АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

СПРАВКА

о результатах проверки текстового документа
на наличие заимствований

Национальный Исследовательский
Университет "Высшая Школа Экономики"

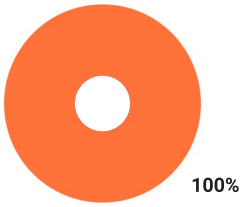
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Асташкин Артемий Андреевич
Самоцитирование рассчитано для:
Название работы: Software project DSBA - Final Report Title Page-signed (1).pdf
Тип работы: Не указано
Подразделение:

РЕЗУЛЬТАТЫ

ЗАИМСТВОВАНИЯ	100%
ОРИГИНАЛЬНОСТЬ	0%
ЦИТИРОВАНИЯ	0%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 28.05.2022



Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по Интернету (EnRu); Переводные заимствования издательства Wiley (RuEn); eLIBRARY.RU; СПС ГАРАНТ; Модуль поиска "ВШЭ"; Медицина; Диссертации НББ; Перефразирования по eLIBRARY.RU; Перефразирования по Интернету; Перефразирования по коллекции издательства Wiley; Патенты СССР, РФ, СНГ; СМИ России и СНГ; Шаблонные фразы; Кольцо вузов; Издательство Wiley; Переводные заимствования

Работу проверил: Асташкин Артемий Андреевич

ФИО проверяющего

Дата подписи:

Подпись проверяющего

Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование корректным, система оставляет на усмотрение проверяющего. Предоставленная информация не подлежит использованию в коммерческих целях.

Software Project Report

Artemiy Astashkin

25 May 2022

Contents

Abstract

Subject field is the computer-simulating trading on financial markets by applying the agent-based models of financial stable and unstable market using programming language s and algorithms in order to build a working simulator and then make more empirical research on the results we gained.In the last 20+ years many artificial financial markets were built , sometimes with some features of agent-based modelling approach , but those results were too complex, we will try to make it less complicated due to our main model.

Part I

Chapter 1

Before the implementation

1.1 Introduction

Nowadays there are a lot of questions which are connected with how to real financial markets work , simulating the finance market to gain profit in future, how to catch anomal state of markets , our implementation will not only help us students to gain new skills such as finance markets structure and its agents , agents-based models theory and its architecture , but also to improve our programming,applied statistics and data analysis skills and build a working simulator which will in best case recreate and replicate the real world market trading.Otherwise and less likely, the impossibility of applying ABM approach will be reported.

1.1.1 The roles of project's members

The another project participant is Karimov Rustam.The student of the same faculty and program, faculty of computer science HSE, Data Science and business analytics, but of the 3 course. Our main and common first task was to determine and develop the Agents-based model architecture, algorithms of traders agents that we want to implement which will be the base of our Python-simulated market. Then, we started coding together with writing our separate parts. All classes we implemented, which are needed for the project were implemented together, but functions, methods which are in those classes are coded separately. Later in the implementation I will explain what function was concretely made by Rustam and what was made by me.

1.1.2 Functional and Non-Functional requirements

Instruments 1)Python 3.9 2)Google Collaboratory 3)PyCharm 3)Python libraries: numpy,pandas,matplotlib.pyplot, seaborn,random 4) ABM theory(will be listed in the links) 5)meetings with mentor and the papers he sent.

1.1.3 Description

The project is divided into 3 parts with a consequent width subtasks.

- ABM research, architecture of simulator.
- Implementation
- Empirical evaluation .

The final result either the working simulator or explanation of why our algorithm can not be practically applied to the existing real life data, in out case why time series graph is wrong and does not replicate the real world situation.In case of failure, Explanation why is the time series built like that is required. The further comparing and statistic methods are applied only to working one.

