NIS 3
BSE : Bristol Stock Exchange

Sergazin Iskander
Borschev Aristarkh
Zakharov Artem
Shatalov Andrew
Keseli Timur

24 November 2023

Abstract: what is BSE and why was it created?

Bristol stock exchange (BSE) is a novel simulation of a centralised financial market based on a **Limit Order Book (LOB)**.

The motivation behind construction of BSE is simple: most of the world's major financial markets are **automated platforms with bots conducting trading operations**. The change to automated trading has altered the dynamics of major financial markets and has created a demand for people with university-level education in the **design and construction** of automated trading systems, and in the **analysis and management** of automated markets.

Abstract: Innovations of BSE

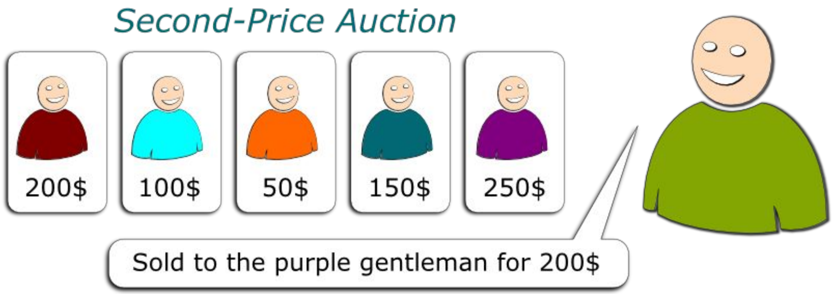
There is a crucial difference between BSE and traditional financial-market simulators. The latter work by regurgitating a time-series database of historical transaction prices. Contrastingly, BSE has taken a different design approach, in BSE the price at the consecutive period is not the historical price, but is the **result of actions and interactions of the traders** active **at the current time period**, thus users of BSE are able to replicate and model **market impact**.

Introduction: BSE design principles

- Open-source - BSE is free to use, modify and experiment with freely available to anyone as a github project.
- Simplicity - BSE does not pursue maximum efficiency, instead it encourages simple design, understandable even for a novice programmer (for example python 2.7 was chosen as a project language for these exact reasons). It also incentives users' modifications and improvements.

Background: Types of auctions

English Auction/first-price ascending-bid auction

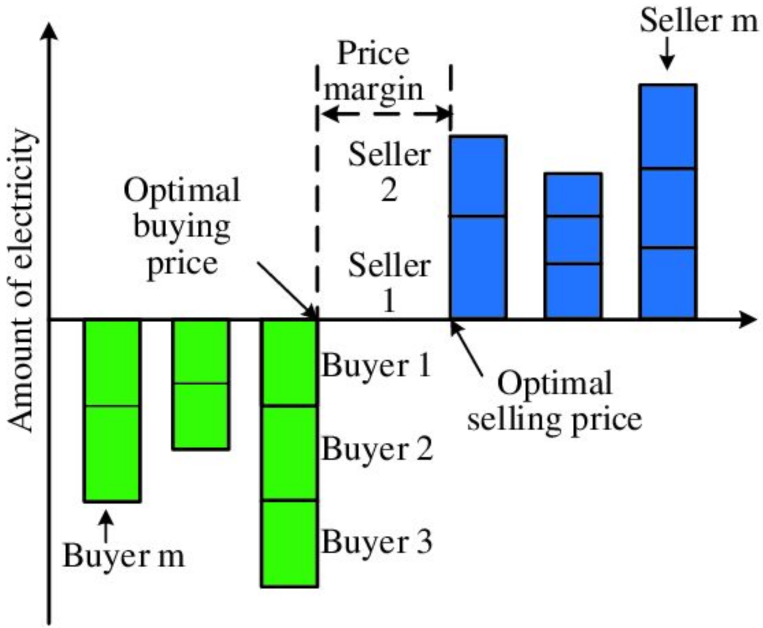


Second-price ascending-bid auction

Posted Offer Auction

Background: Types of auctions (2)

