ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:

Modeling and Simulating Social Systems with MATLAB

Project Report

Simulation of Trading in an Artificial Stock Market

Nicholas Eyring & Youri Popoff

Zürich

14.12.2012

Table of Contents

0 Abstract

This paper describes and presents our implementation of an analytical model of a stock market as well as research into three specific areas: market volatility feedback, price regulation, and financial bubbles. We show that when past market volatility is taken into account by traders, fat tails appear in the investment return plots and the market tends towards instability. We also demonstrate that price regulation policies are very effective in stabilizing a given financial market. However, there is a potential trade-off between market stability and economic welfare when such policies are put into place. Furthermore, we show that it is possible to reproduce the transaction price plot characteristics of a financial bubble using an analytical model of the stock market. In conclusion, we indicate that given enough investment in research, the financial services industry may soon put analytical models to use in order to make better decisions and ultimately circumvent financially unfavorable situations.

1 Introduction

Over the past decades the financial services industry has firmly established itself as a key player in modern society. Its successes and, more importantly, its failures have persisted to have broad repercussions all over the world. History has shown us time and again that markets do crash and that the price of picking up the pieces is a serious burden on economic growth and human progress.

1.1 Motivation

At the heart of the financial system lies the stock market. It can be used as a general indicator of the state of an economy. By analyzing the evolution of the stock market over time, insight into overall investor confidence, individual firm performances and the economy as a whole can be gained. We therefore believe that a good analytical model of trading on the stock market could be used to foresee and ultimately prevent financially unfavorable situations.

Figure 1: Using the stock market as an economic indicator

1.2 Goals

The primary goal of this project is to create a realistic model of a stock market which can simulate the evolution of the market transaction price over time depending on a number of appropriately chosen parameters. Given such a model, we can carry out a number of useful experiments. We identified volatile markets as an extremely undesirable feature of any modern economy and therefore decided to focus our experiments on market stabilization.

1.3 Fundamental Research Questions

- *How does past market volatility affect the future price of a stock?*
- *To what extent can price regulation be used to stabilize a volatile market?*
- *How accurately can financially unfavorable situations be modeled?*

Our first research question is equivalent to that which is handled in a paper we read whilst researching the topic ([1]). Our main model is based on the model described in the paper. Therefore, we wanted to confirm the results presented by the authors in order to validate our implementation of their model before using it for further research. The question itself concerns market volatility. It aims to determine the volatility feedback mechanism which needs to be implemented into the model in order to simulate real-world trading more accurately.

The second research question investigates the effectiveness of price regulation as a tool for government intervention in volatile markets. It aims to show if/how price regulation could be used to stabilize a market.

Our third and final research question involves using our model to simulate financially unfavorable situations. More specifically, we decided to attempt a simulation of a financial bubble using our model.

In order to meet our ambitions, we needed to find a good model which could accurately reproduce the dynamics of a real stock market.

2 The Model

Since the beginning of the project, we realized that modeling the entire stock market would be a monumental task. We had to start small. The model we implemented involves one single publically traded firm on the stock market. It executes an "agent-based" simulation. The "agents" are represented by the traders, each of which have an initial set of shares and liquidities which evolve over time. The "playground" is the market in which the different traders interact. For our simulations, we keep the number of traders finite and constant. We also give each trader the same initial number of shares and liquidities.

2.1 Price Formation

The engine driving any market model is the price formation process. It determines the market price of a good at a specific point in time. In our initial research, we discovered that there are two primary mechanisms used for price formation in artificial financial markets: The "clearing house" mechanism, and the "limit order book" mechanism. The "clearing house" mechanism operates on basic economic theory. Supply and demand data is accumulated over time in order to set up the familiar supply and demand curves. When supply matches demand we have the equilibrium price: "The market is cleared at the intersection of demand and supply curves" (Raberto, 2005):

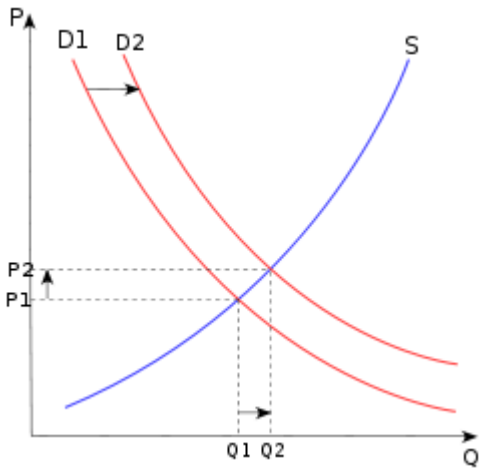

Figure 2: Supply and demand curves with a shift in demand (D1->D2). Source: Wikipedia

On the other hand, the "limit order book" mechanism works similarly to a double-auction. Interested buyers place bids for a good whilst sellers place asking prices. The bid and asking prices are stored in separate "order books". A transaction will occur at either the highest bid price or the lowest asking price (depending on the situation) if there is a price overlap between the two books.

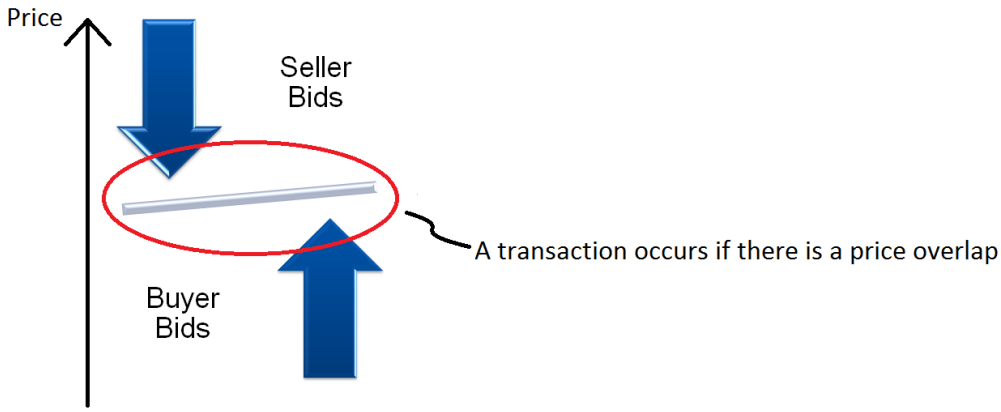

Figure 3: Limit order book mechanism

After an in-depth consideration of these two possibilities we determined that the "limit order book" mechanism for price formation is better suited to the specific market we are modeling:

"The clearing house is an unrealistic description of the way stock exchanges operate around the world. Progress in information technology and the increasing deregulation of exchanges have led to a wide adoption of the limit order book for price formation." (Raberto, 2005)

In this market structure, the potential buyers and sellers are individually responsible for setting the price of their respective bids.

"Modeling and simulation of a double auction artificial financial market" ([1]) is a paper written in 2005 which describes a model using the limit order book mechanism for price formation. The model we implemented is firmly based on the model described in the paper.

Example:

Figure 4: Real world example of a limit order book (Google stock). Source: ece.cmu.edu

2.2 The Market

Time is separated into days in order to imitate a real-world stock exchange. Every day, the market is open for a specific number of hours. Traders can only enter orders into the buyer or seller order books when the market is open. At the end of the day, the market closes. This means that all remaining buyer and seller bids are cleared from the books. Bids which are older than a given age also get cleared and the respective traders issue new bids immediately.

Once the market is open for the day, traders are randomly chosen for issuing orders. The time period between two orders follows an exponential distribution. Once a trader is chosen (following a uniform distribution), it issues either a “buy” or “sell” order with a predetermined probability. The price of the new order follows a Gaussian distribution with the price of the most competitive valid order currently in the respective book as the mean. The number of shares to buy/sell follows a uniform distribution ranging from one single share to the maximum number of shares the trader can buy/sell without ending up with negative asset values. A transaction will occur if there is an overlap between bid and asking prices. Otherwise, the order is stored in the respective book and no transaction occurs. If the trader is a buyer, the transaction occurs at the most competitive (lowest) selling price below the trader’s bid price. If the trader is a seller, the transaction occurs at the most competitive (highest) buying price above the trader’s bid price.

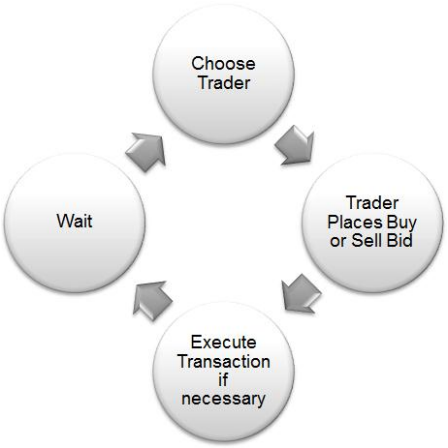

Figure 5: Sequence of events once the market is open

Since the number of shares is likely to be different for the overlapping orders, multiple transactions may occur at the same point in time. There can be one seller for multiple buyers or one buyer for multiple sellers whenever this is the case. We therefore calculate the weighted average of the individual transaction prices in order to determine the current market transaction price of the stock whenever this is the case:

$$current\ market\ price = \frac{\sum shares * price}{\sum shares}$$

It is also possible that a bid cannot fully be executed. For example, a buyer can ask for more shares than are currently available at the bid price. In this case, the buyer purchases the number of shares which are available and a new bid for the remaining shares gets placed in the buyer order book.

2.3 Measuring Market Volatility

In finance, the "rate of return" (ROR) on an investment is an indicator which represents the ratio of money gained or lost on an investment relative to the amount of money invested. It is usually expressed as a percentage. In a stable market, we would expect that the return values (ROR values) would not change very much over time. On the other hand, in a volatile market, fluctuation of return values is expected. Therefore, we can calculate return values as an indicator of the volatility of the market in the past.

Return values are usually calculated either with respect to a single time period ("single-period returns"), or averaged over multiple time periods ("multi-period average returns"). In our model, we take the standard deviation of multiple single-period return values and use the result as an indicator of past market volatility.

The single-period return values are calculated logarithmically using the following formula:

$$r_{log} = 100 \cdot ln\left[\frac{P_{final}}{P_{initial}}\right], where: \begin{cases} P_{final} = final\ transaction\ price \\ P_{initial} = initial\ transaction\ price \end{cases}$$

$$\rightarrow \quad \sigma_T = stdev(r_{log})$$

This is the same method that the authors of the paper which our model is based on ([1]) used to measure market volatility.

In our model we apply a constant time window with a constant number of single periods in it when calculating the standard deviation (σ_T) of the logarithmic returns (r_{log}). If the market is stable, we would expect a small σ_T whereas if the market is volatile, we would expect a larger σ_T.

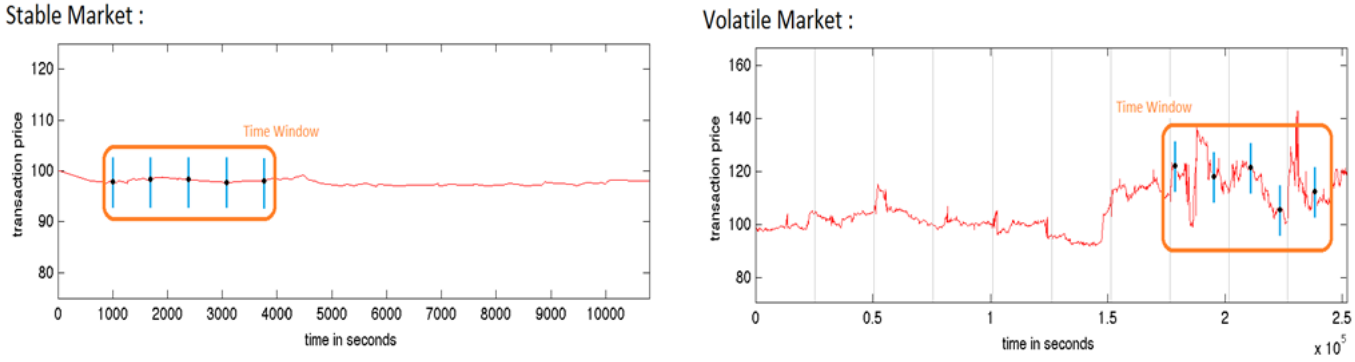

Figure 6: Stable market vs. volatile market

2.4 Taking Past Market Volatility into Account

In our model, volatility feedback occurs solely in the prices of new bids. Heavy fluctuations in the transaction price in the past imply that traders are less confident in their estimations of the real value of the good. The stock had been frequently under- and overvalued in the past. Since the traders are less sure of the true value of the stock, this results in a larger price spread in the new bid prices for both buyers and sellers. We therefore have a positive feedback loop built into the system. This means that fluctuations in the present market transaction price get amplified and turn up in the future transaction price. This again causes prices to deviate more from the latest bid values and the market tends towards instability.

We can implement the volatility feedback into our model by dynamically modifying the variance of the Gaussian distribution that the new bid prices are based upon. We use our volatility measure σ_T in order to do this:

$$\sigma_{new} = k \cdot \sigma_T\ ,\ k = const.; \qquad \sigma = \alpha \cdot \sigma_{new} + (1-\alpha) \cdot \sigma\ ,\ \alpha \in\ ]0,1]\ ; \qquad new\ variance = \sigma^2$$

When volatility feedback is turned on, a new variance is calculated every time a trader makes a new bid. Otherwise, a constant variance is used throughout the simulation.

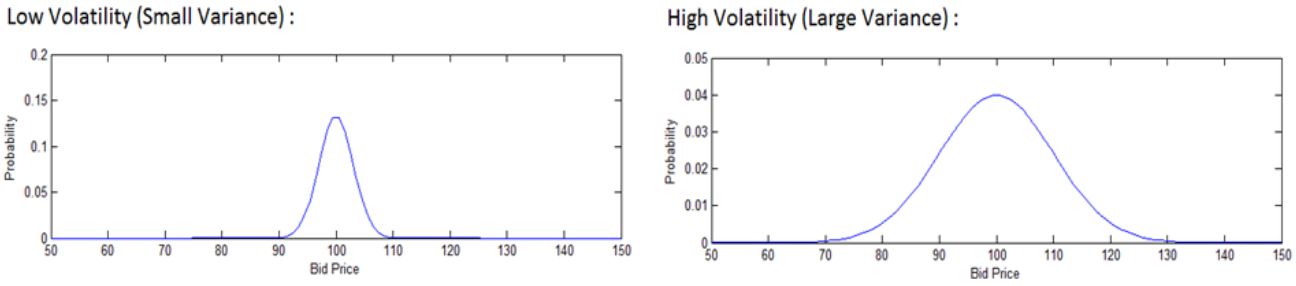

Figure 7: Bid price probability distributions for markets with different volatility

2.5 Price Regulation

Price regulation is a policy which consists of implementing a combination of price floors and price ceilings into the market. It inhibits the transaction price from going below a price floor or above a price ceiling at any point in time. When price regulation is turned on in the model, prices which would be out of bounds are simply set to the nearest legal limit. For example, if a buyer tries to place a bid with a price below the current price floor, the bid price gets adjusted so that it equals the value of the price floor.

For a given simulation, price regulation can be turned on or off. The price regulation consists of a price floor and a price ceiling which are valid throughout the entire simulation. When turned on, three parameters are needed: the initial value of the price floor (p_{f0}), the initial value of the price ceiling (p_{c0}), and the percentage growth per day (g). By allowing growth, the values of both the price floor and ceiling can change over time. We did this in order to account for economic growth (which is desired). The resulting price ceiling and price floor become linear functions of time (t [seconds]):

$$\begin{cases} p_c(t) = p_{c0} + \dfrac{g \cdot p_{c0}}{24 \cdot 60 \cdot 60} \cdot t \ : \ price\ ceiling\ over\ time \\ p_f(t) = p_{f0} + \dfrac{g \cdot p_{f0}}{24 \cdot 60 \cdot 60} \cdot t \ : \ \ price\ floor\ over\ time \end{cases}$$

2.6 Modeling Financial Bubbles

A financial bubble is a phenomenon where stock is consistently overvalued by traders over a period of time. Trading occurs at prices which are considerably inflated. Prices keep rising until at some point an awareness of the overvaluation occurs. Once this happens, the bubble bursts. Prices fall rapidly and consistently until equilibrium is regained. Real-world examples of financial bubbles are numerous. The “Roaring Twenties” stock-market bubble (1922-1929), the Dot-com bubble (1995-2000), and the U.S. real estate bubble (as of 2007) are prominent examples.

