Moving Average Crossover Market Maker in Bristol Stock Exchange (BSE)

Theodoros Constantinides

Department of Computer Science, University of Bristol

Abstract. In this report, we introduce a profitable Market Maker based on the moving average crossover trading strategy and test its performance using a variety of experiments in Bristol Stock Exchange.

Keywords: Market Maker · Crossover Trading Strategy · Bristol Stock Exchange · Limit Order Book · Financial Markets · Automated Trading

1 Introduction

In the last few decades, financial markets have been dominated by automated trading systems. This is not unexpected, as it has been almost 20 years since it was demonstrated that robot trades can outperform human traders[1]. In this paper, we introduce and analyse a Market Maker (MM), called Crossover Market Maker (COMM), that has been shown to consistently generate profit in different market scenarios on the Bristol Stock Exchange (BSE). COMM performs trades based on the moving average crossover trading strategy. In the context of this report, an MM is a trading robot that starts with a given amount of money and is allowed to perform trades in the market with the aim of making a profit by the end of each market session.

2 Bristol Stock Exchange

Bristol Stock Exchange, is a minimal simulation of a financial market that is based on a Limit Order Book (LOB). BSE provides many of the features we desire in a market simulation, namely it does not work with historical data, new transactions have a market impact and has many robot traders already build-in. However, this does not mean that it is an accurate simulation. There are a few simplifications, for once there is zero latency between traders and the exchange, there is no support for trades with quantity of more than one and there is only one tradable commodity.

The main differences of an MM and the other robots that currently exist in BSE, is that an MM can submit both bid and ask orders while existing trading robots only perform one of the two. Similarly, all other robots receive their limit prices as input and then trade to make a profit while an MM does not receive such input but has to observe the market and depending on its algorithm, try to make profitable trades.

3 Crossover Trading Strategy

Moving averages(MA), are indicators in technical analysis. There are many different types of MAs, with the most common ones being the simple(SMA) and exponential(EMA). SMA is the simplest and represents the average of the last n data-points. EMA is a weighted MA and so is influenced more by the latest data-points and less so by previous ones. SMA is usually used by long-term traders because it reacts slower to market change, while EMA reacts much faster and thus is typically used by short-term traders. There is no optimal choice of moving average, the choice depends on the market dynamics, the desired properties and the risk tolerance of each trader.

The moving average crossover technique, is a trading rule of technical analysis that can be used as a trading signal. In its simplest form, it is based on two moving averages, of any type, one that tracks a shorter period and another that tracks a longer period. Using this strategy, a buy signal is generated when the short MA crosses above the longer one. This is used to identify momentum in the market, as the price is moving up and is likely to continue moving up. When the short MA crosses below the long MA, a sell signal is generated as the price is falling and it is likely to continue falling in the following time periods. The periods selected for each MA are critical. Shorter periods, react faster to market changes but they also produce more false signals while longer ones produce fewer false signals but are slower to react[2].

This strategy, and other similar ones, have been shown to perform well in practice when looking at markets over time spans of tens or even hundreds of days[3]. The motivation behind our decision to implement this strategy for our MM, was to check if such a strategy which has been shown to work for much larger time frames could work in a high-frequency trading environment as the one we have in BSE. Instead of the closing prices of a market, we will be using the price of the latest trades as the input to the moving averages.

4 Analysis of the design of COMM

4.1 Design

COMM is designed to follow a constrained moving average crossover strategy similar to the one mentioned above. The main difference is that we define a profit and a loss percentage that are used to sell early. Those percentages are used to force COMM to abandon the crossover strategy and instantly sell at market price. These constrains, are added in an attempt to stay ahead of the market and sell if we made enough profit and potentially avoid the burst of a price bubble or to sell early if the market is crashing and the price is rapidly falling down, taking a small loss that is accepted to avoid an even greater loss. Of course, this means that sometimes COMM will sell before the price reaches its maximum, and so it will lose some potential profit and other times COMM will sell at a loss even though the market could quickly recover. This is a natural trade-off, as it reflects the logic that a human trader could use in a similar

situation and so we thought that it would make a nice addition to the normal crossover strategy.