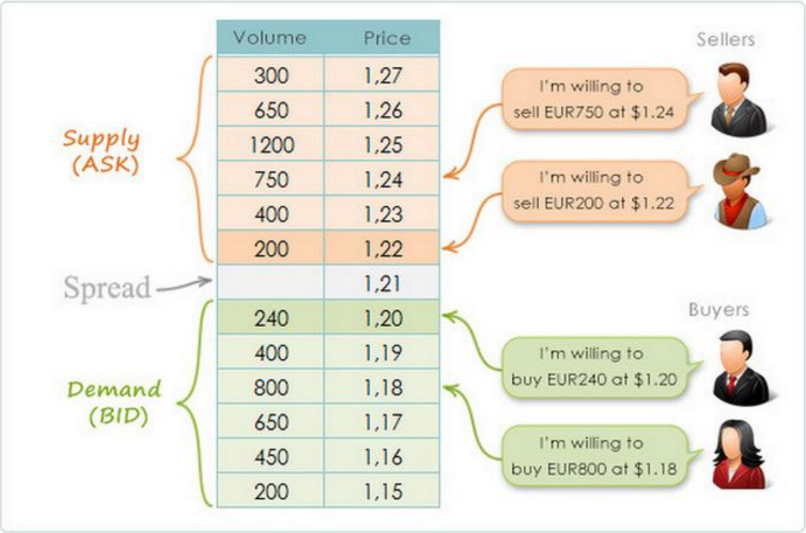
Dutch flower auction



Continuous double auction

Background: Limit order book (LOB)

The continuous double auction **(CDA) is an asynchronous process** and it needs **no centralised auctioneer**. However, it requires some means of storing and classifying non transacted quotes and that is what the **limit order book** (LOB) does.



Background: limitations of historical market data

- **Level 2** is basically LOB data, this data is quite detailed.
- **Level 1** data **is brief summary** of current market state.

Level 2 is computationally and financially expensive and thus is usually unavailable to a typical researcher. Therefore, have no other choice but to use uninformative Level 1 data.

Another limitation is that it is impossible to replicate market impact using historical data.

This encourages the construction of platforms like the BSE.

BSE simplifications

- **Single tradable security** - while in real exchange markets there are many goods and correlations between the goods.

- **Zero latency in communications** between the traders and the exchange. After any transaction takes place, immediately and simultaneously all agents are notified.

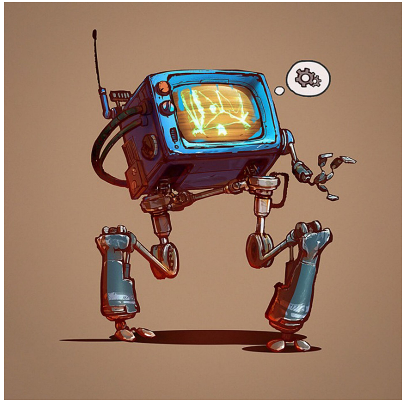
Trading robots – introduction

Trading robot acts as an agent with an objective – buy or sell the certain amount of the stocks within the prescribed price limitations. The better the price is, the more effective bot is.

Customer orders are issued to traders in BSE and then the traders issue their own quotes as bids or asks into the market, trying to get a better price than the limit-price specified by the customer.

Trading robots: Giveaway Bot

- Giveaway bot: sets a quote at a limit price maximising chances of executing the query, but with zero chance to earn profit should a trade result from its quote.