After much discussion, we decided to use non-symmetric probability distributions for bid prices in order to simulate financial bubbles using our model. At a specific point in time, the mean of the Gaussian distribution which new bid prices are based upon increases significantly. Instead of using the most competitive valid bid price as the mean, we set the mean to a value which is considerably higher. This indicates that prices are much more likely to increase than decrease. Later, we reverse the trend by setting the mean to a value which is considerably lower than the most competitive valid bid price. This makes prices extremely likely to fall and keep falling, essentially bursting the bubble. At some point thereafter the model switches back to the regular probability distributions for bid prices, thereby inducing equilibrium once again.

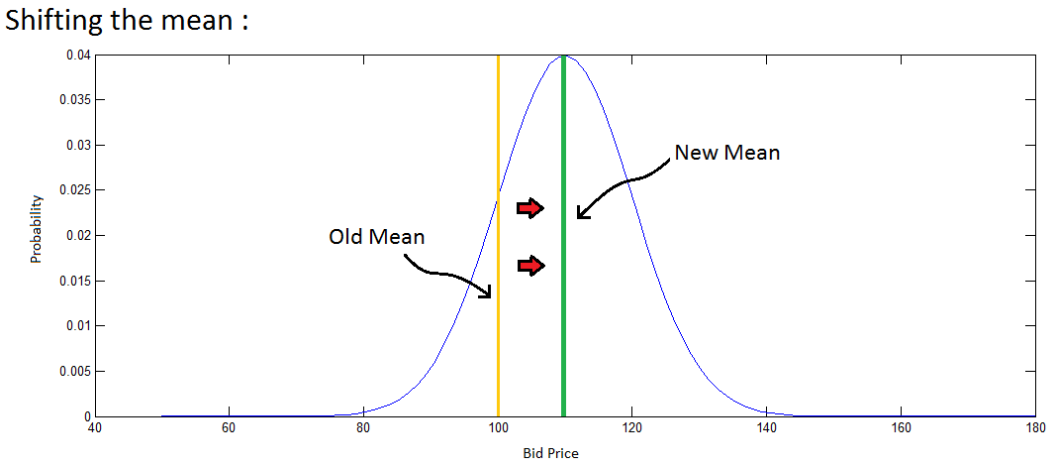

Figure 8: Shifting the mean for new bid prices causing bubble

3 Implementation

The program is divided into three main sections:

- Section 1: input

- Section 2: computations

- Section 3: plotting

The implementation of the model itself is found in the computational section. This is where the stock market gets simulated. The other two sections were created for organizational reasons. The input section (section 1) sets values for all relevant parameters needed for a simulation. The computations section (section 2) carries out a simulation for one set of input parameters. The plotting section (section 3) organizes the results of every simulation for analysis. Given that the three sections work time-independently from one another, they can be called separately. However, they must be called in order.

3.1 Simulation

In order to keep the implementation as clear and convenient as possible, the model is separated into numerous MatLab functions. Each function carries out a specific task. MatLab structs are used to group all relevant parameters and variables together, simplifying function inputs and outputs. At the beginning of a simulation, the function *init.m* is called and creates these structs:

```
function [ SSM, SP, SPL ] = init( )
```

init.m output:

- SSM: System state matrices (previous transaction prices, trader assets, etc…)
- SP: System parameters (probability distribution parameters, number of days to simulate, etc…)
- SPL: System plotting parameters

Once all parameters have been defined, the actual simulation can begin: The function *main.m* gets called:

```
function [ SSM, SP, SPL ] = main( SSM, SP, SPL )
```

main.m is the heart of the model. It contains a for-loop which iterates through each simulation second. For every simulation second, *main.m* has a number of tasks:

1. Shift bid price mean if necessary (for bubble)
2. Update ages of all current bids

3. Calculate the logarithmic returns if necessary (end of a period)
4. Determine when the next trader gets chosen
5. Empty the books if the day is over
6. Choose a trader and create a book entry if necessary
7. Check the ages of existing bids and create new book entries if necessary
8. Update the weighted transaction price values

Most of the tasks above do not get completed inside the function *main.m*. Instead, various other important functions get called:

- 1. If the bubble simulation is turned on, the function *shiftMean.m* gets called to shift the mean of the bid price distributions:

```
%% Shift the mean
SP = shiftMean( SP, m );
```

- 2. In order to update the ages of all current bids, *main.m* calls the function *ageUpdate.m*:

```
%% Age Update
SSM.bookbpaging = ageUpdate( SSM.bookbpaging, SSM.sbbp );
SSM.bookspaging = ageUpdate( SSM.bookspaging, SSM.sbsp );
```

- 3. Logarithmic returns are needed to measure past market volatility. They are calculated with respect to a time period (*dt*) using the function *logReturns.m*:

```
%% Calculate Log Returns
if i == lrt       % time to calculate log returns?

      [ SSM ] = logReturns( SSM, SP );       % calculate log returns

      lrt = lrt + SP.dt;    % increment lrt for next log returns calculation

end
```

- 4. The time to wait until the next trader gets chosen follows an exponential distribution. The time at which a new trader gets chosen is calculated using the following code:

```
%% New Book Entry Section
if i == t         % time to choose a trader?

      Tau = 1+round(exprnd(SP.lambda));     % step between two book entries
                          % (random number; exponential distribution)
      t = t + Tau;      % new time at which to choose next trader!
```

- 5. In order to empty the books when the day is over, *main.m* calls the function *emptyBook.m*:

```
if t - m * SP.T > SP.T      % t - m * T = actual time in the trading day m

      m = m + 1;                        % increment number of days past
      [ SSM ] = emptyBook( SSM, SP );
end
```

- 6. A trader must now be chosen to make a bid. Since we distinguish between buyer and seller bids, *main.m* will either call the function *buyer.m* or *seller.m* in order to create a book entry and execute a transaction if necessary:

```
%% Book entry
ind = randi(SP.tnum, 1, 1);              % index of the chosen trader
stat = randi(2, 1, 1) - 1;        % choose between buyer (0) or seller (1)
if stat == 0                 % we have a buy order (0)

      [ SSM ] = buyer(SSM, SP, m, i, ind, arefresh, orind);

else           % we have a sell order (1)

      [ SSM ] = seller(SSM, SP, m, i, ind, arefresh, orind);

end
```

buyer.m and *seller.m* call various functions including those which execute the transactions. This is also where bid prices are determined. Both *buyer.m* and *seller.m* call the function *volatilityFeedback.m* in order to account for volatility in the market (if turned on):

```
if SP.volfeed == 1

      [ SSM ] = volatilityFeedback( SSM, SP, ind );

end
```

If price regulation is turned on, the bid price gets adjusted (if necessary) by calling the function *regulatePrice.m*:

```
if SP.regulate == 1

      p = regulatePrice(p, SP, t);       % regulate price if applicable

end
```

Once the bid price and number of shares has been set, the order gets executed if possible. *buyer.m* calls the function *buyerTransaction.m* whilst *seller.m* calls the function *sellerTransaction.m*. These are the only functions which actually execute transactions.

- 7. The ages of all the bids which are still stored in their respective books must now be checked to determine if they have expired or not. This is done by calling the functions *ageCheckBuyer.m* and *ageCheckSeller.m* respectively:

```
%% Age Check
[ SSM ] = ageCheckBuyer(SSM, SP, m, i);
[ SSM ] = ageCheckSeller(SSM, SP, m, i);
```

- 8. Since multiple transactions may have occurred during the current simulation second, the weighted average must be calculated. The function *weightedTP.m* is called in order to do so:

```
[ SSM ] = weightedTP( SSM, i );  % update weighted transaction price matrix
```

Once the for-loop has ended (every second has been simulated), the plotting section (section 3) of the program uses the generated data to create results and the simulation is complete.

3.2 Program Structure

-*Input* (folder *code/Input*):

This section generates the input files used in a simulation. A standard set of parameters is initialized in the function *init.m*. These values correspond to those which were chosen in the paper which we based our model on ([1]). Additionally, a parameter sweep is implemented. Section 3.3 of this paper (page!!!) discusses the purpose and implementation of the parameter sweep.

-*Computations* (folder *code/Computation*):

The computations section is the heart of the model. It carries out a simulation for a given set of input parameters. See section 3.1 of this paper (page!!!) for a detailed description of the computational part of the program.

-*Plotting* (folder *code/Plot*):

The results generated from various simulations can be plotted using the third and final section. Several plots are created for every set of output data, including plots of random variables. The random variable plots are present for debugging purposes. They can be used to verify that random variables are following their respective intended distributions.

-*Results* (folder *code/Data*):

Every set of results generated gets saved in the folder *code/Data*. This includes all plots as well as general simulation information (such as parameter choice and execution time).

-*Additional functions* (folders *code/Other* and *code/Test functions*):

The repository *code/Other* contains old versions of key functions. It serves as an internal back-up for functions which are either important or which we had often modified.

In addition, the code in the folder *code/Test functions* was written in order to experiment with various built-in MatLab functions as well as to compare ideas without having to modify the work done so far.

3.3 Parameter Sweep

When using a model to simulate a real social system, it is essential to run it on different sets of parameters as the model will react in various ways when exposed to different initial conditions. Consequently, the choice of appropriate parameter values is crucial.

A parameter sweep involves taking a parameter and varying it according to a sweep rule. The computations section then simulates the stock market for each specific value of the parameter. Sweeping through multiple parameters is also possible. In that case, a simulation will run for every allowed combination of parameters specified by the sweep.

The sweep rule of the parameter sweep is defined in the file *code/Input/param.txt*. This file has a specific structure that needs to be maintained for the program to function.

An example of a valid parameter sweep file is shown below:

```
SP.p0          100       1         110
SP.k           1.40      0.01      1.60
SP.volfeed     0         1         1
```

Figure 9: Example of *code/Input/param.txt*

In this example, the parameter sweep is defined as follows:

- SP.p0 (initial price) ranging from 100 to 110 with an increment of 1
- SP.k (correlation coefficient) starting at 1.40 and incremented by 0.01 until 1.60
- SP.volfeed (volatility feedback toggle) ranging from 0 (off) to 1 (on)

The sweep is implemented in the folder *code/Input* (input section of the model). At first, the function *paramSweepCall.m* is called, which in turn calls the recursive function *paramSweep.m*. The parameter sweep file *param.txt* is read by *paramSweep.m* which automatically generates each set of values. All parameters which do not appear in the sweep are set to their default values as described in *code/Input/init.m*.

The parameter files generated by *paramSweep.m* are saved as MatLab-specific files in various subfolders with the following general name:

code/Data/Simulation/sim**/sim**parameter.mat*

Later on, the computations section runs a simulation for each of the individual parameter sets.

To facilitate finding the results of running the model on a specific set of parameters generated by the sweep, the file *code/Data/Simulation*/structure.txt* is created. This file can be consulted once an entire sweep is complete in order to find the results of a specific simulation.

3.4 Automatic File System

A typical parameter sweep generates thousands of parameter combinations which all need to be simulated individually. Since each simulation usually takes several minutes to complete, it would be very inefficient for users to have to start every simulation themselves. Clear organization of the output data is also desired to ease the analysis of results. Therefore, an automated system is needed.

The simulation is started by running the script *code/control.m*. At first, a file structure for the results needs to be created. To do this, the file *code/path.txt* is read. This file contains the definition of all the folder names of the file structure.

An example of a valid file containing these definitions is shown below:

```
Data
Simulation**
globalsim**info.txt
Compare
Sim**
sim**parameter.mat
sim**info.txt
sim**report.txt
globalsim**report.txt
sim**plot
Input
param.txt
structure.txt
```

Figure 10: Example of a valid file system definition file

In this example, the first line defines the name of the result folder (*Data*). The second line is a definition of the global simulation folder (*Data/Simulation***) whose info-file is called *Data/globalsim**info.txt* (third line). The corresponding report file is named *Data/globalism**report.txt* (ninth line). Note: the "**" are automatically replaced by the program with the simulation number.

These definitions are passed on to the folder management function *code/Input/foldermg.m* and the different subfolders are created. This function also calls the parameter sweep to generate the input. Thereafter, the control script (*control.m*) runs the simulation, plots the results, and saves them back to their respective folders.

Due to the number of random variables used, the model is prone to instability if certain initial values are chosen. Therefore, the control script contains a try and catch section in which the implemented model is called. This way, if one of the sub-simulations fails, the program simply notifies it in the report file and continues.

-*The compare folder:*

Comparing the results of a large number of simulations can be an overwhelming task. A comparison folder *code/Data/Simulation*/sim**/Compare* was created to solve this problem. The code can automatically take a single plot from each simulation and copy it into the compare folder. This facilitates data analysis when regarding a specific output variable of the model.

As an optimization, a condition was added which determines whether or not a plot should be copied to the compare folder. For example, if the transaction price exceeds 500 (five times the default initial value), then the simulation should not be included in the comparison. This way, potentially uninteresting results could be filtered out before comparison.

Further aspects:

Being able to reproduce results is very important when modeling. Therefore, the random generators use a fixed random stream. This is done using the following code:

```
%% Initialisation
s = RandStream('mt19937ar','Seed',0);% choosing & resetting random stream
%RandStream.setDefaultStream(s);
RandStream.setGlobalStream(s);
```

Report files (also called logfiles) containing the list of errors (if occurred), the execution time, and various other simulation details are also implemented into the model. Furthermore, each simulation generates an information file containing a list of all parameter values used.

4 Simulation Results and Analysis

When running simulations, all three of our research questions were handled separately. By turning on/off different functions of the model, specific aspects could be investigated in order to obtain conclusive results. Although the general focus of our experiments is market stabilization, we chose to examine features of volatile markets and approaches to market stabilization separately in order to gain clear and pertinent results.

4.1 Volatility Feedback

Our first research question reads as follows:

"How does past market volatility affect the future price of a stock?"