The main logic of COMM, is implemented in the respond method and therefore COMM adjusts its orders after every successful trade. In more detail, COMM reads the latest trades from the LOB's tape and finds the new moving averages. Then COMM compares the new moving averages with the previous ones to determine if a crossover occurred, and if it occurred it reacts to it by either preparing a new bid or a new ask order to be submitted to the exchange. If a crossover has not occurred, COMM will check to see if it is making enough profit or if it is losing too much and will again update its order accordingly and wait for it to be submitted to the exchange when the get_order method is called. Lastly, if the bookkeep method is called, it means that COMM was involved in a trade and so it updates its internal state, by either increasing its balance and reducing the amount of shares it owns or decreasing its balance and increasing its shares to reflect the trade that just took place.

COMM can be customised in a few different ways as mentioned before. There are three different types of moving averages, exponential, weighted and simple. Related to the MAs, the window sizes (number of latest trades to take into account) for the short and long MA can also be set by the user. Starting with the SMA, this just computes the average of the last n trades, with n either being the value of the window of the short MA, or the value of the window for the long MA. Then, there is also the EMA which is defined as:

$$\begin{aligned} &\text{if t = 1:} && EMA_1 = p_1 \\ &\text{else:} && EMA_t = a * p_t + (1 - a) * EMA_{t-1} \end{aligned}$$

$$\text{where } p_i \text{ is the price of the } i^{th} \text{ trade, and } a = \frac{2}{n+1} \text{ for given n}$$

Lastly, the weighted moving average (WMA) is defined as:

$$WMA = \frac{np_m + (n-1)p_{m-1} + ... + 2p_{m-n+2} + p_{m-n-1}}{n + (n-1) + ... + 2 + 1}$$

As mentioned before, EMAs react faster to market changes than SMAs and the WMA should be somewhere in the middle of the two in terms of reaction time. For this reason, as a default COMM will be using EMA but depending on the experiment this could change for better performance.

The other major customisation, is the type of trader. This can be 1, 2 or 3 with 1 corresponding to a trader that works like a shaver, 2 corresponding to a trader that prices its orders at one of the two MAs and lastly 3 which corresponds to a trader that always buys or sells at the current market price. One on hand, the latter could potentially miss some of the profits that the other two might make but should in theory make more trades as its orders will always execute (given that there was not a change in the best LOB bid or ask from the time respond was called until the time that get_order will be called). One the other

hand, type 1 will always try to buy or sell when the strategy dictates but will try to do so by giving a new best price that is maximised (either lower best ask by one or increase best bid by one) and hoping someone will take its offer. Type 2 represents a strategy that lies in the middle of the other two, as its price is likely to be between the prices of the other two and so it will not make as much profit per trade as type 1 but will make more than type 3, and will also make fewer trades than type 3 but not as few as type 1.

There is an additional constrain that affects the price set by all three types of traders. The price of orders is constrained so that COMM does not loss money (this only happens when we are in the crossover strategy and not when we are selling to avoid additional loss). For bids, this basically limits the bid price to be lower than the last price COMM sold at and for asks, this limits the ask price to be higher than the last price COMM bought at.

4.2 Hyperparameters

Before the execution of an experiment, the user should set up some hyperparameters to specify the exact behaviour of COMM. Those are:

1. Trader type: 1(Shaver-like), 2(Current MA as price), 3(Market price)
2. Moving average type: 'E'(EMA), 'W'(WMA), 'S'(SMA)
3. Short: The window of the short MA, how many trades to look back when calculating the short MA
4. Long: The window of the long MA, how many trades to look back when calculating the long MA
5. Profit: Target profit as a percentage
6. Loss: Loss tolerance as a percentage

4.3 Computational Performance

Care was taken, not to make COMM too computationally demanding. It works based on a decision tree and only looks at the last few items on the LOB. With our selection of hyperparameters, we measured the time it takes for COMM's respond method to run, and we found that it took on average less than 5μs on our machine (Intel i5-4690). This is comparable with ZIP's performance[4]. Although the computational performance of COMM is not important in BSE, it could be the case that it is important in a real world experiment.