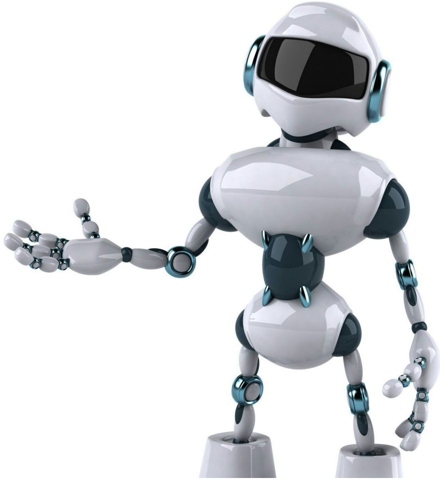
Credit: https://mir-s3-cdn-cf.behance.net/project_modules/1400/733fa750476405.58d18c1756487.jpg

```python
class Trader_Giveaway(Trader):

    def getorder(self, time, countdown, lob):
        if len(self.orders) < 1:
            order = None
        else:
            quoteprice = self.orders[0].price
            order = Order(self.tid,
                          self.orders[0].otype,
                          quoteprice,
                          self.orders[0].qty,
                          time, lob['QID'])
            self.lastquote = order
        return order
```

Trading robots: Zero-Intelligence Constrained

- **ZIC bot**: randomly selects price in uniformal distribution between best market deal and limit price. Market populated by ZIC bots shows the same market efficiency as human populated market.

```
class Trader_ZIC(Trader):

    def getorder(self, time, countdown, lob):
        if len(self.orders) < 1:
            # no orders: return NULL
            order = None
        else:
            minprice = lob['bids']['worst']
            maxprice = lob['asks']['worst']
            qid = lob['QID']
            limit = self.orders[0].price
            otype = self.orders[0].otype
            if otype == 'Bid':
                quoteprice = random.randint(int(minprice), int(limit))
            else:
                quoteprice = random.randint(int(limit), int(maxprice))
                # NB should check it == 'Ask' and barf if not
            order = Order(self.tid, otype, quoteprice, self.orders[0].qty, time, qid)
            self.lastquote = order
        return order
```



Credit: https://tvmag.by/wa-data/public/site/new_file/icons/robots.jpg

Trading robots: Shaver

- **Shaver (SHVR)**: analyses the best market price and offers better price by the smallest step - 0.01, as long as it is within his limits, using max/min method if there is no best

```
def getorder(self, time, countdown, lob):
    if len(self.orders) < 1:
        order = None
    else:
        limitprice = self.orders[0].price
        otype = self.orders[0].otype
        if otype == 'Bid':
            if lob['bids']['n'] > 0:
                quoteprice = lob['bids']['best'] + 1
                if quoteprice > limitprice:
                    quoteprice = limitprice
            else:
                quoteprice = lob['bids']['worst']
        else:
            if lob['asks']['n'] > 0:
                quoteprice = lob['asks']['best'] - 1
                if quoteprice < limitprice:
                    quoteprice = limitprice
            else:
                quoteprice = lob['asks']['worst']
        order = Order(self.tid, otype, quoteprice, self.orders[0].qty, time, lob['QID'])
        self.lastquote = order
    return order
```

Credit: Stuff Made Here,
https://www.youtube.com/watch?v=7zBrbdU_y0s

Trading robots: Sniper

- **Sniper (SNPR)**: this bot won the bots' market competition. Its strategy is staying low, until the time is about to run out or the gap between best and worst price is low and steal the show

```
def getorder(self, time, countdown, lob):
    lurk_threshold = 0.2
    shavegrowthrate = 3
    shave = int(1.0 / (0.01 + countdown / (shavegrowthrate * lurk_threshold)))
    if (len(self.orders) < 1) or (countdown > lurk_threshold):
        order = None
    else:
        limitprice = self.orders[0].price
        otype = self.orders[0].otype

        if otype == 'Bid':
            if lob['bids']['n'] > 0:
                quoteprice = lob['bids']['best'] + shave
                if quoteprice > limitprice:
                    quoteprice = limitprice
            else:
                quoteprice = lob['bids']['worst']
        else:
            if lob['asks']['n'] > 0:
                quoteprice = lob['asks']['best'] - shave
                if quoteprice < limitprice:
                    quoteprice = limitprice
            else:
                quoteprice = lob['asks']['worst']
        order = Order(self.tid, otype, quoteprice, self.orders[0].qty, time, lob['QID'])
        self.lastquote = order
    return order
```