As mentioned earlier, the question is equivalent to that which is handled in the research paper which our model is based upon ([1]). Our goal was to confirm the results presented by the authors in order to validate our implementation of their model. They had first simulated the market without taking past market volatility into account (no volatility feedback mechanism) before comparing the results with those obtained when past market volatility was taken into account. Below are the primary results which they presented in their paper:

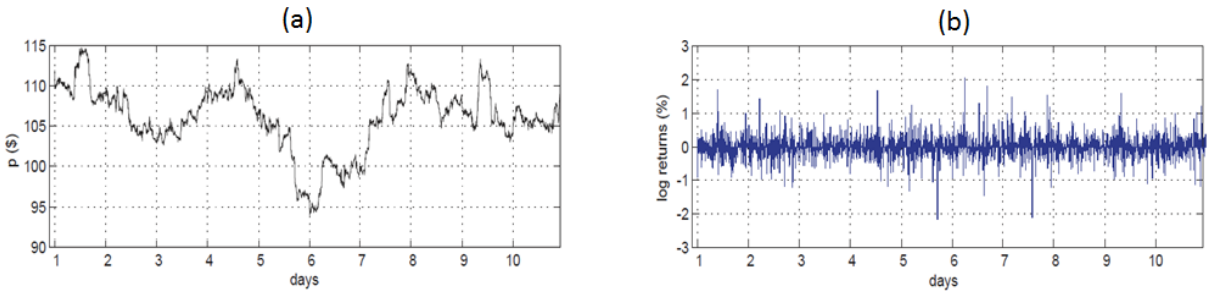

Figure 11: Results without volatility feedback as presented in paper ([1])

(a): Transaction price over time (b): Logarithmic return values over time

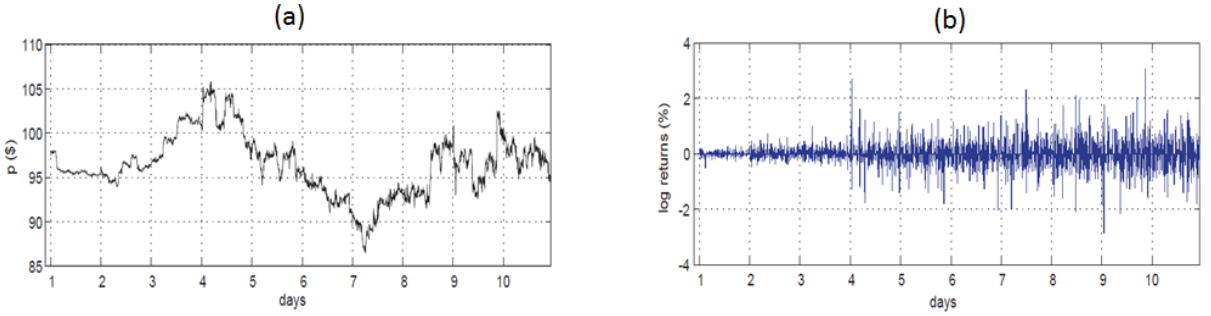

Figure 12: Results with volatility feedback as presented in paper ([1])

(a): Transaction price over time (b): Logarithmic return values over time

It is clear that when volatility feedback was not incorporated, the market behaved consistently over the entire simulation period. On the other hand, when volatility feedback was taken into account, the market was clearly more stable at the beginning of the simulation and fluctuated more and more as the simulation continued. This is clearly characterized by the "fat tail" which can be observed in the logarithmic return values plot. Wikipedia identifies "excessive investor optimism or pessimism leading to large market moves" as behavioral origins of fat tails in market return distributions. Below are results from our model of simulations with and without volatility feedback. Note that all parameters took their default values (see appendix 6.1 – page!!!) unless otherwise specified. This is valid for all results presented in this paper.

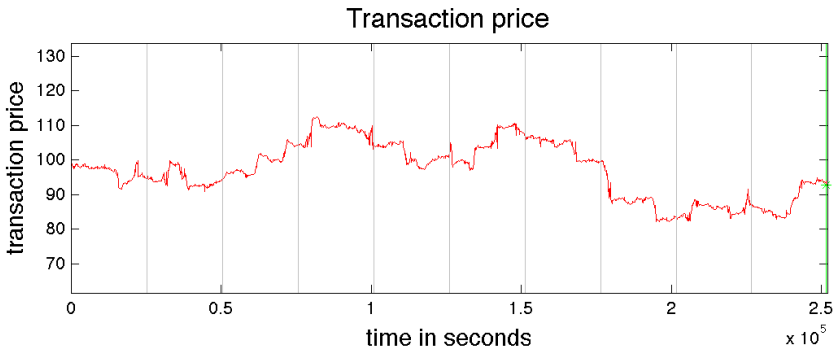

Figure 13: Our results without volatility feedback – Transaction price over time

Non-default parameters: SP.sigma = 0.0048

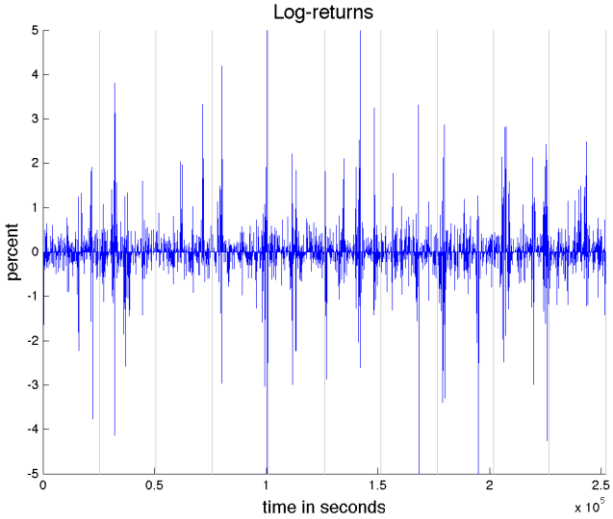

Figure 14 : Our results without volatility feedback – Logarithmic return values over time

Non-default parameters: SP.sigma = 0.0048

The vertical lines in both plots represent days gone past. This makes it easier to compare our results to those presented in Figure 11. Looking at the transaction price plot (Figure 13), we see that the market was more or less consistent throughout the simulation. This confirms what is seen in Figure 11a.

Concerning logarithmic return values, there clearly is jitter in our results. If we ignore the jitter, our results are remarkably similar to those presented in Figure 11b: the logarithmic returns appear to be independent of time. It seems we have succeeded in reproducing the results without volatility feedback. We believe that the jitter we see is simply a consequence of the random number generators used in the implementation of the model.

Following are the results of our model when volatility feedback is turned on:

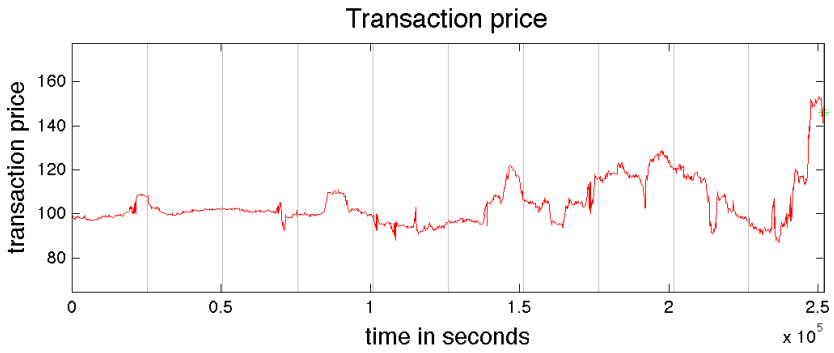

Figure 15: Our results with volatility feedback – Transaction price over time

Non-default parameters: SP.volfeed = 1 , SP.k = 1.66 , SP.korrk = 0.00043

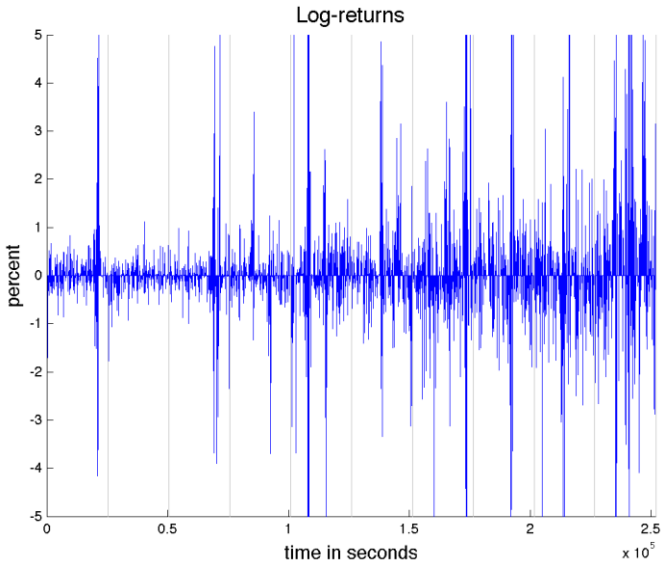

Figure 16: Our results with volatility feedback – Logarithmic return values over time

Non-default parameters: SP.volfeed = 1 , SP.k = 1.66 , SP.korrk = 0.00043

Looking at Figure 15, it is clear that the volatility feedback mechanism is working. The market transaction price is very stable at the beginning of the simulation and fluctuates more and more as time goes on. This is consistent with the results in Figure 12a.

Regarding logarithmic return values over time, a “fat tail” is clearly noticeable in Figure 16. This is a direct consequence of increasing fluctuation of the transaction price as time goes on. Once again there

seems to be jitter in the results. However, the underlying volatility feedback mechanism is behaving consistently with the results in Figure 12a and Figure12b.

When presenting our project, we were asked if the market stays unstable even after a long period of time. Below are the results of the same simulation which generated Figure 15 and Figure 16, except the simulation period is increased from 10 days to 20 days:

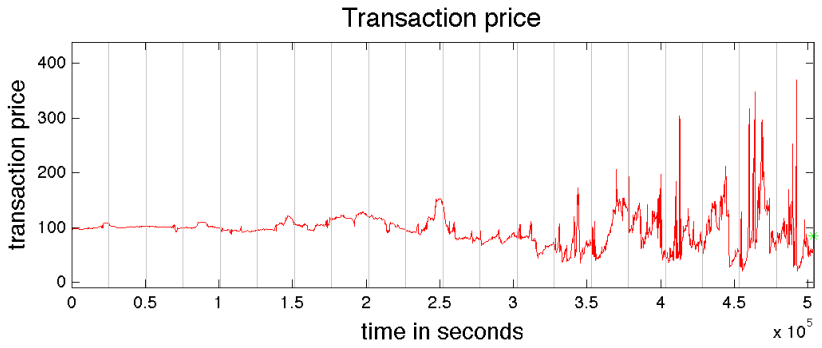

Figure 17: Long simulation results with volatility feedback (20 days) – Transaction price over time

Non-default parameters: SP.volfeed = 1 , SP.M = 20 , SP.k = 1.66 , SP.korrk = 0.00043

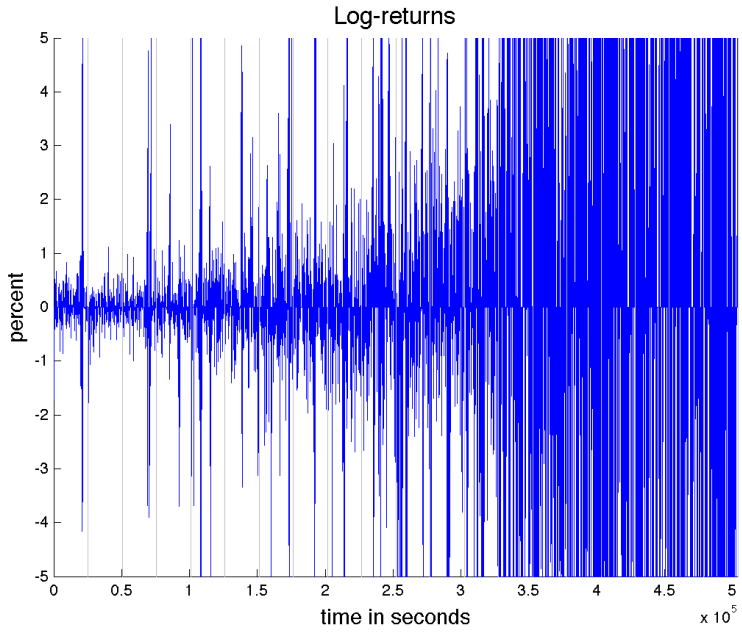

Figure 18: Long simulation results with volatility feedback (20 days) – Logarithmic return values over time

Non-default parameters: SP.volfeed = 1 , SP.M = 20 , SP.k = 1.66 , SP.korrk = 0.00043

As expected, it appears that the market does indeed tend towards instability. The “fat tail” in Figure 18 is extremely noticeable, and the transaction price plot in Figure 17 displays consistently increasing price fluctuation as time goes on.

4.2 Price Regulation

Our research question concerning price regulation reads as follows:

"To what extent can price regulation be used to stabilize a volatile market?"

In order to use price regulation in our model, three parameters are needed. The first two are the initial values of the price floor and price ceiling. The last parameter is the allowed growth per day (expressed as a percentage). Below are the results of a simulation when the allowed growth is set to 0% per day (no growth):

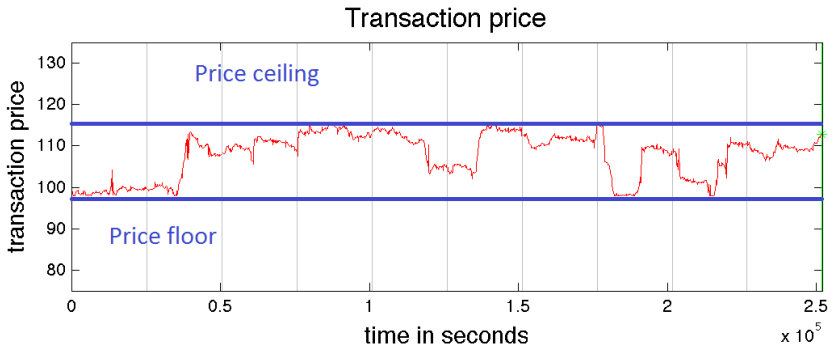

Figure 19: Price regulation results with zero growth

Non-default parameters: SP.growth = 0 , SP.regulate = 1 , SP.pF = 98 , SP.pC = 115

When growth is allowed ($g \neq 0$), the following results are obtained:

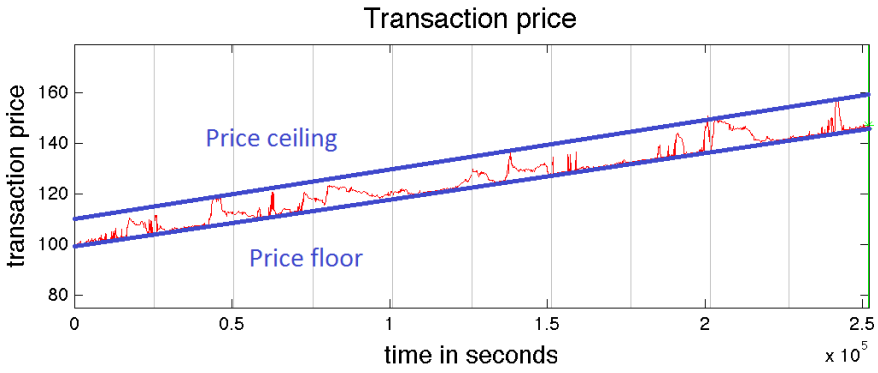

Figure 20: Price regulation results with non-zero growth

Non-default parameters: SP.growth = 16 , SP.regulate = 1

It is clear that the regulation works. By setting bounds on the transaction price, the market is forced into stability. However, we no longer have a free market. If the true (equilibrium) price of the good lies outside the legal bounds, economic welfare is lost. Price regulation therefore results in a possible trade-off between market stability and a free market environment. The challenge is choosing the right parameters so that as little economic welfare as possible is lost.

4.3 Financial Bubbles

Our final research question reads as follows:

"How accurately can financially unfavorable situations be modeled?"

Specifically, we attempted to simulate a financial bubble using our model. We wanted to reproduce the transaction price plot characteristics of real-world examples of financial bubbles using our model. Below is a plot of the NASDAQ Composite index illustrating the Dot-com bubble:

Figure 21: NASDAQ index illustrating the Dot-com bubble. Source: mathmajor.org

Following are the results obtained using our model:

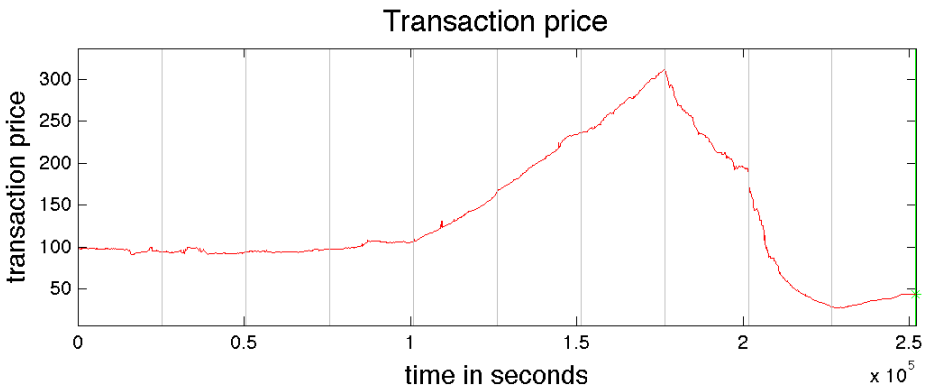

Figure 22: Financial bubble results

Non-default parameters: SP.devon = 1 , SP.t1 = 3 , SP.mu1 = 1.01 , SP.mu2 = 0.96

Clearly, the results shown in Figure 22 do have the characteristics of a financial bubble to an extent. We see a steady increase in price, then a peak, and then a steady decrease in price before flattening out again. However, the rates of change in price before and after the peak seem to be almost constant. This is not the case in the NASDAQ plot of the Dot-com bubble (Figure 21). The index seems to

increase exponentially before the peak and decrease exponentially afterwards. This result is, however, not surprising. Our model simply shifts the mean of the bid price distributions by a constant amount whenever there is a change. It appears that a gradual change in the mean would result in a transaction price plot which better represents the bubble in Figure 21. Despite this, our results do show promise for analytical models of financial bubbles. We show in Figure 22 that it is possible to reproduce at least some of the characteristics of financial bubbles using an analytical model. We can safely say that we achieved our goals concerning our third and final research question.