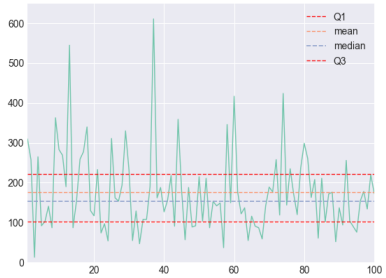
5 Hyperparameter Tuning

With the design of COMM done, it is now time to tune its hyperparameters to maximise COMM's performance. A restricted grid search was used, searching for the values that maximised the average profit made by COMM. We kept the experiment constant, fixed a subset of hyperparameters, and exhaustively tuned the remaining. Then we fixed the tuned hyperparameters to the values that produced the optimal results and tuned the remaining hypeparameters. Without

this restriction, the number of different tests needed to tune the hyperparameters would have been 56715, but with the restriction the number dropped to 6309. The split we choose was to first test all trader types and all MAs (9 tests) while keeping the rest fixed. Then, we set the trader_type=1 and the ma_type='E' which were optimal, and tuned the other four hyperparameters (6300 tests). This split was chosen because during the development of COMM we observed that both trader type 1 and EMA were constantly giving much better results, so we were fairly confident that in most scenarios they would produce the optimal outcome. For each test, the environment was kept the same and 10 markets sessions were performed each lasting 10 minutes using the M_1 market that will be described later. This again reduces the generality, but it was done to reduce the number of tests to a manageable level, since if we were to include different experiments as well, the number of tests would have grown exponentially. To help generalise, a list of the fifteen best hyperparameter values was kept, and afterwards they were tested in a more dynamic experiment environment, one introducing a market shock in the middle of each trial and the best performing was selected (15 more tests). The set of hyperparameters that maximised the final balance is shown in Table 1, along with a graph showing the profit made by COMM on 100 runs of market M_1. Tuning took around ten hours on a desktop computer using an Intel i5-4690 processor. It is recommended that users alter this preset to more accurately reflect the markets they are transacting in and to fit their specific trading style, so the values here are just given as a reference and for the replication of our results.

Table 1. Tuned Hyperparameters

Parameter	Range Tested	Optimal
trader_type	(1, 2, 3)	1
ma_type	('E', 'W', 'S')	'E'
short	[2, 8]	2
long	[3, 14]	8
profit	[1.1, 2.0] step=0.1	1.6
loss	[0.1, 1.0] step=0.1	0.7



6 Experiments

6.1 Experimental Setup

In order to test the performance of COMM, we used a few different experiments, following the methods used by Vytelingum[5] and later Snashall and Cliff[4]. We first experimented using static markets, where the supply schedule(SS) and demand schedule(DS) remained constant for the duration of each market session and then using dynamic markets, where the supply and demand schedule(SDS) changes at some point in the market session introducing a market shock. Additionally, for the four static markets, we run experiments using the offset function

provided in BSE. The offset function, is a sine wave that grows in amplitude and decreases in wavelength over time. For all experiments, we simulated 100 market sessions each lasting 15 market minutes. Each experiment, consisted of COMM and 16 other traders, half of them sellers and half buyers. Initially, we had 8 of each kind (GVWY, SHVR, ZIP, ZIC) but then experiments with different proportions were performed as well. We performed all experiments with all 3 available stepmodes (fixed, jittered, random).

6.2 Supply and Demand Curves

Static markets: We will be using four static markets with different SDS. Equilibrium prices (eq) are for the case of fixed stepmode. Those are:

1. M_1: SS: (50, 150), DS: (50,150), eq: 100,
2. M_2: This market has elastic supply, SS: (75, 125), DS: (50, 150), eq: 100,
3. M_3: This market has elastic demand, SS: (50, 150), DS: (75, 125), eq: 100,
4. M_4: M1: SS: (150, 250), DS: (150,250), eq: 200.