Credit:
https://gamerwall.pro/uploads/posts/2022-03/1647196483_15-gamerwall-pro-p-snaiper-budushchego-art-krasivie-oboi-17.jpg

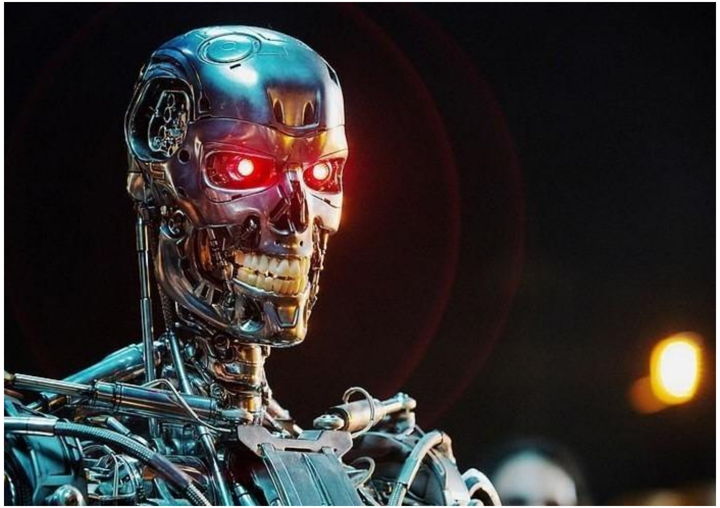
Trading robots: Zero-Intelligence Plus

Zero-Intelligence Plus (ZIP): A ZIP trader uses simple machine learning and a shallow heuristic decision tree to dynamically alter the margin that it aims to achieve on the order it is currently working. Robot managed to outperform human in controlled laboratory experiments

Credit:
https://wp-s.ru/wallpapers/15/19/366917070739812/nauchno-fantasticheskie-kartinki-s-cherepom-i-robotom.jpg

Trading robots: Adaptive-Aggressive

Adaptive-Aggressive (AA): Significantly improved ZIP algorithm with an aggressiveness variable that determines how quickly the trader alters its margin, and this variable is itself adaptively altered over time in response to events in the market. Shown to dominate prior algorithms and humans.



Credit: Terminator

BSE operation principles: Concept

BSE exchange market operates according to the following scheme:

1. The end time is selected and the clock time is set to zero.
2. At each step random agent is selected. He gets a chance to accept someone’s offer or make a bid.
3. All market agents are notified about new actions made by the selected agents and algorithm switches to step 2 until the current time is less than end time, otherwise the algorithm ends.

Altering supply and demand schedules in BSE

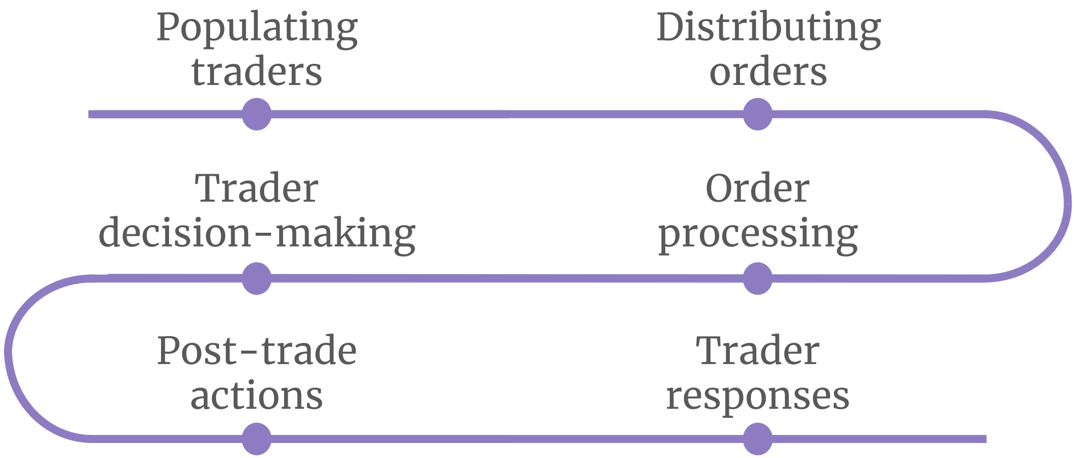
The current version of BSE can be configured to run the “traditional” economic experiments, switching between static supply/demand equilibrium and periodic simultaneous replenishments of orders. Additionally, it can be altered to continuous “drip-feed” replenishments. This is extremely helpful as most real-world markets, for much of the time, experience a continuous random feed of orders .