1.1.4 Aim

The aim of Agent-based model in economy and our project is to implement and create such a **model** and **algorithm** which will try to simulate and build the artificial financial market, which will be in the golden mean of simple generalization and realistic representation of the worldly known objective facts , metrics of financial time series -that is volatility , fundamental price and all volume on market, state of market maker at each iteration . The main and most important feature and process is the formation of a price on market, that is matching of demand and supply curves(not curves,exactly, but the linear regressions based on all agents choices, to be explained later), but in our case of financial markets , it is actually the bid-ask spread analogue in the orderbook. The agents have finite resources(i.e cash), the agents are heterogeneous and self-organized , in our case we implemented 2 types of agents and in both their structure is made the prerequisite, which is close to the real world, that the Noisy traders and Market Makers do not rely on cash parameter due their behaviour . The "invisible hand of market " is the market owner will invisibly impact on by just one realistic assumption-the liquidity process, which is being made by market maker and stock market agent(not an agent , but the stock exchange glass).We have teams(groups) of different groups of agents(traders, which have the same algorithms , but different attributes due to random sampling of these attributes and its overall limit random process of some attributes) In the first part, ABM is being described , its structure,simulator implementation in Python secondly and the last one is the evaluation of results.The end is quick overlook of the situation and further possible improvements would be suggested.

1.1.5 Key Terms and Definitions

1. Terms
   (a) **ABMS**-Agent-based modelling and simulation (ABMS) is an approach to modelling complex systems composed of interacting, autonomous 'agents'. Agents have behaviours, often described by simple rules, and interactions with other agents, which in turn influence their behaviours. By modelling agents individually, the full effects of the diversity that exists among agents in their attributes(arguements)and behaviours(function of actions they perform based on situation they are in) observed as it gives rise to the behaviour of the system as a whole.To run an agent-based model is to have agents repeatedly execute their behaviours and interactions. This process often does, but

is not necessarily modelled to, operate over a timeline, as in time-stepped, activity-based, or discrete-event simulation structures.(in our case , the first one, time-series are time-stepped)

(b) **Structure of an agent based model**- A typical agent-based model has three elements: set of agents, their attributes and behaviours. A set of agent relationships and methods of interaction: An underlying topology of connectedness defines how and with whom agents interact.(Basically, the scheme , or an algorithm set) The agents' environment: Agents interact with their environment in addition to other agents.In our case agents interact only with Stock Market.

(c) **Autonomous agents** -defining characteristic of an autonomous agent is its capability to act autonomously, that is, to act on its own without external direction in response to situations it encounters(in our case agents will be quite simple ans base)

(d) **Autonomous agents characteristics**-self-contained(modular,has bounary),self directive,agent has a state, it changes,agent is a social -has ineractions with ither agents.Adaptivity (learning , different patterns of behaviour.

(e) **Time series**-Time Series – is a series of values describing a process running in time, measured at consecutive changeover points in time.

(f) **Stock Exchange Glass** - is the list of orders (limit and market) that a trading venue (in our case stock exchanges) uses to record the activity and interest of buyers and sellers in a particular asset of . Exchange agent uses the book to determine which orders can be fully or partially executed.

   i. bid - is an order to buy asset or assets . More concretely, maximum price that a trader-byer is willing to pay for one asset.
   ii. ask - is an order to sell asset or assets.More concretely, minimum price that a trader-seller is willing to earn for an asset.
   iii. (spread- The difference between ask and bid , indicator of liquidity of an asset on a financial market.

(g) **limit order** - is the maximum or minimum price at which trader wants to complete the transaction, a buy or sell of concrete number of assets.

(h) **market order** - are orders which need to be executed as soon as possible at the current best bid or best ask of the market.

(i) **market maker** -trader who operates both a buy and a sell price in a asset held in inventory simultaneously, trying to make earn on the bid–ask spread, limits volatility, provides liquidity.

(j) **Noisy Trader** - trader , which decisions to buy or sell are based on factors he believes to be helpful to profit , but in reality it is the random choices.