Dynamic Markets: We will be introducing shocks to the markets by switching from one SDS to another and in some cases back to the first. For example we denote M_{12}, as the market that starts as M_1 and then half way through the experiment switches to M_2. In the same way, M_{121} will denote the market that starts as M_1, then switches to M_2 and then back to M_1. M_1' denotes the market that has the same SDS as M_1 but with an offset function.

6.3 Results

In general it seems that COMM is able to make a profit in most cases, our full results are shown in the Table 2. The average value was chosen instead of other data aggregates since it reflects the performance that someone running COMM for multiple sessions could expect to get at the end.

Of interest are the cases when COMM is losing money. These are highlighted in Table 2 and correspond to M_{41}, M_{42} and M_{43}. This is expected though, as in these cases we have a market crash and since COMM owns shares in the market it is expected to lose money. The amount we are losing is relatively small compared to the amount of profit in other cases. Note that fixed and jittered stepmodes are affected, if random stepmode is used, COMM is making a small profit (M_{41} and M_{43}) or almost breaking even (M_{42}). The reverse, a sudden jump in prices, is interesting as well as those are the cases where COMM is making the most profit (M_{14} and M_{34} - highlighted in the table) for all stepmodes. M_{24} is also one of the most profitable configurations but is considerably less than the other two. Before the shock, the supply is elastic so it is harder for COMM to buy shares since fewer crossover events will occur, and the shares it buys will be at a lower price. So, after the shock, when COMM is likely to want to sell, it will own less shares and will make less profit from each one.

Table 2. Average Profit of COMM for each Market and Stepmode

Market	Fixed	Jittered	Random
M_1	176.72	188.44	306.98
M_2	108.67	118.97	198.12
M_3	124.13	132.11	199.56
M_4	135.5	153.13	263.58
M_1'	327.99	347.54	395.85
M_2'	263.86	278.93	294.85
M_3'	301.92	320.13	355.01
M_4'	216.74	219.87	262.73
M_{12}	132.7	142.71	249.23
M_{13}	152.97	145.74	248.65
M_{14}	**250.25**	**287.74**	**403.65**
M_{21}	158.3	160.23	261.9
M_{23}	119.47	132.35	204.23
M_{24}	170.02	197.61	243.85
M_{31}	149.79	154.22	265.88
M_{32}	124.39	142.98	204.36

Market	Fixed	Jittered	Random
M_{34}	**280.8**	**326.98**	**439.32**
M_{41}	***-53.44***	***-44.44***	22.34
M_{42}	***-62.06***	***-75.45***	***-0.7***
M_{43}	***-61.16***	***-53.98***	22.5
M_{121}	153.18	164.18	265.55
M_{131}	149.41	181.69	275.51
M_{141}	113.7	138.87	187.74
M_{212}	148.58	148.04	249.59
M_{232}	120.32	131.85	201.3
M_{242}	53.28	61.93	112.04
M_{313}	140.35	163.99	240.37
M_{323}	120	136.54	195.69
M_{343}	92.37	100.8	174.07
M_{414}	69.72	51.8	155.68
M_{424}	10.83	17.34	97.17
M_{434}	42.52	59.02	156.81

Using M_1, we also performed experiments with varying ratios of traders. In those experiments we were mostly interested in the cases where one type of trader dominates the market. To reduce the number of experiments we assume all other trader types, apart from the one that dominates, to be equally numbered as shown in Table 3. As the percentage of traders approaches a uniform split, the profits of COMM seem to be converging to the profits we saw in Table 2.