Further discussion

Trader bots in conventional trading simulators usually act as price-takers, so selling 1 share or 10 million shares at time t won't affect the historical price at time t+1.
At the same time, trader robots in BSE can be price-makers and impact the historical data, thus simulating market events that occur in real life.

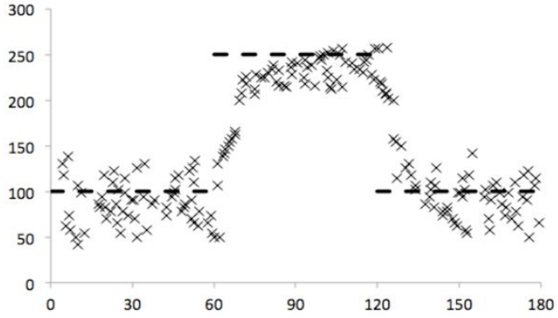
Despite that, BSE lacks latency which is present on real-life markets. The problem can be overcome by adding latency to LOB data or let the data issued by trader at time t not to arrive to the exchange until $t + \Delta t$.

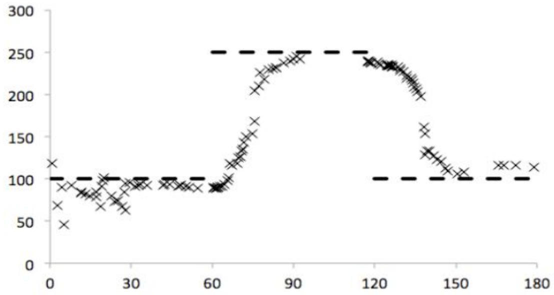
BSE operation principles: Market Session

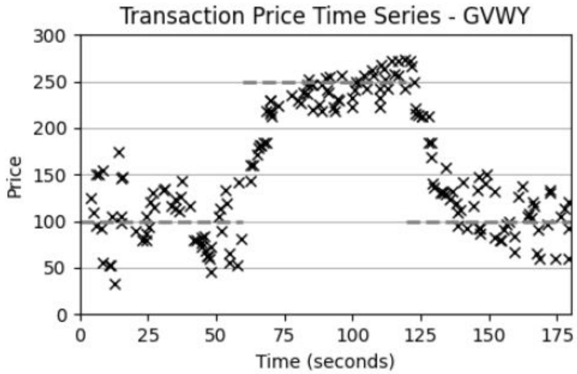


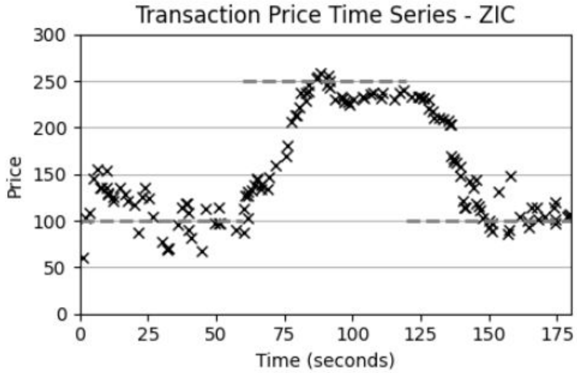

```python
traders = {('GVWY',10),('SHVR',10),('ZIC',10),('ZIP',10)}
time, endtime = 0, 180
timestep = 0

populate_market(n_traders, traders)

while time < endtime:
    duration = float(endtime - starttime)
    time_left = (endtime - time) / duration

    customer_orders(time, traders, order_schedule)

    tid = random_tid(traders)
    lob = exchange.publish_lob(time)

    order = traders[tid].gitorder(time, time_left, lob)

    if order != None:
        trade = exchange.process_order(time, order)

        lob = exchange.publish_lob(time)

        if trade != None:
            traders[trade['party1']].bookkeep(trade, order)
            traders[trade['party2']].bookkeep(trade, order)

            trade_stats(expid, traders, tdump, time, lob)

        lob = exchange.publish_lob(time)
        for t in traders:
            traders[t].respond(time, trade, lob)

    time = time + timestep
```