5 Discussion

It is clear that much work can still be done on the subject matter of this project. The results of our experiments are interesting, however much more time and effort would be needed to draw truly valuable conclusions from this kind of project. The main obstacle is the complexity of the system we are trying to model. Despite significant advances in information technology over the past couple of decades, no current model is sophisticated enough to be very accurate. As a result, traders still do not rely on analytic models of the stock market when investing. Further research needs to be done and the model which we implemented can serve as a basis for future research concerning this subject.

5.1 Shortcomings/Limitations

One of the most evident shortcomings of our model is that it is a closed system which only handles a single firm. The market is isolated. This means that the model does not take into account anything which happens outside of it. Being that the real-world stock market is undoubtedly affected by countless world-wide events, this is a serious limitation.

When implementing price regulation into our model, we simply forced traders to adjust their prices to be within the legal bounds imposed on the market. It is highly debatable whether or not this would actually be the reaction of real-world traders to an equivalent real-world market policy. An alternative possibility would be to simply wait until traders place legal bids. This may result in fewer transactions and perhaps a slowdown in trading activity. Either way, more research would be required to determine a more accurate bid mechanism for traders in these particular situations.

A significant general limitation of this type of model is that there are just too many important parameters involved in real-world stock market prices to model without a tremendous amount of research. There are simply too many factors which may affect the stock market which are not incorporated into our model. This fact imposes a severe shortcoming of our model. It would be hard to convince someone of the accuracy of results generated from our simulations given the number of parameters which are not taken into account.

One must be very aware of the shortcomings and limitations of such a model in order to correctly interpret the meaning of results generated by simulations. As the complexity of the model increases, this becomes harder to do. Generally speaking, a realistic model of a stock market will be inherently complex due to the high complexity of the real-world system. As a result, most stock market models bear the mutual shortcoming of having a long learning curve. This is potentially a serious barrier to the future adoption of analytic models of the stock market by members of the financial community.

5.2 Possible Improvements

An evident possible improvement to our model would be to complete the research on price regulation and causes of financial bubbles and build more accurate mechanisms into the model. This would better the credibility of our results and conclusions drawn with respect to our research questions.

Another obvious improvement would be to incorporate further parameters into the model. For example, so-called "landscape event" parameters could potentially be built into our model. As a result, the influence of real-world events outside of the market itself could be modeled to an extent. Clearly, a large amount of research would be involved.

A very interesting addition to our model would be to write a program where real-world stock market data could be fed into the model or analyzed against various simulation results. This could prove to be a useful tool when analyzing various market stabilization methods. Building an intuitive graphical user interface (GUI) to go with it would make the model easier to use and potentially attract more users.

Lastly, a natural continuation of the model would be to have multiple single-firm stock markets interacting with each other all in the same model. Entire real-world stock markets could potentially be modeled whilst tweaking certain parameters for each individual market.

5.3 Conclusion

Given what we have learned throughout this project, it seems to us that it will be some time before stock market models are taken seriously by members of the financial services industry. In spite of this, we consider our project a success with respect to our fundamental research questions. We did achieve our primary goal of creating a realistic model of a stock market which can simulate the evolution of the market transaction price over time. Furthermore, we were able to reproduce results from previous research using our own version of the model.

The experiments which we carried out on the topic of price regulation did yield interesting albeit non-surprising results. We showed that our model could be used to investigate potential market policies. This could be very valuable when undergoing further research.

Concerning financial bubbles, we are able to mimic the transaction price plot characteristics of real-world examples of the phenomenon using our model. However, there is no mechanism built into the model which causes a bubble without explicit non-provoked outside intervention. In conclusion, we showed that it is possible to reproduce the characteristics of a financial bubble using an analytical model of the stock market. As a result, it seems feasible that financially unfavorable situations can be modeled analytically. This is a terrific result as it clearly opens the door for further research into this topic.

Taking into consideration all which has been imparted by this project, there appears to be enormous potential for analytical models of financial markets with regard to managing financially unfavorable situations. We believe that if there is enough investment in research, the financial services industry may soon put analytical models to use in order to make better decisions and ultimately circumvent financially unfavorable situations.

6 Appendix

6.1 Default Parameters

```
- SIMULATION MODUS -
                    Empty book (On/Off) -        bkempty:  1
                  Entry refresh (On/Off) -     entrefresh:  0
       Entry erasure when aged (On/Off) -         entage:  1
              Multiple shares (On/Off) -      mulshares:  1
          Volatility feedback (On/Off) -        volfeed:  0
             Price regulation (On/Off) -       regulate:  0
             Financial bubble (On/Off) -          devon:  0


- STANDARD PARAMETERS (used in all of the modi)
                      Number of traders -           tnum:  100
                       Amount of shares -      totShares:  100000
                         Starting price -             p0:  100.00

            Mean of normal distribution -             mu:  1.0000
Initial std. deviation of normal dist. -          sigma:  0.0050
 Parameter of exponential distribution -         lambda:  20.0000

                              Total days -              M:  10
                   Total time in seconds -              T:  25200

              Last tick iteration window -             dt:  60.00


- VOLATILITY FEEDBACK PARAMETERS -
                   Sigma constant factor -              k:  1.68
             Buyer - Seller correlation -         korrbs:  1.00
           Old - New sigma correlation -          korrk:  0.000400
                    Maximum age of entry -             a0:  600.00


- PRICE REGULATION PARAMETERS -
                                 Growth -         growth:  15.00
                             Price floor -             pF:  100.00
                           Price ceiling -             pC:  110.00


- FINANCIAL BUBBLE PARAMETERS -
          Stability mean of norm. dist. -            mu3:  1.00
                    Bubble starting day -             t1:  1
           Increase mean of norm. dist. -            mu1:  1.00
                    Day of bubble burst -             t2:  7
           Decrease mean of norm. dist. -            mu2:  0.98
           Day of stability after burst -             t3:  9
```

6.2 Source Code

code/

- code/control.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


clear all; clf; close all; clc;


%% Get path information
pathfile = fopen('path.txt','r');

frewind(pathfile);

mf = fscanf(pathfile, '%s', 1 );
% main folder
sf = fscanf(pathfile, '%s', 1 );
% simulation folder
gsin = fscanf(pathfile, '%s', 1 );
% global simulation info file
comp = fscanf(pathfile, '%s', 1 );
% compare folder name
simf = fscanf(pathfile, '%s', 1 );
% sim folder name
spa = fscanf(pathfile, '%s', 1 );
% sim parameter file
sin = fscanf(pathfile, '%s', 1 );
% sim info file
srep = fscanf(pathfile, '%s', 1 );
% sim report file
gsrep = fscanf(pathfile, '%s', 1 );
% global simulation report
plt = fscanf(pathfile, '%s', 1 );
% sim plot files (more than one)
parfold = fscanf(pathfile, '%s', 1 );
% parameter sweep instruction folder
spr = fscanf(pathfile, '%s', 1 );
% parameter sweep file
stru = fscanf(pathfile, '%s', 1 );
% tree structure file


fclose(pathfile);

newfolder = 0;
% create new folder


%% Create input
cd('Input');
    [simpath, simnum, amount] = foldermg( mf, sf, gsin, comp, simf, spa,
sin, newfolder, parfold, spr, stru );
```

```
cd('../');


%% Load input & simulate
startdate = date;
starttime = clock;
ticID2 = tic;

i = 0;
%digit = floor(log10(amount-1))+1;
% amount of digits needed for numerotation
%simustr = sprintf('%0*.0f', digit, i);
paramfile = sprintf( '%s/%s/%s', simpath, simf, spa );
tparamfile = regexprep( paramfile, '**', num2str(i) );

fs = 0;
failsim = zeros( 1, 1000 );
while ( exist(tparamfile, 'file' ) ~= 0 )


    try
% in case of error, the global simulation

% continues but the subsimulation stops

% and the error is reported
        ticID1 = tic;
% start timer

        %% Start simulation
        load( tparamfile );

        cd( 'Computation/' );

            [ SSM, SP, SPL ] = main( SSM, SP, SPL );

        cd( '../' );


        %% Plot section
        cd( 'Plot/' );

            plotpath = sprintf( '%s/%s/%s', simpath, simf, plt );
            plotpath = regexprep( plotpath, '**', num2str(i) );
            plotpath = regexprep( plotpath, 'plot', '' );

            comparepath = sprintf( '%s/%s/%s', simpath, comp, plt );
            comparepath = regexprep( comparepath, '**', num2str(i) );
            comparepath = regexprep( comparepath, 'plot', '' );

            plotCall( SSM, SP, SPL, plotpath, comparepath );

        cd( '../' );


        rstat = 'Simulation was succesful!';
        errmsg = '-';
```

```
    catch ME

        fs = fs + 1;
        failsim( fs, : ) = i;
        rstat = 'Simulation failed!';
        errmsg = ME.message;

        cd( '../' );

    end


    ctime = toc(ticID1);

    %% Create report file
    reportfilename = sprintf( '%s/%s/%s', simpath, simf, srep );
    reportfilename = regexprep( reportfilename, '**', num2str(i) );
    file = fopen( reportfilename, 'w' );

    % overhead
    dstr = date;
    cstr = clock;
    fprintf(file, '©2012 ETH Zuerich\nBambolei\nNicholas Eyring, Youri
Popoff\nSimulation of trading in an artificial stock market\n\n\n\n');
    fprintf(file, '                                          **** SIMULATION %u
****\n\n', i);
    fprintf(file, '                                                Report
file\n\n');

    fprintf(file, '
Date:  %s\n',    dstr);
    fprintf(file, '
Time:  %02.0f:%02.0f:%02.0f\n\n', cstr(4), cstr(5), cstr(6));

    % text
    fprintf(file, '                    Computation duration -
ctime:  %f\n',    ctime);
    fprintf(file, '                            Report status -
rstat:  %s\n',    rstat);
    fprintf(file, '                            Error message -
errmsg:  %s\n\n',  errmsg);

    fclose(file);


    i = i + 1;
    tparamfile = regexprep( paramfile, '**', num2str(i) );


end

enddate = date;
endtime = clock;

dur = toc(ticID2);

%% Global simulation report
grepfilename = sprintf('%s/%s', mf, gsrep);
grepfilename = regexprep(grepfilename, '**', num2str(simnum));
```

```
rep = fopen(grepfilename, 'w');

% overhead
dstr = date;
cstr = clock;
fprintf(rep, '©2012 ETH Zuerich\nBambolei\nNicholas Eyring, Youri
Popoff\nSimulation of trading in an artificial stock market\n\n\n\n');
fprintf(rep, '                                          **** Global SIMULATION
%u ****\n\n', simnum);
fprintf(rep, '                                                Report
file\n\n');

fprintf(rep, '                                                        Date:
%s\n',     dstr);
fprintf(rep, '                                                        Time:
%02.0f:%02.0f:%02.0f\n\n', cstr(4), cstr(5), cstr(6));

% text
fprintf(rep, '           Amount of failed simulations -             fs:
%u\n\n',   fs);

fprintf(rep, '                             Starting date -      startdate:
%s\n',     startdate);
fprintf(rep, '                              Sarting time -      starttime:
%02.0f:%02.0f:%02.0f\n\n', starttime(4), starttime(5), starttime(6));

fprintf(rep, '                               Ending date -        enddate:
%s\n',     enddate);
fprintf(rep, '                               Ending time -        endtime:
%02.0f:%02.0f:%02.0f\n\n', endtime(4), endtime(5), endtime(6));

fprintf(rep, '                                  Duration -            dur:
%f\n\n\n', dur);

% Failed simulation index list
if fs > 0

fprintf(rep, '                                                Index of failed
simulations\n\n');

end

for n = 1:1:fs

fprintf(rep, '                                          - %u\n', failsim(n));


end



fclose(rep);
```

- code/path.txt

```
Data
Simulation**
globalsim**info.txt
Compare
Sim**
sim**parameter.mat
sim**info.txt
sim**report.txt
globalsim**report.txt
sim**plot
Input
param.txt
structure.txt
```

code/Computation

- code/Computation/ageCheckBuyer.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = ageCheckBuyer( SSM, SP, m, t )
%AGECHECKBUYER checks if the entry in the book is older than a0 (lifespan)
%   If aged: trader makes new offer


    %% Find auction to be refreshed
    pagedind = find( SSM.bookbpaging( [1:1:SSM.sbbp], 5 ) <= 0 );


    if isempty(pagedind) ~= 1 && SP.entage == 1
```

```
        lgt = length( pagedind );

        if lgt > 1
% queue of traders who want to refresh auction

            warning('W05 Buyer: Number of entries to be renewed: %d', lgt);

        end


        %just refresh the first element
        %pagedind = pagedind(1);


        %% Get data from bid to be refreshed
        %   Book Format:
        %   day, time, seller/buyer id, s/b price, shares, dirty bit, age
bit, new
        %   entry number, index of aged entry

        for i = 1:1:lgt

% loop for each auction to be refreshed

            %orind = SS.bookbpaging( i, 4 );
% get original auction index
            orind = SSM.bookbpaging( pagedind(i), 4 );
% get original auction index

            auction = SSM.bookb( orind, : );
            ind = auction(3);
% index of the chosen trader
            arefresh = auction(8) + 1;
% amount of refresh

            SSM.bookb( orind, [6, 7] ) = [0, 1]';
% old entry invalid

% old entry aged
            SSM.bookbpaging( pagedind(i), : ) = [];
% erase old entry in paging matix
            SSM.sbbp = SSM.sbbp - 1;
            pagedind = pagedind(:,1) -1;
% update index of entries to renew (shifted due

% to actual renewal)


            if SP.entrefresh == 1

                [ SSM ] = buyer( SSM, SP, m, t, ind, arefresh, orind );

            end

        end
```

```
    end

end
```

- code/Computation/ageCheckSeller.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = ageCheckSeller( SSM, SP, m, t )
%AGECHECKSELLER checks if the entry in the book is older than a0 (lifespan)
%   If aged: trader makes new offer


    %% Find bid to be refreshed
    pagedind = find( SSM.bookspaging( [1:1:SSM.sbsp], 5 ) <= 0 );


    if isempty(pagedind) ~= 1 && SP.entage == 1

        lgt = length( pagedind );

        if lgt > 1
% queue of traders who want to refresh auction

            warning('W06 Seller: Number of entries to be renewed: %d',
lgt);

        end


        %just refresh the first element
        %pagedind = pagedind(1);


        %% Get data from bid to be refreshed
        %   Book Format:
        %   day, time, seller/buyer id, s/b price, shares, dirty bit, age
bit, new
        %   entry number, index of aged entry

        for i = 1:1:lgt

% loop for each auction to be refreshed

            %orind = SS.bookspaging( i, 4 );
% get original auction index
```

```
            orind = SSM.bookspaging( pagedind(i), 4 );
% get original auction index

            auction = SSM.books( orind, : );
            ind = auction(3);
% index of the chosen trader
            arefresh = auction(8) + 1;
% amount of refresh

            SSM.books( orind, [6, 7] ) = [0, 1]';
% old entry invalid

% old entry aged
            SSM.bookspaging( pagedind(i), : ) = [];
% erase old entry in paging matix
            SSM.sbsp = SSM.sbsp - 1;
            pagedind = pagedind(:,1) -1;
% update index of entries to renew (shifted due