(k) **volatility** -

(l) **liquidity** -

Part II

Chapter 2

Description of the model

As stated earlier , our approach is based on abm. In our case, we want to simulate financial market by understanding how main traders work. In our case , we implemented 3 main agents: Stock Market, which is not an agent , but an orderbook which has functions of its development , sorting and updating based on orders it gains.Noisy trader - we named it Zero Intelligence Agent, who performs trading, by random chooses whether he wants to sell or buy. If the random variable , which determines in our simulation his decision, he either hits the bid and ask(between them), so he making price quite near to the actual best bids and asks, or not, so he has some bias , which causes him to lose position and to wait and not to earn , as prices which he proposes can not be executed , so , he loses time and profit as well. The quantity he trades is also made with that algo. Market maker trader- he bases all his actions to stable liquidity and on his inventory.He wants assets to be as much tradeable as possible, together with earning on bid ask spread. He has states ob being in normal or in panic- when his inventory(assets he holds) is less the lower limit, or higher than the upper limit. When in panic, he tries to make his invenory equal to 0 , as to sell or buy many number of assets, to try to increase the liquidity. When he is normal state, he operates on formulas, updates inventory on what plus what he bought minus what he sold. After that he cleares all his previous orders, before the next iteration of trading. Now let me provide the formulas and other processes we make for all 3 types of traders explained higher.

Stock Exchange- We need an orderbook. It can not be void initially, so we create by having an table with columns price, assets , agent- type of agent, and agent id - which concrete one made this order. To not be void, we initate it initiallly with prices and quatities from normal distribution, where in price the $\mu = 300$, $\sigma = 64$. The same for assets(quantity) with 1000 and 64. These parametres can change, as well as size of table, initial-1000. After creation, we sort it descending by price in . Then, we know from theory , that bid ask spread , are spread into 2 parts, by taking median of its price column, so the least price of column of the bigger prices is ask, and biggest in the lesser part - bid. For comfort, we take then these 2 parts as alone tables, in implementation will be explained why. Then, we have procesess of this update of order book, when traders trade- sell and buy for limit orders and for market orders, cancel orders. They are common,for limit we just add to the table of bids or asks and sorting these tables again? for cancels it is the same, we just delete them, like in the

time and sort again(having that concrete trader did it, and of concrete type(all cancelled are limit orders). For market orders, we look either for our bid or ask quantity is bigger then assets, which we want to buy, if less, we just substract from quantity assets we bought, else , we will drop all values when it is equal to zero, it is no more bid or ask price. It is made by while algo, calculating the difference . Then we again sort tables of asks and bids after drops.Now, we are done, we described all types of ineractions with Stock Market

Noisy Trader - has lognormal parametres for mu, sigma. Also has, his price deviation, quantity and quantity deviation, on which lambda and mu, sigma are based on.Has reference to an orderbook object.lambda equal to 1/pricestd.We also follow agents by tracking their id.Now we have a price process, if our random state is less the 0.35,(rs is from 0 to 1 with equal probability for all numbers in this stage), the we hit the bid or ask and from unifrom distributiom between bidrequest and askrequest choose the price in there. In other 65 percent we generate delta, which is bias, and add or plus to minus to the bidrequest or askrequest(expovariate distributed).Quantity is made by mu and sigma we discussed before(lognormally). Now we trade, if the random state bigger 0.5- bid, else ask. if next random state is bigger 0.85, we have market order, less- but bigger then 0.5- limit order, less then 0.5 -limit order cancellation. That is all for noisy.The constants for all parametres were chosen from literature.

Market Maker- we defined him with cash also, but not use it. It is to big , that we can perform many actions.First, he performs both bids and asks at the same time for, as well as volumes. Bid volume is equal to $bidvolume = max(0, Ul - inventory - 1) * U(UniformLow, uniformHigh)$. Where , uniformlow and uniformhigh are constants different for each Id of Market makers. The same for askvolume, but $askvolume = max(0, inventory - 1 - LL) * U(UniformLow, uniformHigh.$ After that we track normal or panic situation, if normal, he updates inventory like that:$inventory + bidvolume - askvolume$, then we update orderbook by $bidRequest-((askRequest-bidRequest)*(inventory/.UL,$ like limit order for bid and $(askRequest+((askRequest-bidRequest)*(inventory/UL)$ for ask. In case of panic he trades to zero his inventory executing market orders.

Simulation-Simulator algo is that we want to track the fundamental price , whole volume on market,state of market makers, volatility at each iteration. In here we have n iterarations, which further can be interpretated like time , and in each iteraion for different group of traders do their 1 action. So inside this cycle we also go through all MMs and Noise traders , where they do, what was explained above. Also, we added the opportunity to visualise the time series of main indicators of market stability in this simulation class.