Experiments: Transaction Price Time Series







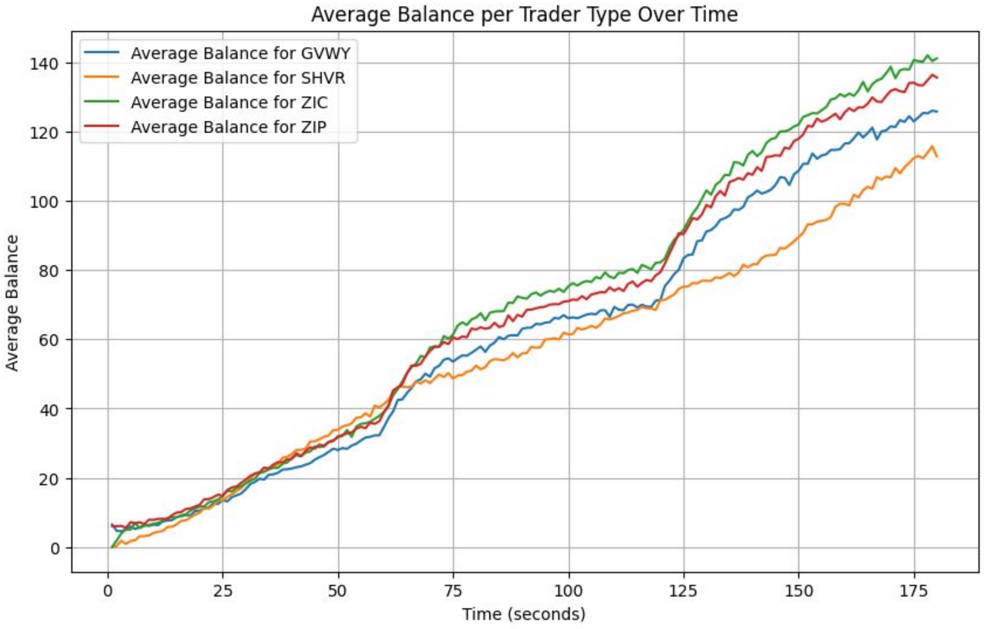


- Original experiment
- Our implementation

Experiments: Market Simulation

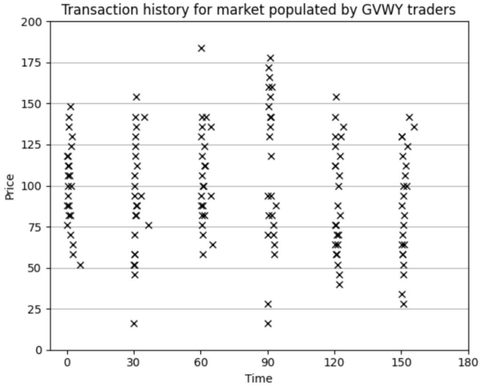
```
start_time = 0
start_time = 240

n_trader_types = 4
equal_ratio_n = 4
n_trials_per_ratio = 50
n_traders = n_trader_types * equal_ratio_n

fname = 'balances_%03d.csv' % equal_ratio_n

tdump=open(fname,'w')

min_n = 1

trialnumber = 1
trdr_1_n = min_n
while trdr_1_n <= n_traders:
        trdr_2_n = min_n
        while trdr_2_n <= n_traders - trdr_1_n:
                trdr_3_n = min_n
                while trdr_3_n <= n_traders - (trdr_1_n + trdr_2_n):
                        trdr_4_n = n_traders - (trdr_1_n + trdr_2_n + trdr_3_n)
                        if trdr_4_n >= min_n:
                                buyers_spec = [('GVWY', trdr_1_n),
                                               ('SHVR', trdr_2_n),
                                               ('ZIC',  trdr_3_n),
                                               ('ZIP',  trdr_4_n)]
                                sellers_spec = buyers_spec
                                traders_spec = {'sellers':sellers_spec,
                                                'buyers':buyers_spec}
                                print buyers_spec
                                trial = 1
                                while trial <= n_trials_per_ratio:
                                        trial_id = 'trial%07d' % trialnumber
                                        market_session(trial_id,
                                                       start_time, end_time,
                                                       traders_spec, order_sched,
                                                       tdump, False)
                                        tdump.flush()
                                        trial = trial + 1
                                        trialnumber = trialnumber + 1
                        trdr_3_n += 1
                trdr_2_n += 1
        trdr_1_n += 1
tdump.close()

print trialnumber-1
```