% to actual renewal)


            if SP.entrefresh == 1

                [ SSM ] = seller( SSM, SP, m, t, ind, arefresh, orind );

            end

        end


    end

end
```

- code/Computation/ageUpdate.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ bookpaging ] = ageUpdate( bookpaging, sb )
%AGEUPDATE updates age of bid

    if sb > 0

        bookpaging( 1:1:sb, 5 ) = bookpaging( 1:1:sb, 5 ) - ones( sb, 1 );
%maximal age column minus 1 in paging matrix
```

```
    end

end
```

- code/Computation/buyer.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = buyer( SSM, SP, m, t, ind, arefresh, orind )
%BUYER Completes the tasks of the buyer (stat = 0)
%   Calculates the price of the bid and executes the transaction if there
%   is a price overlap


        if SP.volfeed == 1

            [ SSM ] = volatilityFeedback( SSM, SP, ind );
% account for past market volatility

        end

        n = normrnd(SP.mu, SSM.sigma, 1, 1);
% factor ni used to calculate price

        SSM.sd3 = SSM.sd3 + 1;
        SSM.debug( SSM.sd3, 3 ) = n;

        %if arefresh ~= 0

         %   n = SS.mu + abs( SS.mu - n );
% more interesting price: higher for buyer

        %end
```

```
        %% Compute new price in function of old price
        if SSM.sbbp ~= 0 && SSM.sbsp ~= 0

            d = SP.korrbs * SSM.bookbpaging(1, 1) + ( 1 - SP.korrbs ) *
SSM.bookspaging(1, 1);

% take actual best bid price in book d(th-1)
            p = d * n;

        else

            % book empty but already transactions
            if SSM.sbp ~= 0

                %p = SS.tprice(SS.sbp, 1) * n;
% if book empty, take last best price
                p = SSM.d * n;

            else

                p = SP.p0;

            end

            warning('W03 Buyer book empty!');

        end


        %% Price regulation 1
        %   variations of more than varwidth are not allowed

        varwidth = 0.06*SSM.sigma/0.005;

        if SSM.sbp ~= 0

            tpr = SSM.tprice( SSM.sbp, 1 );

        else

            tpr = SP.p0;

        end

        highlim = ( 1 + varwidth ) * tpr;
        lowlim = ( 1 - varwidth ) * tpr;

        if p < lowlim || p > highlim

            p = tpr;

        end

        %% Price regulation 2
        %   regulates price with ceiling/floor if applicable

        if SP.regulate == 1
```

```
            p = regulatePrice(p, SP, t);
% regulate price if applicable

        end

        %% Transaction section
        %   Compute amount of shares (buy) & write entry in book
        maxShares = floor((SSM.treg(ind,1))/p);
% maximum number of shares trader can buy

        if maxShares > 0 && p > 0
% otherwise no order

            %% Maximum & minimum price ever occurred
            if p > SSM.pmax

                SSM.pmax = p;

            end

            if p < SSM.pmin

                SSM.pmin = p;

            end


            %% Shares
            if SP.mulshares == 1

                shares = randi(maxShares);
% random fraction of maximum

            else

                shares = 1;

            end


            SSM.sbb = SSM.sbb + 1;
% increment number of elements in buy order book
            SSM.bookb( SSM.sbb, : ) = [ m, t, ind, p, shares, 1, 0,
arefresh, orind ];

% new entry in buyer book

            [ SSM ] = buyerTransaction( SSM, SP, ind, shares, p, SP.a0, t
);

% execute the order if possible

        end


end
```

- code/Computation/buyerTransaction.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock mark

function [ SSM ] = buyerTransaction( SSM, SP, ind, shares, p, a0, t )
%BUYERTRANSACTION Checks if a transaction should occur when a new buyer
%   entry is created and carries it out if need be (eventually with
%   multiple sellers)

    if SSM.sbsp > 0

        %% Information fetch
        priceS = SSM.bookspaging(1,1);
% mimimum sell order unit price

        pCounter = 1;
% incremented when buyerid = sellerid

% the seller is then skipped

        %% Transaction loop
        %   (until no more money or no more interesting auction)
        while( p >= priceS && SSM.sbsp >= pCounter )
% buyerTransaction -> price == asking price !


            priceS = SSM.bookspaging(pCounter,1);
% new mimimum sell order unit price
            EntryIndexS = SSM.bookspaging(pCounter,4);
% index of entry in books
            sellerID = SSM.books(EntryIndexS,3);
% ID of the seller
```

```
            %% Both involved traders distinct
            if sellerID ~= ind

                sharesS = SSM.bookspaging(pCounter,3);
% desired number of shares for seller


                %% Shares control
                %   seller wants to sell shares but may have already sold
                %   some shares due to another transaction

                if sharesS > SSM.treg(sellerID,2)

                    sharesS = SSM.treg(sellerID,2);
% negative shares not allowed ! -> set to maximum allowed
                    warning('W02 Seller lacks shares');

                end


                trshares = min(shares, sharesS);
% amount of shares involved in transaction
                trliq = priceS * trshares;
% cash to debit / credit


                %% Holdings update
                SSM.treg(sellerID,1) = SSM.treg(sellerID,1) + trliq;
% update seller's account
                SSM.treg(sellerID,2) = SSM.treg(sellerID,2) - trshares;
% udpate seller's share holdings

                SSM.treg(ind,1) = SSM.treg(ind,1) - trliq;
% update buyer's account
                SSM.treg(ind,2) = SSM.treg(ind,2) + trshares;
% update buyer's share holdings


                %% Seller book & paging seller book update

                % seller not sold all
                if sharesS > trshares

                    SSM.bookspaging(pCounter,3) = sharesS - trshares;
% update seller order shares

                else

                    SSM.bookspaging(pCounter,:) = [];
% delete seller entry from paging book
                    SSM.sbsp = SSM.sbsp - 1;
                    SSM.books(EntryIndexS,6) = 0;
% set dirty-bit to 0 in books

                end
```

```
                %% Buyer book update
                shares = shares - trshares;
% buyer remaining shares

                % buyer bought all
                if shares == 0

                    SSM.bookb(SSM.sbb, 6) = 0;
% set valid bit to 0 in bookb
                    break;

                end


                %% New entry in tprice
                SSM.sbp = SSM.sbp + 1;
% increment number of entries in tprice
                SSM.tprice(SSM.sbp,:) = [priceS, sharesS, sellerID,
EntryIndexS, ind, SSM.sbb, t];


            else

                pCounter = pCounter + 1;
% skip trader next loop

            end


        end
% end of while loop

    end


    %% Buyer still desires shares
    %   - A new auction appears in the bookb
    %   - If the buyer has already bought all of the desired shares,
    %     then there is no new entry in the bookb paging
    if shares > 0

        SSM.sbbp = SSM.sbbp + 1;
% increment number of elements in paging book
        SSM.bookbpaging( SSM.sbbp, : ) = [ p, t, shares, SSM.sbb, a0 ];
% new entry in the paging book
        SSM = sortBookb( SSM );
% sort the paging book

    end

end
```

- code/Computation/shiftMean.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market
```

```
function [ SP, m ] = shiftMean( SP, m )
%SHIFTMEAN changes the mean of the price

    if SP.devon == 1

        if m > SP.t1 && m < SP.t2

            SP.mu = SP.mu1;

        elseif m > SP.t2 && m < SP.t3

            SP.mu = SP.mu2;

        else

            SP.mu = SP.mu3;

        end

    end


end
```

- code/Computation/emptyBook.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market



function [ SSM ] = emptyBook( SSM, SP )
%EMPTYBOOK empties book if needed


    if SP.bkempty == 1

        SSM.bookbpaging(1:1:SSM.sbbp, :) = [];
% book emptying
        SSM.sbbp = 0;
        SSM.bookspaging(1:1:SSM.sbsp, :) = [];
        SSM.sbsp = 0;

        SSM.bookb( 1:1:SSM.sbb, 6 ) = zeros( SSM.sbb, 1 );
% valid bit to zero
        SSM.books( 1:1:SSM.sbs, 6 ) = zeros( SSM.sbs, 1 );
```

```
    end


end
```

- code/Computation/logReturns.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = logReturns( SSM, SP )
%LOGRETURNS computes log returns each dt = 60s
%   returns logreturns in the SSM.ret matrix


%Matrix format:
%       Column 1            Column 2        (Time intervall)
% 1     price p. tick       log-ret         1   -  60s
% 2     price p. tick       log-ret         61  - 120s
% 3     price p. tick       log-ret         121 - 180s
% ...


    tindex = SSM.sbp;

    SSM.retsize = SSM.retsize + 1;
% increment retsize


    %% Find last tick (previous-tick interpolation)
    if tindex > 0
% we have transaction values

        SSM.ret(SSM.retsize,1) = SSM.tprice( tindex, 1 );
% write tick price into matrix


        %% Compute log-return
        % in percent

        if SSM.retsize > 1
```

```
            SSM.ret(SSM.retsize, 2) = 100 * log(SSM.ret(SSM.retsize,1) /
SSM.ret(SSM.retsize - 1,1) );

        else

            SSM.ret(SSM.retsize, 2) = 0;
% set return to zero if we don't have enough entries

        end


    else

        SSM.ret(SSM.retsize, 1) = SP.p0;
% use initial price if no transactions have occured (previous tick ?)
        SSM.ret(SSM.retsize, 2) = 0;
% set return to zero (nothing has changed yet!)

    end



end
```

- code/Computation/main.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM, SP, SPL ] = main( SSM, SP, SPL )
%MAIN Determines trading period and calls buyer or seller function to
create
%   entries in the Stock Market book


    % Liveplot
```

```
    if SPL.liveplot == 1

        cd('../Plot');

            liveplot;

        cd('../Computation');

    end


    % Debug coefficient: time evolution
    coeff = 1/(SP.M*SP.T)*100;
    pe = -1;

    % Function scope
    m = 0;

    %tic;

    %% Initialisation
    s = RandStream('mt19937ar','Seed',0);
% choosing & resetting random stream
    %RandStream.setDefaultStream(s);
    RandStream.setGlobalStream(s);

    t = 1+round(exprnd(SP.lambda));
% time of first entry
    lrt = SP.dt;
% used for log returns calculation intervals


    %% Simulation section
    for i = 1:1:(SP.M)*(SP.T)


        %% Shift the mean
        SP = shiftMean( SP, m );


        %% Age Update
        SSM.bookbpaging = ageUpdate( SSM.bookbpaging, SSM.sbbp );
        SSM.bookspaging = ageUpdate( SSM.bookspaging, SSM.sbsp );

        %% Calculate Log Returns
        if i == lrt

            [ SSM ] = logReturns( SSM, SP );
% calculate log returns

            lrt = lrt + SP.dt;
% increment lrt for next log returns calculation

        end


        %% New Book Entry Section
        if i == t
```

```
            Tau = 1+round(exprnd(SP.lambda));
% step between two book entries

% (random number; exponential distribution

            SSM.st = SSM.st + 1;
% saving event occurrence
            SSM.tocc( SSM.st, 1 ) = Tau;


            t = t + Tau;
            if t - m * SP.T > SP.T
% t - m * T = actual time in the trading day m


                m = m + 1;
% increment number of days past

                [ SSM ] = emptyBook( SSM, SP );


            end



            %% Book entry
            ind = randi(SP.tnum, 1, 1);
% index of the chosen trader
            stat = randi(2, 1, 1) - 1;
% choose between buyer (0) or seller (1)

            arefresh = 0;
% bit to tell whether the entry is new or refreshed
            orind = 0;
% new auction: no aged entry line

            if stat == 0
% we have a buy order (0)

                [ SSM ] = buyer(SSM, SP, m, i, ind, arefresh, orind);

            else
% we have a sell order (1)

                [ SSM ] = seller(SSM, SP, m, i, ind, arefresh, orind);

            end


        end


        %% Age Check
        [ SSM ] = ageCheckBuyer(SSM, SP, m, i);
        [ SSM ] = ageCheckSeller(SSM, SP, m, i);
```

```
        [ SSM ] = weightedTP( SSM, i );
% update weighted transaction price matrix


        %% Plot section live
        if SPL.liveplot == 1

            cd('../Plot');
            plotPrice( i, SSM, SP, SPL, fig1 );
            cd('../Computation');

        end


        %% Percentage (evolution)
        npe = floor(coeff * i);
        if npe > pe

            pe = pe + 1;
            disp(['Completed to ', num2str(pe), ' percents!']);

        end


    end


    %% Control
    if sum(SSM.bookb(:,6)) ~= SSM.sbbp

        warning('W07 Buyer bookb & bookbpaging do not coincide!');

    end
    if sum(SSM.books(:,6)) ~= SSM.sbsp

        warning('W08 Seller books & bookspaging do not coincide!');

    end



    %ctime = toc;

end
```

- code/Computation/regulatePrice.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market

function [ p ] = regulatePrice( p, SP, t )
% Returns regulated price p using given price
%   regulation parameters

    pCeiling = SP.pC + (SP.growth*SP.pC/(100*24*60*60))*t;
% calculate current price ceiling
    pFloor = SP.pF + (SP.growth*SP.pF/(100*24*60*60))*t;
% calculate current price floor

    if p > pCeiling

        p = pCeiling;           % set price to nearest legal limit

    else
        if p < pFloor

            p = pFloor;         % set price to nearest legal limit

        end

    end

end
```

- code/Computation/seller.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = seller( SSM, SP, m, t, ind, arefresh, orind )
%SELLER Completes the tasks of the seller (stat = 1)
%   Calculates the asking price and executes the transaction if there if a
%   price overlap


        if SP.volfeed == 1

            [ SSM ] = volatilityFeedback( SSM, SP, ind );
% account for past market volatility

        end

        n = normrnd(SP.mu, SSM.sigma, 1, 1);
% factor ni to calculate the price

        SSM.sd3 = SSM.sd3 + 1;
        SSM.debug( SSM.sd3, 3 ) = n;

        %if arefresh ~= 0

         %   n = SS.mu - abs( SS.mu - n );
% more interesting price: lower for seller

        %end

        %% New price in function of old price
        if SSM.sbsp ~= 0 && SSM.sbbp ~= 0

            a = SP.korrbs * SSM.bookspaging(1, 1) + ( 1 - SP.korrbs ) *
SSM.bookbpaging(1, 1);

% take actual best ask price in book a(th-1)
            p = a * n;

        else

            % book empty but already transactions
            if SSM.sbp ~= 0

                %p = SS.tprice(SS.sbp, 1) * n;
% if book empty, take last best price
```

```
                p = SSM.a * n;

            else

                p = SP.p0;

            end

            warning('W04 Seller book empty!');

        end

        %% Price regulation 1
        %   variations of more than varwidth are not allowed

        varwidth = 0.06*SSM.sigma/0.005;

        if SSM.sbp ~= 0

            tpr = SSM.tprice( SSM.sbp, 1 );

        else

            tpr = SP.p0;

        end

        highlim = ( 1 + varwidth ) * tpr;
        lowlim = ( 1 - varwidth ) * tpr;


        if p < lowlim || p > highlim

            p = tpr;

        end

        %% Price regulation 2
        %   regulates price with ceiling/floor if applicable

        if SP.regulate == 1

            p = regulatePrice(p, SP, t);
% regulate price if applicable

        end

        %% Transaction section
        %   Compute amount of shares (sell) & write entry in book
        if SSM.treg(ind,2) > 0 && p > 0
% otherwise no order

            %% Maximum & minimum price ever occurred
            if p > SSM.pmax

                SSM.pmax = p;
```

```
            end

            if p < SSM.pmin

                SSM.pmin = p;

            end

            %% Shares
            if SP.mulshares == 1

                shares = randi(SSM.treg(ind,2));
% random fraction of quantity of stocks owned by trader

            else

                shares = 1;

            end


            SSM.sbs = SSM.sbs + 1;
% increment number of elements in seller order book
            SSM.books( SSM.sbs, : ) = [ m, t, ind, p, shares, 1, 0,
arefresh, orind ];

% new entry in seller book

            [ SSM ] = sellerTransaction( SSM, SP, ind, shares, p, SP.a0, t
);

% execute the order if possible


        end


end
```

