention for further improvements are the choice of the offer, given the last price and the forecast, the choice of the offer when the logs are empty and the setting of the parameters.

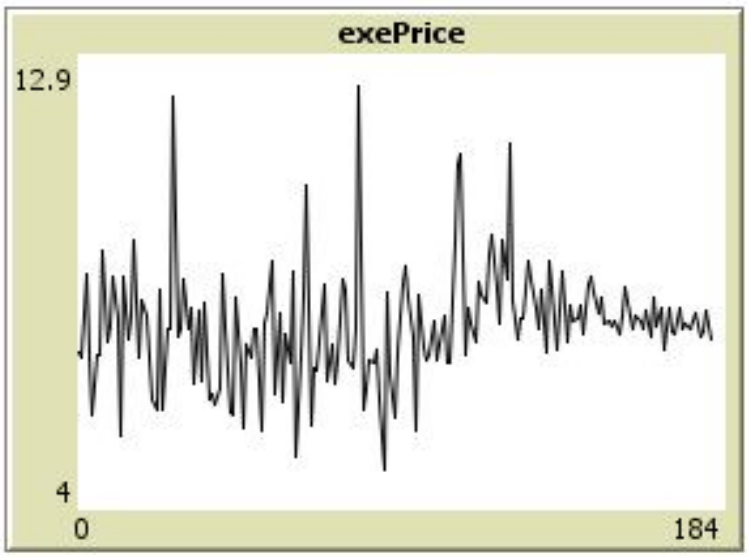

Figure 4: Example of the strong oscillation of the price.

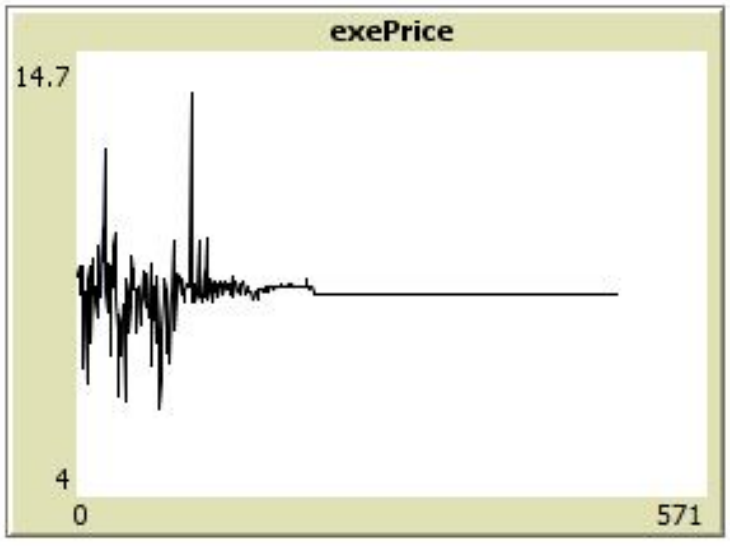

Figure 5: Example of the behavior of the price for long time scale, when all the agents have renounced to negotiate because they have not enough cash or stocks.

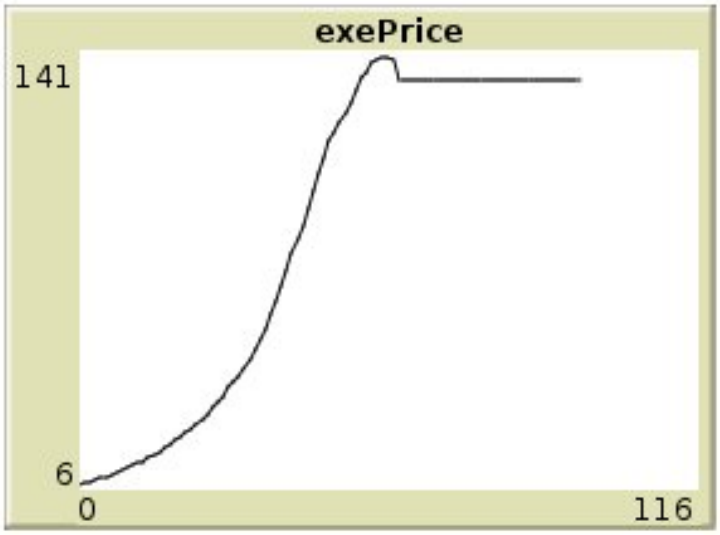

Figure 6: Example of the behaviour of the price in the rescaled case with 100 traders knowing 2 strategies.

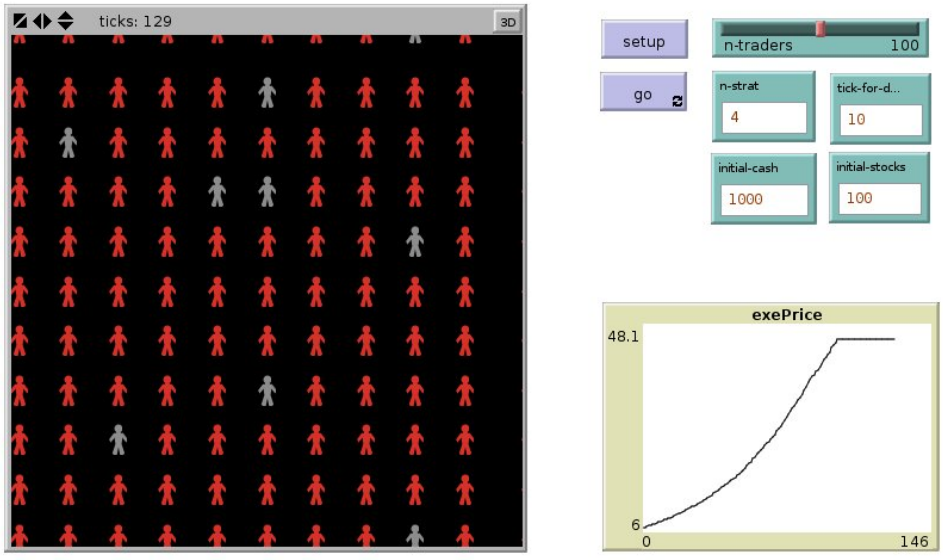

Figure 7: Example of the behaviour of the price in the rescaled case with 100 traders knowing 4 strategies.

References

[1] P.Terna, http://web.econ.unito.it/terna/

[2] N. Schemenauer, http://python.ca/nas/

[3] E. Moro, *The Minority Game: an introductory guide*, Advances in Condensed Matter and Statistical Physics (2004)

[4] C. Gou, *The Simulation of Financial Markets by an Agent-Based Mix-Game Model*, JASSS (2006)