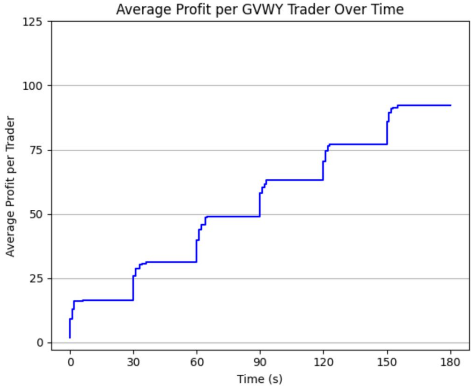
- Original pseudocode

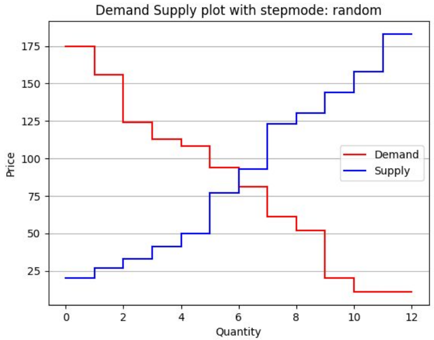


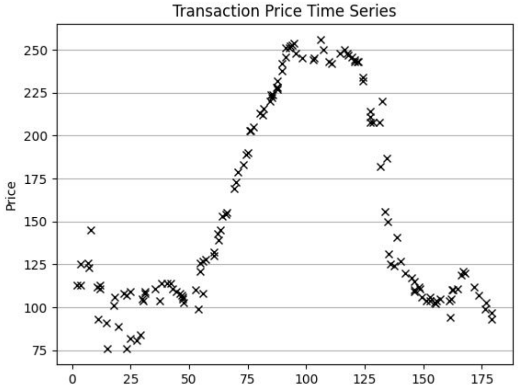
- Our Output