- code/Computation/sellerTransaction.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock mark

function [ SSM ] = sellerTransaction( SSM, SP, ind, shares, p, a0, t )
%SELLERTRANSACTION   Checks if a transaction should occur when a new seller
%   entry is created and carries it out if need be (eventually with
%   multiple buyers)

    if SSM.sbbp > 0

        %% Information fetch
        priceB = SSM.bookbpaging(1,1);
% maximum buy order unit price

        pCounter = 1;
% used if buyer index == seller index


        %% Transaction loop
        %   (until buyer no more money or price too high)
        while( p <= priceB && SSM.sbbp >= pCounter )
% sellerTransaction -> price == bid price !

            priceB = SSM.bookbpaging(pCounter,1);
% new maximum buy order unit price
            EntryIndexB = SSM.bookbpaging(pCounter,4);
% index of entry in bookb
            buyerID = SSM.bookb(EntryIndexB,3);
% ID of the buyer


            %% Traders distinct
            if buyerID ~= ind

                sharesB = SSM.bookbpaging(pCounter,3);
% desired number of shares for buyer


                %% Shares control
                %   buyer desires to buy shares but may already have given
                %   away his money due to a newer auction
                maxBshares = floor((SSM.treg(buyerID,1))/priceB);
% maximum number of shares buyer can buy

                if maxBshares < sharesB

                    sharesB = maxBshares;
% negative liquidities not allowed ! -> set to maximum allowed
                    warning('W01 Buyer lacks liquidities');
```

```
                end


                trshares = min(shares, sharesB);
% amount of maximum shares involved in transaction
                trliq = priceB * trshares;
% cash involved

                %% Holdings update
                SSM.treg(buyerID,1) = SSM.treg(buyerID,1) - trliq;
% update buyer's account
                SSM.treg(buyerID,2) = SSM.treg(buyerID,2) + trshares;
% udpate buyer's share holdings

                SSM.treg(ind,1) = SSM.treg(ind,1) + trliq;
% update seller's account
                SSM.treg(ind,2) = SSM.treg(ind,2) - trshares;
% update seller's share holdings

                %% Buyer book & paging buyer book update

                % buyer not bought all
                if sharesB > trshares

                    SSM.bookbpaging(pCounter,3) = sharesB - trshares;
% update buyer order shares

                else

                    SSM.bookbpaging(pCounter,:) = [];
% delete buyer entry from paging book
                    SSM.sbbp = SSM.sbbp - 1;
% decrement number of entries in bookbpaging
                    SSM.bookb(EntryIndexB,6) = 0;
% set dirty-bit to 0 in bookb

                end


                %% Seller book update
                shares = shares - trshares;
% remaining seller shares (cannot be negative)

                if shares == 0

                    SSM.books(SSM.sbs, 6) = 0;
% set valid bit to 0 in books
                    break;

                end


                %% New entry in tprice
                SSM.sbp = SSM.sbp + 1;
% increment number of entries in tprice
                SSM.tprice(SSM.sbp,:) = [priceB, sharesB, ind, SSM.sbs,
buyerID, EntryIndexB, t];
```

```
            else

                pCounter = pCounter + 1;
% skip trader next loop

            end


        end

    end


    %% Seller still desires to sell shares
    %   - New auction in books
    %   - If all the shares were sold, then no new entry in seller paging
    %     book
    if shares > 0

        SSM.sbsp = SSM.sbsp + 1;
% increment number of elements in paging book
        SSM.bookspaging( SSM.sbsp, : ) = [ p, t, shares, SSM.sbs, a0 ];
% new entry in the paging book
        SSM = sortBooks( SSM );
% sort the paging book

    end

end
```

- code/Computation/sortBookb.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = sortBookb( SSM )
%SORTBOOKB sorts the paging book (buyer)
%   - The highest price is i the first row
%   - The sorting is in decreasing order
%   - If 2 or more entries have the same price, the oldest one comes first
%   (increasing order)

    A = SSM.bookbpaging( 1:1:SSM.sbbp, : );
    sortedpart = sortrows( A, [ -1 2 ] );
% 1. sort: decreasing for price

% 2. sort: increasing for time
    SSM.bookbpaging( 1:1:SSM.sbbp, : ) = sortedpart;

    if SSM.sbbp ~= 0

        SSM.d = SSM.bookbpaging(1, 1);
```

```
    end

end
```

- code/Computation/sortBooks.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM ] = sortBooks( SSM )
%SORTBOOKS sorts the paging book (seller)
%    - The lowest price is i the first row
%    - The sorting is in increasing order
%    - If 2 or more entries have the same price, the oldest one comes first
%    (increasing order)

    A = SSM.bookspaging( 1:1:SSM.sbsp, : );
    sortedpart = sortrows( A, [ 1 2 ] );
% 1. sort: increasing for price

% 2. sort: increasing for time
    SSM.bookspaging( 1:1:SSM.sbsp, : ) = sortedpart;

    if SSM.sbsp ~= 0

        SSM.a = SSM.bookspaging(1, 1);

    end

end
```

- code/Computation/volatilityFeedback.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market

function [ SSM ] = volatilityFeedback( SSM, SP, ind )
%   updates sigma value depending on past market volatility

    SSM.sd4 = SSM.sd4 + 1;

    %% Time window
    %   Randomly initialised for each trader
    %T = 599 + randi(5401);
% determine time window
    T = SSM.treg( ind, 3 );

    baseIndex = SSM.retsize - floor(T/SP.dt);
% calulate base index

    if baseIndex < 1

        baseIndex = 1;
% take entire history into account

    end

    if SSM.retsize > 0
% we can only affect sigma once log returns are available

        logRet = SSM.ret(baseIndex:SSM.retsize,2)./100;
% extract wanted vector from ret matrix

        sigmaT = std(logRet);
% determine the standard deviation of log returns in period T
        sigmaN = SP.k * sigmaT;
% sigmaN is a function of sigmaT


        SSM.sigma = SP.korrk * sigmaN + ( 1 - SP.korrk ) * SSM.sigma;
% set new value for sigma

        %% Evolution of sigma
        SSM.se = SSM.se + 1;
        SSM.sige( SSM.se, 1 ) = SSM.sigma;


    end

end
```

- code/Computation/weightedTP.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock mark

function [ SSM ] = weightedTP( SSM, i )
%WEIGHTEDTP Updates the weighted average transaction price matrix used for
plotting

   index = SSM.sbp;
% get number of elements in tprice

   if index > 0

         lastTtime = SSM.tprice(index,7);
% get the most recent transaction time

         if lastTtime == i
% only needs updating if a transaction

% has been carried out during the current loop

            % initialize totals
            totalV = 0;
% total amount of money involved in transactions

% at time i
            totalS = 0;
% total amount of shares in transactions

            while( index > 0 && SSM.tprice(index,7) == i)
```

```
                totalS = totalS + SSM.tprice(index,2);
                totalV = totalV + SSM.tprice(index,1)*SSM.tprice(index,2);

                index = index - 1;
% decrement index

            end

            avgP = totalV/totalS;
% calculate weighted average

            SSM.savtp = SSM.savtp + 1;
% increment number of elements in avgtprice
            SSM.avgtprice(SSM.savtp,:) = [avgP, i];
% add element to avgtprice

        end

    end

end
```

code/Input

- code/defaultInfo.txt

```
©2012 ETH Zuerich
Bambolei
Nicholas Eyring, Youri Popoff
Simulation of trading in an artificial stock market



                                    **** Default parameter values ****

                                            Information file

                                                      Date:  13-Dec-2012
                                                      Time:  22:03:18

- SIMULATION MODUS -
                    Empty book (On/Off) -        bkempty:  1
                 Entry refresh (On/Off) -     entrefresh:  0
       Entry erasure when aged (On/Off) -         entage:  1
              Multiple shares (On/Off) -      mulshares:  1
          Volatility feedback (On/Off) -        volfeed:  0
             Price regulation (On/Off) -       regulate:  0
             Financial bubble (On/Off) -          devon:  0


- STANDARD PARAMETERS (used in all of the modi)
                      Number of traders -           tnum:  100
                       Amount of shares -      totShares:  100000
                         Starting price -             p0:  100.00

            Mean of normal distribution -             mu:  1.0000
Initial std. deviation of normal dist. -          sigma:  0.0050
 Parameter of exponential distribution -         lambda:  20.0000

                              Total days -              M:  10
                  Total time in seconds -              T:  25200

              Last tick iteration window -             dt:  60.00


- VOLATILITY FEEDBACK PARAMETERS -
                  Sigma constant factor -              k:  1.68
              Buyer - Seller correlation -         korrbs:  1.00
            Old - New sigma correlation -          korrk:  0.000400
                   Maximum age of entry -             a0:  600.00


- PRICE REGULATION PARAMETERS -
                                 Growth -         growth:  15.00
                            Price floor -             pF:  100.00
                          Price ceiling -             pC:  110.00


- FINANCIAL BUBBLE PARAMETERS -
          Stability mean of norm. dist. -            mu3:  1.00
                    Bubble starting day -             t1:  1
           Increase mean of norm. dist. -            mu1:  1.00
```

```
                Day of bubble burst -            t2:  7
        Decrease mean of norm. dist. -           mu2:  0.98
        Day of stability after burst -            t3:  9
```

- code/Input/foldermg.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ simpath, simnum, amount ] = foldermg( mainfolder,
simulationfolder, simulationinfo, comp, simfold, sparam, sinfo, newfolder,
parfold, spr, stru )
%FOLDERMG Creates simulation file structure
%   Returns actual simualtion folder and amount of sim to execute


    cd('../');
% change to main folder


    %% Remove old folder if existing
    if ( exist(mainfolder,'dir') == 7 )

        warning('Folder already existing');

        if newfolder == 1

            rmdir(mainfolder,'s');

        end

    else

        mkdir(mainfolder);

    end


    %% Simulation folder
    i = 0;
```

```
    prefix = sprintf('%s/%s', mainfolder, simulationfolder);
    tprefix = regexprep(prefix, '**', num2str(i));

    while ( exist(tprefix, 'dir') == 7 )

        i = i + 1;
        tprefix = regexprep(prefix, '**', num2str(i));

    end

    mkdir(tprefix);
    simpath = tprefix;
    simnum = i;


    %% Create compare folder
    cf = sprintf( '%s/%s', tprefix, comp );
    mkdir(cf);


    %% Simulation file

    cd( 'Input/' );

        [SSM, SP, SPL] = init();
% call standard parameters
        [ amount ] = paramSweepCall( SSM, SP, SPL, simpath, simfold,
sparam, sinfo, parfold, spr, stru );

    cd( '../' );

    %% Create info file
    infofilename = sprintf('%s/%s', mainfolder, simulationinfo);
    infofilename = regexprep(infofilename, '**', num2str(i));
    simfile = fopen(infofilename, 'w');

        % overhead
        dstr = date;
        cstr = clock;
        fprintf(simfile, '©2012 ETH Zuerich\nBambolei\nNicholas Eyring,
Youri Popoff\nSimulation of trading in an artificial stock
market\n\n\n\n');
        fprintf(simfile, '                                        **** Global
SIMULATION %u ****\n\n', simnum);
        fprintf(simfile, '                                                Info
file\n\n');

        fprintf(simfile, '
Date:  %s\n',            dstr);
        fprintf(simfile, '
Time:  %02.0f:%02.0f:%02.0f\n\n', cstr(4), cstr(5), cstr(6));

        fprintf(simfile, '                                        Number of
simulations:  %u\n',           amount);
        fprintf(simfile, '
Simulation info:  %s/%s/%s\n',      tprefix, simfold, sinfo);
        fprintf(simfile, '                                        Simulation
parameters:  %s/%s/%s\n\n',    tprefix, simfold, sparam);
```

```
        fclose(simfile);


    cd( 'Input/' );
% go back to function folder


end
```

- code/Input/init.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SSM, SP, SPL ] = init( )
%INIT Summary of this function goes here
%   Detailed explanation goes here


    %% Initialisation
    %   Contains all initial parameters
    %   Creates trader matrix & empty book matrixes

    bkempty = 1;
% book emptying parameter

% 0: Off, 1: On

    lambda = 20;
% mean of the exponential distribution
    mu = 1;
% deviation
    sigma = 0.005;
% mean of the gaussian (normal) distribution
```

```
    M = 10;
% number of days
    %M = 10;
    m = 0;
% starting at day 0
    h = 7;
% hours in one trading day
    %h = 2;
    T = 60*60*h;
% number of seconds in one trading day
    t = 0;
% global time variable

    %% Trader & liquidities
    %   initial trader matrix (treg) is determined
    %   treg format: liquidities, shares -> row number is trader ID

    tnum = 100;
% number of traders

    tliq = 10^5;
% individual trader liquidity

    totShares = 1000*tnum;
% total number of distributed shares

    p0 = 100;
% starting unit price

    tshares = totShares/tnum;
% individual trader shares

    a = p0;
% asking price: seller
    d = p0;
% bid price: buyer

    one = ones(tnum,1);

    treg = [tliq*one, tshares*one, one * 0];
% trader matrix (2 columns)

    for i = 1:1:tnum

        treg( i, 3 ) = 599 + randi(5401);
% determine time window

    end


    %% Books initialisation section (seller & buyer book)
    %   - Book format:
    %   day, time, seller/buyer id, s/b price, shares, dirty bit, age bit,
new
    %   entry number, index of aged entry
    %
    %   - For practical purposes, the book entries are never erased
```

```
    %   - A dirty bit is added to each entry, to inform whether the entry
    %   is active or not
    %   - The number of the entry is the index of the matrix row)
    %   - The age bit is 1 if the entry was made inactive due to its age
    %   - The new entry bit is 1 if the auction was added by a trader who
wants
    %   to refresh his auction

    books = zeros(20000, 9);
% seller book
    sbs = 0;
% actual amount of elements in books
    bookb = zeros(20000, 9);
% buyer book
    sbb = 0;
% actual amount of elements in bookb


    %% Book paging section
    %   - The paging of the book is used to sort the still valid book
entries
    %   without changing the actual book order (sorted chronologically)
    %   - The paging book is sorted after the price
    %   i.e. 1. row : lowest price,        time,    amount of shares,
index of entry
    %        in book, lifespan
    %        2. row : second lowest price, time,    amount of shares,
index of entry
    %        in book, lifespan
    %        3 ...
    %   - When transaction is done: the amount of shares is decreased, or
the
    %   whole entry is ereased if amount of shares == 0

    bookspaging = zeros(2000, 5);
% seller book paging
    sbsp = 0;
% actual #elements in bookspaging
    bookbpaging = zeros(2000, 5);
% buyer book paging
    sbbp = 0;
% actual #elements in bookbpaging


    %% Transaction initialisation section
    %   - Transaction format:
    %   transaction price, amount of shares, seller id, index of entry in
seller book,
    %   buyer id, index of entry in buyer book, transaction time
    %   - Weighted Transaction format:
    %   weighted transaction price, transaction time

    tprice = zeros(1000, 7);
% transaction price matrix
    sbp = 0;
% actual amount of elements in tprice

    avgtprice = zeros(1000, 2);
% weighted transaction price matrix
```

```
    savtp = 0;
% actual number of elements in avgtprice


    %% Plot parameter section
    ymin = p0 - 25;
    ymax = p0 + 25;


    %% Log returns
    ret = zeros( 1000, 2 );
    retsize = 0;
    dt = 60;


    %% Debug
    debug = zeros( 1000, 4 );
    sd1 = 0; sd2 = 0; sd3 = 0; sd4 = 0;

    %% Maximum & minimum price ever occuring
    pmax = p0;
    pmin = p0;


    entrefresh = 0;
    volfeed = 0;
    entage = 1;
    mulshares = 1;

    korrbs = 1.0;
    korrk = 0.0004;
    a0 = 600;
    k = 1.5*1.12;

    tocc = zeros( 1000, 1 );
    st = 0;
    sige = zeros( 1000, 1 );
    se = 0;

    liveplot = 0;
    retymin = -5;
    retymax = 5;

    tregB4 = treg;
% initial trader matrix used for comparison
    tnumB4 = tnum;