2.1 Price formation process

That is just average between best bid and best ask at each iteration. $P_t = (Pbid_t + pask_t)/2$

2.1.1 Volatility

- Standart deviation of price from 300(what we initialized above, when created data)

2.1.2 Volume

-volume is sum of volume traded by MM and Noisy traders at each iterations.

Part III

Chapter 3

Implementation

3.1 Architecture

Stages

1. **Tools**-We chose Google collab, because there easily to get the results of each script you right in cell, in our project working with tables it was very important. Next, we used Pycharm for debugging and finding mistakes if it was not possible to catch it Collab.Main libraries we use are pandas, numpy, first one because dataframe is the base of our different tables and easy to calculate many simple statistics, numpy for a quick solution of math operations ans statictical distributions, as well as random module. Seaborn and matplotlib are needed for visualisation.

2. **Structure**- We have 3 parts. The first part is Stock Market created like Class. In init funtion we create base dataframe, then with 4 fields, price, asssets, agent,agentid.Then we sort it by price column, reset indexes, find median index, create tables from this dataframe for bids and asks.$medianIndex = orderBook[orderBook['Price'] == orderBook['Price'].median()].index.values[0$

   Then we have fuctions, that were described in the model description to update dataframes after trader made action. I will show you 3 of 6 of them.

   $defupdBuyLimitOrder(self, price, quantity, agentName, agentid) : self.dfBid = self.dfBid.append(pd.Series([quantity, price, agentName, agentid], index = self.dfBid.columns), ignore_index = True)self.dfBid = self.dfBid.sort_values(by = ['Price'], ascending = False)self.dfBid = self.dfBid.reset_index()self.dfBid = self.dfBid.drop(['index'], axis = 1)$

   $defupdCancelBuyLimitOrder(self, price, quantity, agentName, agentid) : to_cancel = self.dfBid[(self.dfBid['Agent'] == agentName)(self.dfBid['AgentID'] == agentid)].indexself.dfBid.drop(to_cancel, inplace = True)self.dfBid.dropna(inplace = True)self.dfBid = self.dfBid.sort_values(by = ['Price'], ascending = False)self.dfBid = self.dfBid.reset_index()self.dfBid = self.dfBid.drop(['index'], axis = 1)$

   $defupdSellMarketOrder(self, quantity, agentName, agentid) : difference = 0quantity = np.float64(quantity)bestBidPrice, bestBidAssets = self.dfBid.iloc[0]['Price'], self.dfBi$

$bestBidAssets : res = np.float64(bestBidAssets - quantity) self.dfBid.at[0,'Assets'] = res self.dfBid.iloc[0]['Assets'] = self.dfBid.iloc[0]['Assets'] - quantity else : while quantity > 0 : quantity - = bestBidAssets self.dfBid.drop(index = self.dfBid.index[0], axis = 0, inplace = True) self.dfBid = self.dfBid.reset_index() self.dfBid = self.dfBid.drop(['index'], axis = 1) bestBidAssets = self.dfBid.iloc[0]['Assets'] self.dfBid = self.dfBid.sort_values(by = ['Price'], ascending = False) self.dfBid = self.dfBid.reset_index() self.dfBid = self.dfBid.drop(['index'], axis = 1)$

That is more comfortable to follow them at the code itself. These 3 functions were made by me, as well as the idea and code for resetting indexes and median index finding. Other in this class was made by Rustam, after that we create object of this class , which is ORDERBOOK. The second part is for classes for agents themselves. Rustam made classes for agents id as well. Noisy trader has in attributes of class reference to the ORDERBOOK we inizialized in order to be able to interact and send orders to it. Then we have fuctions of init, detepinePrie, determinequantity , and trade itself. I wrote init method : $def_{init}(self, counter, price_std, quantity_mean, quantity_std, orderbook = ORDERBOOK) : self.Lambda = 1/price_std self.mu, self.sigma = logno$
Also, determinequantity: $defdetermineQuantity(self) : value = float('NaN') while np.isnan(value) : value = np.ceil(random.lognormvariate(self.mu, self.sigma)) return value$
In big trade method I wrote part for type of the order: $if not bidRequest or not askRequest : rs0, rs1, orderType = random.random(), random.random(), '' if rs0 > 0.5 : orderType = 'bid' self.orderty$

$elif rs1 > 0.5 : execute limit order print('limit order') price, quantity = self.determinePrice(orderType, bidRequest, askRequest), self.determineQuantity() print(price, 'æ, qu$
And also for execution of limit orders.