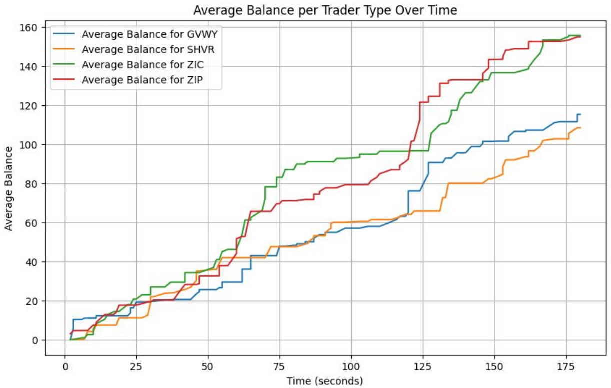
Extra outcome: BSE operation principles

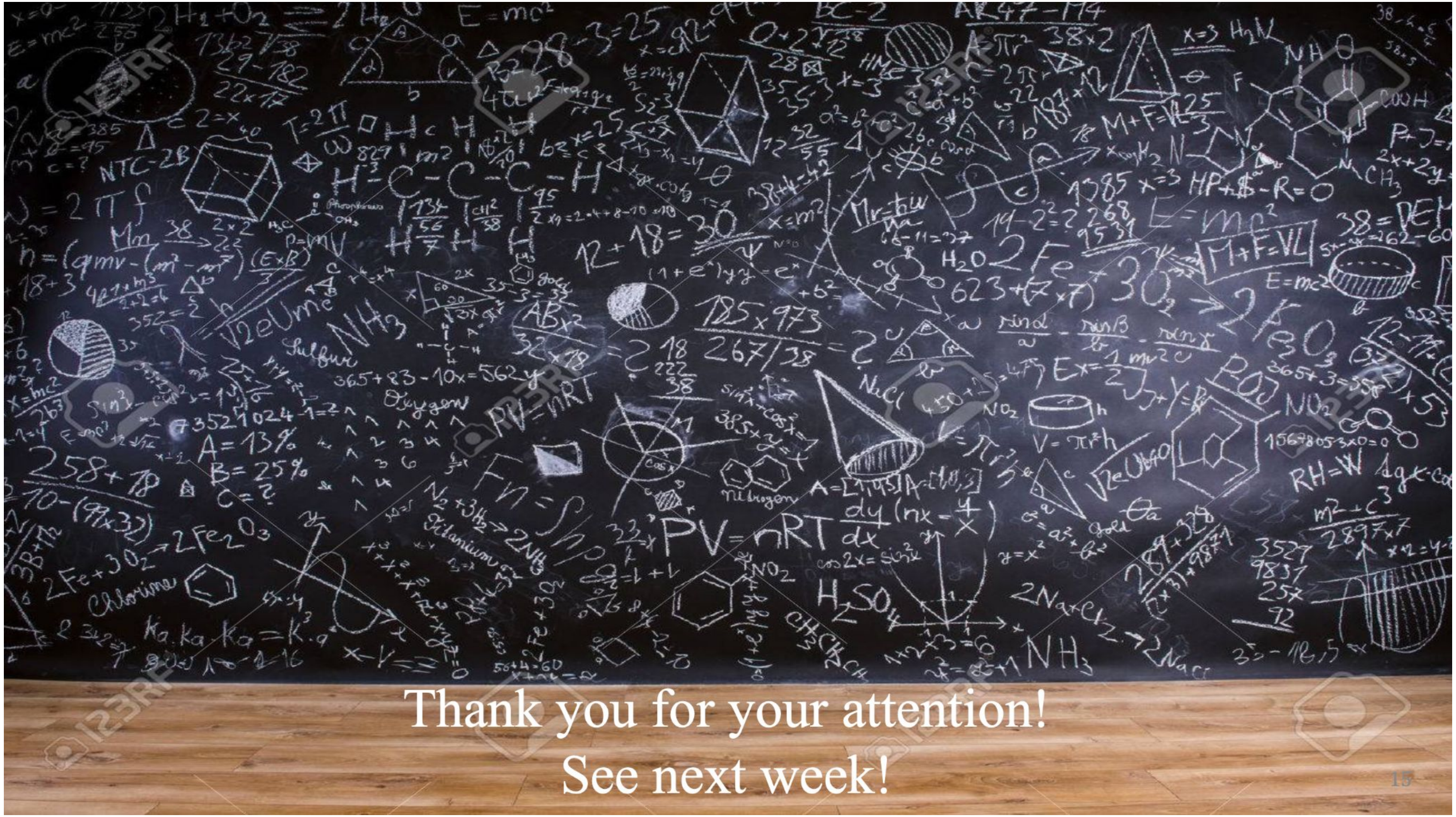












Thank you for your attention!

See next week!