    %% Price Regulation

    regulate = 0;
% toggle price regulation : 1 - on , 0 - off

    pC = 110;
% price ceiling
    pF = 100;
% price floor

    % allowing growth
```

```
    growth = 15;
% constant allowed growth : % per day


    %% Financial bubble

    t1 = 1;
% growth start
    t2 = 7;
% crash
    t3 = 9;
% back to stability

    mu1 = 1.003;
% increase
    mu2 = 0.980;
% crash
    mu3 = 1;
% stability

    devon = 0;


    %% Plot label caption!!!
    xfs = 14;
    yfs = 14;
    tfs = 16;



    %% Define and initialize current system state structure variable
    %   Group system parameters in a single structure for convenience and
    %   clarity
    global SSM;
    SSM = struct(   'treg',          treg,           ...
                    'bookbpaging',   bookbpaging,    ...
                    'sbbp',          sbbp,           ...
                    'bookspaging',   bookspaging,    ...
                    'sbsp',          sbsp,           ...
                    'bookb',         bookb,          ...
                    'sbb',           sbb,            ...
                    'books',         books,          ...
                    'sbs',           sbs,            ...
                    'a',             a,              ...
                    'd',             d,              ...
                    'sigma',         sigma,          ...
                    'tprice',        tprice,         ...
                    'sbp',           sbp,            ...
                    'ret',           ret,            ...
                    'retsize',       retsize,        ...
                    'avgtprice',     avgtprice,      ...
                    'savtp',         savtp,          ...
                    'sige',          sige,           ...
                    'se',            se,             ...
                    'tocc',          tocc,           ...
                    'st',            st,             ...
                    'debug',         debug,          ...
                    'sd1',           sd1,            ...
                    'sd2',           sd2,            ...
```

```
                    'sd3',          sd3,            ...
                    'sd4',          sd4,            ...
                    'pmax',         pmax,           ...
                    'pmin',         pmin            );


    %% System parameters
    %   Structure containing the parameters of the system
    global SP;
    SP = struct(    'tnum',         tnum,           ...
                    'totShares',    totShares,      ...
                    'bkempty',      bkempty,        ...
                    'mu',           mu,             ...
                    'lambda',       lambda,         ...
                    'p0',           p0,             ...
                    'T',            T,              ...
                    'M',            M,              ...
                    'dt',           dt,             ...
                    'korrbs',       korrbs,         ...
                    'korrk',        korrk,          ...
                    'a0',           a0,             ...
                    'k',            k,              ...
                    'entage',       entage,         ...
                    'entrefresh',   entrefresh,     ...
                    'mulshares',    mulshares,      ...
                    'volfeed',      volfeed,        ...
                    'regulate',     regulate,       ...
                    'pC',           pC,             ...
                    'pF',           pF,             ...
                    'growth',       growth,         ...
                    'devon',        devon,          ...
                    't1',           t1,             ...
                    't2',           t2,             ...
                    't3',           t3,             ...
                    'mu1',          mu1,            ...
                    'mu2',          mu2,            ...
                    'mu3',          mu3             );


    %% Plot parameters
    global SPL;
    SPL = struct(   'liveplot',     liveplot,       ...
                    'retymax',      retymax,        ...
                    'retymin',      retymin,        ...
                    'ymax',         ymax,           ...
                    'ymin',         ymin,           ...
                    'tregB4',       tregB4,         ...
                    'tnumB4',       tnumB4,         ...
                    'xfs',          xfs,            ...
                    'yfs',          yfs,            ...
                    'tfs',          tfs             );

end
```

- code/Input/param.txt (example)

```
SP.growth           0           1           0
SP.regulate         1           1           1
SP.pF               95          1           100
SP.pC               110         1           115
```

- code/Input/paramSweep.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ctrl ] = paramSweep( SSM, SP, SPL, path, simfold, sparam, sinfo,
ctrl, file, depth, space, tree )
%PARAMSWEEP Parameter sweep recursion function


    depth = sprintf( '%s%s', depth, space );
% tree line spacing
```

```
    if feof(file) == 0

        pos1 = ftell(file);
% save actual position in file

        var = fscanf(file, '%s', 1 );
% read line
        startval = fscanf(file, '%f', 1);
        incval = fscanf(file, '%f', 1);
        endval = fscanf(file, '%f', 1);


        %% Loop of parameter set
        for i = startval:incval:endval


            exec = sprintf( '%s = %u', var, i );
% change to execute
            eval( [ exec, ';' ] );
% evaluate expression


            %% Tree
            stru = sprintf( '%s%s\n', depth, exec );
            fprintf( tree, stru );
% add line in tree


            %% Recursion
            [ ctrl ] = paramSweep( SSM, SP, SPL, path, simfold, sparam,
sinfo, ctrl, file, depth, space, tree );


        end

        fseek(file,pos1,'bof');


    else


        %% Tree
        stru = sprintf( '%s=> simulation %u\n', depth, ctrl );
        fprintf( tree, stru );
% add line in tree


        %% Saving input file
        savefile( SSM, SP, SPL, path, simfold, sparam, sinfo, ctrl );

        ctrl = ctrl + 1;
% increment simulation index


    end

end
```

- code/Input/paramSweepCall.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ctrl ] = paramSweepCall( SSM, SP, SPL, path, simfold, sparam,
sinfo, parfold, spr, stru )
%PARAMSWEEPCALL Creates the simulation input files

    cd( '../' );

        parampath = sprintf( '%s/%s', parfold, spr );
        file = fopen( parampath, 'r' );
% file containing sweep information
        treepath = sprintf( '%s/%s', path, stru );
        tree = fopen( treepath, 'w' );
% file with simulation structure

    cd( 'Input/' );

    %% Initialise simulation structure
    depth = '';
    space = '                ';


    %% Paramater sweep recursion
    ctrl = 0;
% index of simulation

    frewind( file );
    [ ctrl ] = paramSweep( SSM, SP, SPL, path, simfold, sparam, sinfo,
ctrl, file, depth, space, tree );


    fclose( tree );
    fclose( file );


end
```

- code/Input/savefile.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ] = savefile( SSM, SP, SPL, path, simfold, sparam, sinfo, ctrl )
%SAVEFILE Save input file


    cd( '../' );

    %% Actual simulation folder
    simufolder = sprintf( '%s/%s', path, simfold );
    simufolder = regexprep( simufolder, '**', num2str(ctrl) );
    mkdir( simufolder );


    %% Create parameters file
    filename = sprintf( '%s/%s', simufolder, sparam );
    filename = regexprep( filename, '**', num2str(ctrl) );
    save( filename, 'SSM', 'SP', 'SPL' );


    %% Create info file
    infofilename = sprintf( '%s/%s', simufolder, sinfo );
    infofilename = regexprep( infofilename, '**', num2str(ctrl) );
    file = fopen( infofilename, 'w' );


        %% Overhead
        dstr = date;
        cstr = clock;
        fprintf(file, '©2012 ETH Zuerich\nBambolei\nNicholas Eyring, Youri
Popoff\nSimulation of trading in an artificial stock market\n\n\n\n');
        fprintf(file, '                                          **** SIMULATION
%u ****\n\n', ctrl);
        fprintf(file, '
Information file\n\n');

        fprintf(file, '
Date:  %s\n',     dstr);
        fprintf(file, '
Time:  %02.0f:%02.0f:%02.0f\n\n', cstr(4), cstr(5), cstr(6));


        %% Parameters
        fprintf(file, ' - SIMULATION MODUS -
\n');
        fprintf(file, '                    Empty book (On/Off) -
bkempty:  %u\n',     SP.bkempty);
        fprintf(file, '                 Entry refresh (On/Off) -
entrefresh:  %u\n',     SP.entrefresh);
        fprintf(file, '       Entry erasure when aged (On/Off) -
entage:  %u\n',     SP.entage);
```

```
        fprintf(file, '               Multiple shares (On/Off) -
mulshares:  %u\n',     SP.mulshares);
        fprintf(file, '           Volatility feedback (On/Off) -
volfeed:  %u\n',     SP.volfeed);
        fprintf(file, '              Price regulation (On/Off) -
regulate:  %u\n',     SP.regulate);
        fprintf(file, '              Financial bubble (On/Off) -
devon:  %u\n\n\n', SP.devon);

        fprintf(file, ' - STANDARD PARAMETERS (used in all of the modi)
\n');
        fprintf(file, '                       Number of traders -
tnum:  %u\n',     SP.tnum);
        fprintf(file, '                        Amount of shares -
totShares:  %u\n',     SP.totShares);
        fprintf(file, '                          Starting price -
p0:  %.2f\n\n', SP.p0);

        fprintf(file, '            Mean of normal distribution -
mu:  %.4f\n',   SP.mu);
        fprintf(file, ' Initial std. deviation of normal dist. -
sigma:  %.4f\n',   SSM.sigma);
        fprintf(file, '  Parameter of exponential distribution -
lambda:  %.4f\n\n', SP.lambda);

        fprintf(file, '                             Total days -
M:  %u\n',     SP.M);
        fprintf(file, '                Total time in seconds -
T:  %u\n\n',   SP.T);

        fprintf(file, '             Last tick iteration window -
dt:  %.2f\n\n\n', SP.dt);

        fprintf(file, ' - VOLATILITY FEEDBACK PARAMETERS -
\n');
        fprintf(file, '                  Sigma constant factor -
k:  %.2f\n',   SP.k);
        fprintf(file, '              Buyer - Seller correlation -
korrbs:  %.2f\n',   SP.korrbs);
        fprintf(file, '            Old - New sigma correlation -
korrk:  %.6f\n',   SP.korrk);
        fprintf(file, '                  Maximum age of entry -
a0:  %.2f\n\n\n', SP.a0);

        fprintf(file, ' - PRICE REGULATION PARAMETERS -
\n');
        fprintf(file, '                                   Growth -
growth:  %.2f\n',   SP.growth);
        fprintf(file, '                            Price floor -
pF:  %.2f\n',   SP.pF);
        fprintf(file, '                          Price ceiling -
pC:  %.2f\n\n\n', SP.pC);

        fprintf(file, ' - FINANCIAL BUBBLE PARAMETERS -
\n');
        fprintf(file, '          Stability mean of norm. dist. -
mu3:  %.2f\n',   SP.mu3);
        fprintf(file, '                     Bubble starting day -
t1:  %u\n',     SP.t1);
        fprintf(file, '           Increase mean of norm. dist. -
mu1:  %.2f\n',   SP.mu1);
```

```
        fprintf(file, '                    Day of bubble burst -
t2:  %u\n',     SP.t2);
        fprintf(file, '           Decrease mean of norm. dist. -
mu2:  %.2f\n',    SP.mu2);
        fprintf(file, '           Day of stability after burst -
t3:  %u\n\n\n', SP.t3);


        fclose(file);


        cd( 'Input/' );


end
```

code/Plot

- code/Plot/comparePlot.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ] = comparePlot( SSM, SP, SPL, fig9 )
%COMPAREPLOT Plots the log returns, the price and the sigma versus time
%   Used for result comparison


    set( 0, 'CurrentFigure', fig9 );

    %% Log returns plot
    subplot(2,2,1);
```

```
    xmax = SP.M * SP.T;

    T = SP.dt:SP.dt:SP.dt*SSM.retsize;

    hold on;

    plot( T, SSM.ret(1:1:SSM.retsize,2));
    xlim([0 xmax]);
    ylim([SPL.retymin SPL.retymax]);
    xlabel('time in seconds', 'fontsize', SPL.xfs);
    ylabel('percent', 'fontsize', SPL.yfs);
    title('Log-returns', 'fontsize', SPL.tfs);


    for j = 1:1:SP.M

        line( [ j*SP.T j*SP.T ], [ SPL.retymin SPL.retymax ], 'Color',
[0.75, 0.75, 0.75] );

    end

    hold off;

    %% Sigma plot
    subplot(2,2,2);

    if SP.volfeed == 0
        plot([ones(SP.M*SP.T,1),SSM.sigma * ones(SP.M*SP.T,1)]);
        xlim([0 SP.M*SP.T]);
        xlabel('time in seconds', 'fontsize', SPL.xfs);

        for j = 1:1:SP.M

            line( [ j*SP.T j*SP.T ], [ 0.0 0.2 ], 'Color', [0.75, 0.75,
0.75] );

        end

    else
        plot([ones(SSM.se,1),SSM.sige(1:SSM.se,1)]);
        xlim([0 SSM.se]);
        xlabel('time (trading occurence)', 'fontsize', SPL.xfs);
    end
    ylim([0.0 0.2]);
    title('Sigma', 'fontsize', SPL.tfs);
    ylabel('sigma', 'fontsize', SPL.yfs);



    %% Transaction price plot
    [ SPL.ymin, SPL.ymax ] = dynamicLimity( SSM.pmax, SSM.pmin, SPL.ymin,
SPL.ymax );

    xmax = SP.M * SP.T;


    %% Transaction price subplot
    subplot(2,2,[3 4]);
```

```
    hold on;

        Ap3 = [ 0; SSM.avgtprice( 1:1:int32(SSM.savtp), 2 ) ];
        Bp3 = [ SP.p0; SSM.avgtprice( 1:1:int32(SSM.savtp), 1 ) ];
        plot( Ap3, Bp3, 'r' );

    hold off;

        xlim([0 xmax]);
        ylim([SPL.ymin SPL.ymax]);
        xlabel('time in seconds', 'fontsize', SPL.xfs);
        ylabel('transaction price', 'fontsize', SPL.yfs);
        title('Transaction price', 'fontsize', SPL.tfs);



    %% Vertical day line

    for j = 1:1:SP.M

        subplot( 2, 2, [3 4] );
        line( [ j*SP.T j*SP.T ], [ SPL.ymin SPL.ymax ], 'Color', [0.75,
0.75, 0.75] );

    end



end
```

- code/Plot/dynamicLimity.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ymin, ymax ] = dynamicLimity( pmax, pmin, ymin, ymax )
%DYNAMICLIMITY computes the limit of the price graph
%   The limits adapt itself to the min & max price

    margin = 20;
```

```
    max1 = pmax + margin;
    min1 = pmin - margin;

    if max1 > ymax

        ymax = max1;

    end


    if min1 < ymin

        ymin = min1;

    end


end
```

- code/Plot/graphics.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


%% Graphics setup section
%   Computes figure size depending on screen size

close all;
% close opened figures

set(gcf,'doublebuffer', 'on');
% remove flicker

scrsize = get( 0, 'ScreenSize' );
% get screen dimensions (main, secondary screen)
wd = scrsize( 3 );
hg = scrsize( 4 );

scrratio = hg / wd;
% compute ratio; 4/3, 16/10, etc.

fig1wd = wd * 0.75;
% fig1 75% of the screen
fig1hg = fig1wd * scrratio;

fig1 = figure( 1 );
% select figure for plot
set( fig1, 'OuterPosition', [ 0, hg, fig1wd, fig1hg ] );
% position & size of the figure
```

```
fig2wd = wd * 0.25;
% fig2 25% of the screen
fig2hg = fig2wd * scrratio;

fig2 = figure( 2 );
% position & size of the figure
set( fig2, 'OuterPosition', [ fig1wd, hg, fig2wd, fig2hg ] );

fig3wd = wd * 0.25;
fig3hg = fig3wd * scrratio;

fig3 = figure( 3 );
set( fig3, 'OuterPosition', [ 0.75/2 * wd, hg, fig3wd, fig3hg  ] );

fig4 = figure( 4 );
fig5 = figure( 5 );
fig6 = figure( 6 );
fig7 = figure( 7 );
fig8 = figure( 8 );

fig9 = figure( 9 );
% compare plot
set( fig9, 'OuterPosition', [ 0, hg, fig1wd, fig1hg ] );
% same size as fig1
```

- code/Plot/liveplot.m