For market maker we also have reference to the ORDERBOOK. Has trade, init, bidvolume, askvolume methods. I wrote bidvolume, askvolume.

$defgetBidVolume(self) : bid_volume = float('NaN') while np.isnan(bid_volume) : bid_volume = np.ceil(max(0.0, (self.UL - 1 - self.inventory)) * np.random.uniform(self.uniformLow,$

$defgetAskVolume(self) : ask_volume = float('NaN') while np.isnan(ask_volume) : ask_volume = np.ceil(max(0.0, (self.inventory - self.LL - 1)) * np.random.uniform(self.uniformLow,$
Also, debugged for mistakes and changed a little bit trade fuction, always returning 0 in case of panic of Trader. The last part is about simulator Class, where we just have some for cycles for implementing the needed indicators in iterations.Has init , Simulate and plot methods. I wrote all plot methods and also for cycle for Noisy traders agents.

$defplot_price(self) : plt.plot(np.arange(self.proceedingTime), self.pricesAtEachIteration) plt.legend($
$plt.plot(np.arange(self.proceedingTime), self.volumesAtEachIteration) plt.legend() plt.show() defplo$
$plt.plot(np.arange(self.proceedingTime), self.priceVolatilityatEachIteration) plt.legend() plt.show()$

$for j in range(0, len(self.team0iq)) : self.team0iq[j].trade(np.random.normal(0.75 * 300, 32), np.random.normal(1.25 * 300, 32)) bid, ask = np.random.normal(0.75 * 300, 32), np.random.normal(1.25 * 300, 32) bid = min(bid, ask) ask = max(bid, ask) self.team0iq[j].trade($
$----------') print(self.orderbook.dfAsk.iloc[-1]['Price'], 'æ, self.order$

3. **Lauching**- For launching you need to create objects of ierataions over agents for both MarketMakers and Noisy, then initialize them with params and append to their teams. After this lists of teams of traders are given to the simulator and number of iterations , which is time. THat is all,

after the Simulator object created and done, you can watch all indicators we wrote, as well as plotting.

4. **Results**- Simulator is working, but very slowly for big number(n¿50) of iterations. Tqdm we did not use. We built it , for the 3 course for me I am going to improve it which I will describe in the special section. We are not sure, whether we need to work with dataframe for data strcuture, doubly linked lists works much faster in other implementations of project like that.

3.2 link to collab

https://colab.research.google.com/drive/10eodJmEnh5LT2XcT8BwhRchKPEfgdTNWscrollTo=1tc8c0

Part IV

Conclusion

- We made great job and built the simulator working, writing and realizing orderbook , noisy traders and market makers. However, not everything is bright. We did not have time for coding long-term type of agent and High frequency. We did not have time for realizing the great idea of shocks simulation - lag of time, flashcrash, and how to exactly simulate this in our code so far. We have understanding how this works in real world and pseudocode, but did not manage to implement- will be in the next year , I hope. Also we need ti understand how to track the state of the market itself , not the market make only. The possible variants are increasing and decreasing volatility, spread depth, which indicator for liquidity, flashcrash and opposite, market maker impact on the market (the volume he trades to the whole one), patient state of market e.t.c

Part V

bibliography

Bibliography

[1] Christopher D. Clack, Elias Court, Dmitrijs Zaparanuks Dynamic Coupling and Market Instability https://arxiv.org/abs/2005.13621v1

[2] Samuel Peter, 05-909-122 Agent Based Simulation on a financial market

3.3 quotations

Bid - buy, ask-sell.