```
%©2012 ETH Zürich
%Bambolei
```

```
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


set(gcf,'doublebuffer', 'on');
%remove flicker

scrsize = get( 0, 'ScreenSize' );
%get screen dimensions (main, secondary screen)
wd = scrsize( 3 );
hg = scrsize( 4 );

scrratio = hg / wd;
%compute ratio; 4/3, 16/10, etc.

fig1wd = wd * 0.75;
%fig1 75% of the screen
fig1hg = fig1wd * scrratio;

fig1 = figure( 1 );
%select figure for plot
set( fig1, 'OuterPosition', [ 0, hg, fig1wd, fig1hg ] );
%position & size of the figure
```

- code/Plot/plotAssets.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ] = plotAssets( SSM, SP, SPL )
%PLOTASSETS Plot trader assests & estimated firm value

    %% Plot trader assets

    subplot(3,2,1)

    bar(1:1:SPL.tnumB4, SPL.tregB4(:,1))
% initial trader liquidities
    xlabel('trader ID', 'fontsize', SPL.xfs);
```

```
    ylabel('initial trader liquidities', 'fontsize', SPL.yfs);

    subplot(3,2,2)

    bar(1:1:SPL.tnumB4, SPL.tregB4(:,2))
% initial trader share holdings
    xlabel('trader ID', 'fontsize', SPL.xfs);
    ylabel('initial trader share holdings', 'fontsize', SPL.yfs);

    subplot(3,2,3)

    bar(1:1:SP.tnum, SSM.treg(:,1))
% final trader liquidities
    xlabel('trader ID', 'fontsize', SPL.xfs);
    ylabel('final trader liquidities', 'fontsize', SPL.yfs);

    subplot(3,2,4)

    bar(1:1:SP.tnum, SSM.treg(:,2))
% final trader share holdings
    xlabel('trader ID', 'fontsize', SPL.xfs);
    ylabel('final trader share holdings', 'fontsize', SPL.yfs);


    %% Plot Estimated firm value

    subplot(3,2,[5 6])

    inds = 1 + (SSM.savtp - mod(SSM.savtp,3))/3;

    Value =
[SP.totShares*SP.p0,SP.totShares*SSM.avgtprice(inds,1),SP.totShares*SSM.avg
tprice(2*inds,1), SP.totShares*SSM.avgtprice(SSM.savtp,1)];

    bar([0,SSM.avgtprice(inds,2),SSM.avgtprice(2*inds,2),
SSM.avgtprice(SSM.savtp,2)],Value)
    xlabel('time (s)', 'fontsize', SPL.xfs);
    ylabel('estimated firm value', 'fontsize', SPL.yfs);


end
```

- code/Plot/plotCall.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ ] = plotCall( SSM, SP, SPL, plotpath, comparepath )
%PLOTCALL Calls the different plotting functions


    graphics;



    i = SP.M * SP.T;
% ending time

    %% Evolution of price
    [ SPL ] = plotPrice( i, SSM, SP, SPL, fig1 );


    %% Assets
    set( 0, 'CurrentFigure', fig2 );
    plotAssets( SSM, SP, SPL );


    %% Log returns
    plotLogReturns( SSM, SP, SPL, fig4 );


    %% Evolution of occurrence
    set( 0, 'CurrentFigure', fig5 );
    hist(SSM.tocc(1:SSM.st,1));
    xlabel('time b/t 2 occurences', 'fontsize', SPL.xfs);
    ylabel('amount of occurences', 'fontsize', SPL.yfs);
    title('Trade occrrence', 'fontsize', SPL.tfs);


    %% Evolution of the normal distribution
    set( 0, 'CurrentFigure', fig6 );
    hist(SSM.debug(1:SSM.sd3,3));

    xlabel('price coefficient', 'fontsize', SPL.xfs);
    ylabel('amount', 'fontsize', SPL.yfs);
    title('Price distribution', 'fontsize', SPL.tfs);
```

```
    %% Evolution of sigma
    set( 0, 'CurrentFigure', fig7 );
    if SP.volfeed == 0
        plot([ones(SP.M*SP.T,1),SSM.sigma * ones(SP.M*SP.T,1)]);
        xlim([0 SP.M*SP.T]);
        xlabel('time in seconds', 'fontsize', SPL.xfs);

        for j = 1:1:SP.M

            line( [ j*SP.T j*SP.T ], [ 0.0 0.2 ], 'Color', [0.75, 0.75,
0.75] );

        end

    else
        plot([ones(SSM.se,1),SSM.sige(1:SSM.se,1)]);
        xlim([0 SSM.se]);
        xlabel('time (trading occurence)', 'fontsize', SPL.xfs);
    end
    ylim([0.0 0.2]);
    title('Sigma', 'fontsize', SPL.tfs);
    ylabel('sigma', 'fontsize', SPL.yfs);



    %% Wealth of taders
    set( 0, 'CurrentFigure', fig8 );
    hist(SSM.treg(1:SP.tnum,1) + SP.p0 * SSM.treg(1:SP.tnum,2));

    xlabel('total wealth (liquidities + price * shares)', 'fontsize',
SPL.xfs);
    ylabel('amount', 'fontsize', SPL.yfs);
    title('Wealth of the traders', 'fontsize', SPL.tfs);


    %% Debug info
    Ubooks = unique(SSM.books(1:1:SSM.sbs,2));
    length(Ubooks);
    SSM.sbs;
    Ubookb = unique(SSM.bookb(1:1:SSM.sbb,2));
    length(Ubookb);
    SSM.sbb;

    %% Compare plot
    comparePlot(SSM, SP, SPL, fig9);


    %% Saving section
    cd('../');

        if SPL.ymax < 400
            figname = sprintf('%sprice', comparepath );
            saveas(fig9, figname, 'png' );
        end


        figname = sprintf('%sprice', plotpath );
        saveas(fig1, figname, 'png' );
```

```
        figname = sprintf('%sassets', plotpath );
        saveas(fig2, figname, 'png' );

        figname = sprintf('%slogreturns', plotpath );
        saveas(fig4, figname, 'png' );

        figname = sprintf('%seventocc', plotpath );
        saveas(fig5, figname, 'png' );

        figname = sprintf('%spricedist', plotpath );
        saveas(fig6, figname, 'png' );

        figname = sprintf('%ssigma', plotpath );
        saveas(fig7, figname, 'png' );

        figname = sprintf('%swealth', plotpath );
        saveas(fig8, figname, 'png' );

    cd('Plot/');


end
```

- code/Plot/plotLogReturns.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
```

```
%Simulation of trading in an artificial stock market


function [ ] = plotLogReturns( SSM, SP, SPL, fig4 )
%PLOTLOGRETURNS Plots log-returns over time
%   Uses previous-tick interpolation

    set(0, 'CurrentFigure', fig4 );
    xmax = SP.M * SP.T;

    T = SP.dt:SP.dt:SP.dt*SSM.retsize;

    hold on;

    plot( T, SSM.ret(1:1:SSM.retsize,2));
    xlim([0 xmax]);
    ylim([SPL.retymin SPL.retymax]);
    xlabel('time in seconds', 'fontsize', SPL.xfs);
    ylabel('percent', 'fontsize', SPL.yfs);
    title('Log-returns', 'fontsize', SPL.tfs);


    for j = 1:1:SP.M

        line( [ j*SP.T j*SP.T ], [ SPL.retymin SPL.retymax ], 'Color',
[0.75, 0.75, 0.75] );

    end

    hold off;


end
```

- code/Plot/plotPrice.m

```
%©2012 ETH Z?rich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


function [ SPL ] = plotPrice( i, SSM, SP, SPL, fig1 )
%PLOTPRICE Plots every deltat the price evolution

    set( 0, 'CurrentFigure', fig1 );

    % Plot intervall
    deltat = 50;
% plot every 50s


    if mod(i, deltat) == 0

        [ SPL.ymin, SPL.ymax ] = dynamicLimity( SSM.pmax, SSM.pmin,
SPL.ymin, SPL.ymax );
```

```
xmax = SP.M * SP.T;


%% Get last transaction infos & plot in green
if SSM.sbp ~= 0

    info = SSM.tprice( SSM.sbp, : );
    ent1 = info( 4 );
    ent1info = SSM.books( ent1, : );
    ent2 = info( 6 );
    ent2info = SSM.bookb( ent2, : );

    x = ent2info( 2 ); y = ent2info( 4 );
    subplot(2,2,1); plot( x, y, 'g*');
    line( [ x x ], [ SPL.ymin SPL.ymax ], 'Color', 'g' );

    x = ent1info( 2 ); y = ent1info( 4 );
    subplot(2,2,2); plot( x, y, 'g*' );
    line( [ x x ], [ SPL.ymin SPL.ymax ], 'Color', 'g' );

    x = info( 7 ); y = info( 1 );
    subplot(2,2,[3 4]); plot( x, y, 'g*' );
    line( [ x x ], [ SPL.ymin SPL.ymax ], 'Color', 'g' );

end



%% Buyer price subplot
subplot(2,2,1);
    hold on;

    Ap1 = [ 0; SSM.bookb(1:1:int32(SSM.sbb),2) ];
    Bp1 = [ SP.p0; SSM.bookb(1:1:int32(SSM.sbb),4) ];
    plot(Ap1, Bp1, 'b');

    hold off;

    xlim([0 xmax]);
    ylim([SPL.ymin SPL.ymax]);
    xlabel('time in seconds', 'fontsize', SPL.xfs);
    ylabel('buyer entry price', 'fontsize', SPL.yfs);
    title('Buyer price', 'fontsize', SPL.tfs);


%% Seller price subplot
subplot(2,2,2);
    hold on;

    Ap2 = [ 0; SSM.books(1:1:int32(SSM.sbs),2) ];
    Bp2 = [ SP.p0; SSM.books(1:1:int32(SSM.sbs),4) ];
    plot(Ap2, Bp2, 'b');

    hold off;

    xlim([0 xmax]);
    ylim([SPL.ymin SPL.ymax]);
```

```
            xlabel('time in seconds', 'fontsize', SPL.xfs);
            ylabel('seller entry price', 'fontsize', SPL.yfs);
            title('Seller price', 'fontsize', SPL.tfs);


        %% Transaction price subplot
        subplot(2,2,[3 4]);
            hold on;

            Ap3 = [ 0; SSM.avgtprice( 1:1:int32(SSM.savtp), 2 ) ];
            Bp3 = [ SP.p0; SSM.avgtprice( 1:1:int32(SSM.savtp), 1 ) ];
            plot( Ap3, Bp3, 'r' );

            hold off;

            xlim([0 xmax]);
            ylim([SPL.ymin SPL.ymax]);
            xlabel('time in seconds', 'fontsize', SPL.xfs);
            ylabel('transaction price', 'fontsize', SPL.yfs);
            title('Transaction price', 'fontsize', SPL.tfs);



            %% Vertical day line

        for j = 1:1:SP.M

            subplot( 2, 2, 1 );
            line( [ j*SP.T j*SP.T ], [ SPL.ymin SPL.ymax ], 'Color', [0.75,
0.75, 0.75] );

            subplot( 2, 2, 2 );
            line( [ j*SP.T j*SP.T ], [ SPL.ymin SPL.ymax ], 'Color', [0.75,
0.75, 0.75] );

            subplot( 2, 2, [3 4] );
            line( [ j*SP.T j*SP.T ], [ SPL.ymin SPL.ymax ], 'Color', [0.75,
0.75, 0.75] );

        end


        drawnow;


    end


end
```

Other

- code/Other/defaultParam.m

```
%©2012 ETH Zürich
%Bambolei
%Nicholas Eyring, Youri Popoff
%Simulation of trading in an artificial stock market


% This code creates an info file with the default (generic) parameter
% values


cd('../Input');

    [SSM, SP, SPL] = init();

    infofilename = 'defaultInfo.txt';

    file = fopen( infofilename, 'w' );


        %% Overhead
        dstr = date;
        cstr = clock;
        fprintf(file, '©2012 ETH Zuerich\nBambolei\nNicholas Eyring, Youri
Popoff\nSimulation of trading in an artificial stock market\n\n\n\n');
        fprintf(file, '                                        **** Default
parameter values ****\n\n');
        fprintf(file, '
Information file\n\n');
```

```
        fprintf(file, '
Date:  %s\n',     dstr);
        fprintf(file, '
Time:  %02.0f:%02.0f:%02.0f\n\n', cstr(4), cstr(5), cstr(6));


        %% Parameters
        fprintf(file, ' - SIMULATION MODUS -
\n');
        fprintf(file, '                    Empty book (On/Off) -
bkempty:  %u\n',     SP.bkempty);
        fprintf(file, '                 Entry refresh (On/Off) -
entrefresh:  %u\n',     SP.entrefresh);
        fprintf(file, '       Entry erasure when aged (On/Off) -
entage:  %u\n',     SP.entage);
        fprintf(file, '               Multiple shares (On/Off) -
mulshares:  %u\n',     SP.mulshares);
        fprintf(file, '           Volatility feedback (On/Off) -
volfeed:  %u\n',     SP.volfeed);
        fprintf(file, '              Price regulation (On/Off) -
regulate:  %u\n',     SP.regulate);
        fprintf(file, '              Financial bubble (On/Off) -
devon:  %u\n\n\n\n', SP.devon);

        fprintf(file, ' - STANDARD PARAMETERS (used in all of the modi)
\n');
        fprintf(file, '                      Number of traders -
tnum:  %u\n',     SP.tnum);
        fprintf(file, '                       Amount of shares -
totShares:  %u\n',     SP.totShares);
        fprintf(file, '                         Starting price -
p0:  %.2f\n\n', SP.p0);

        fprintf(file, '            Mean of normal distribution -
mu:  %.4f\n',   SP.mu);
        fprintf(file, ' Initial std. deviation of normal dist. -
sigma:  %.4f\n',   SSM.sigma);
        fprintf(file, '  Parameter of exponential distribution -
lambda:  %.4f\n\n', SP.lambda);

        fprintf(file, '                             Total days -
M:  %u\n',     SP.M);
        fprintf(file, '                Total time in seconds -
T:  %u\n\n',   SP.T);

        fprintf(file, '             Last tick iteration window -
dt:  %.2f\n\n\n', SP.dt);

        fprintf(file, ' - VOLATILITY FEEDBACK PARAMETERS -
\n');
        fprintf(file, '                  Sigma constant factor -
k:  %.2f\n',   SP.k);
        fprintf(file, '             Buyer - Seller correlation -
korrbs:  %.2f\n',   SP.korrbs);
        fprintf(file, '            Old - New sigma correlation -
korrk:  %.6f\n',   SP.korrk);
        fprintf(file, '                  Maximum age of entry -
a0:  %.2f\n\n\n', SP.a0);
```

```
        fprintf(file, ' - PRICE REGULATION PARAMETERS -
\n');
        fprintf(file, '                                    Growth -
growth:  %.2f\n',   SP.growth);
        fprintf(file, '                               Price floor -
pF:  %.2f\n',   SP.pF);
        fprintf(file, '                             Price ceiling -
pC:  %.2f\n\n\n', SP.pC);

        fprintf(file, ' - FINANCIAL BUBBLE PARAMETERS -
\n');
        fprintf(file, '          Stability mean of norm. dist. -
mu3:  %.2f\n',   SP.mu3);
        fprintf(file, '                     Bubble starting day -
t1:  %u\n',     SP.t1);
        fprintf(file, '           Increase mean of norm. dist. -
mu1:  %.2f\n',   SP.mu1);
        fprintf(file, '                     Day of bubble burst -
t2:  %u\n',     SP.t2);
        fprintf(file, '           Decrease mean of norm. dist. -
mu2:  %.2f\n',   SP.mu2);
        fprintf(file, '           Day of stability after burst -
t3:  %u\n\n\n', SP.t3);


        fclose(file);

cd('../Other');
```

6.3 References

[1] M.Raberto, S. Cincotti, Modeling and simulation of a double auction artificial financial market, Physica A 355 (2005) 34-45, 2005