Table 3. Average Profit of COMM for ratios of (ZIP,ZIC,SHVR,GVWY) in M_1

Ratio	Fixed	Jittered	Random
0:0:0:16	1692.12	1452.69	1230.99
0:0:16:0	56.77	52.14	35.15
0:16:0:0	382.01	374.87	300.0
16:0:0:0	**-1.63**	**-17.58**	326.51
1:1:1:13	1090.08	974.24	912.91
1:1:13:1	70.61	61.53	56.88
1:13:1:1	463.3	467.83	421.14
13:1:1:1	**-23.38**	7.29	323.43

Ratio	Fixed	Jittered	Random
2:2:2:10	612.82	570.6	658.07
2:2:10:2	81.33	80.9	89.37
2:10:2:2	381.23	382.16	430.0
10:2:2:2	6.43	44.8	324.23
3:3:3:7	331.71	351.94	449.5
3:3:7:3	121.61	136.19	163.9
3:7:3:3	261.4	271.67	375.03
7:3:3:3	73.5	104.03	316.92

It is clear, that COMM performs best against higher percentages of GVWY, which is to be expected. Against a market dominated by SHVRs, COMM consistently makes only a small profit for all 3 stepmodes. This is due to the similarity in they way COMM type 1 and a SHVR price their orders. In a market that contains only SHVRs and COMM, COMM is only able to transact if a trade is made before some other SHVR has the chance to make a better offer.

The profit earned by COMM is consistent in markets dominated by ZIC traders across all stepmodes. When the stepmode is fixed, COMM is unable to make a profit against a market dominated by ZIP traders, but makes a small loss, this however does not hold true when the stepmode is random. This is again expected as ZIP traders rapidly converge to the equilibrium where COMM is unable to make a profit, and if it bought shares at a bad price before the convergence, then a loss is made. When random stepmode is introduced however, the equilibrium is not constant, and this allows COMM to make a profit.

6.4 Statistical Tests

By observing Table 2, it seems like COMM is more profitable when the stepmode is random. In order to test this, we will be using the non-parametric Friedman Test, which works with data from more than two samples. Testing all markets in Table 2 and using an alpha value of 0.05, we can see that the difference between each pair is significant for all markets except M_2'. So this tells us that COMM is statistically more profitable when the stepmode is random compared to one of the other stepmodes, except for market M_2'.

We can also see similarities between the the results when stepmode is fixed and jittered. We performed the Mann-Whitney U Test with an alpha value of 0.05 to test the significance of the difference between the two for all markets in Table 2. Our results show that in most markets the difference between them is insignificant, except M_{23}, M_{32}, M_{34}, M_{131}, M_{141}, M_{313} and M_{323}.

7 Conclusion

In total we run 14440 market sessions totalling 216000 market minuets to test the performance of COMM. In general, it seems like the MM we are proposing has some good qualities and is able to consistently produce profit in most situations, especially when random stepmode is used. We believe that this a good start, but additional experiments with dynamic markets are needed to to evaluate its performance further to determine if COMM could be used in real world markets.

References

[1] Rajarshi Das et al. “Agent-Human Interactions in the Continuous Double Auction”. In: *IJCAI International Joint Conference on Artificial Intelligence* (May 2001).

[2] Ihab A El-Khodary. *A decision support system for technical analysis of financial markets based on the moving average crossover*. 2009.

[3] Ikhlaas Gurrib. *The Moving Average Crossover Strategy: Does It Work for the S&P500 Market Index?* 2016.

[4] Daniel Snashall and Dave Cliff. *Adaptive-Aggressive Traders Don't Dominate*. 2019. arXiv: `1910.09947 [q-fin.TR]`.

[5] Perukrishnen Vytelingum. “The Structure and Behaviour of the Continuous Double Auction”. 2006.

8 Appendix: Python Code of COMM class

```
#This market maker uses moving averages and the crossover
#method to decide when to transact in the market
class Trader_COMM(Trader):
    def __init__(self, ttype, tid, balance, time):
        super().__init__(ttype, tid, balance, time)
        # ----- hyperparameters -----
        self.type = 1 #type is 1(shaver-like), 2(bid/ask at avg) or
            3(bid/ask at market)
        self.ma_type = 'E' #E: exponential, W: weighted, S: simple
        self.short = 2 #short moving average window
        self.long = 8 #long moving average window
        self.profit = 1.6 #target profit
        self.loss = 0.7 #max loss willing to take
        # ---------------------------
        self.mem = (0, 0)
        self.bid_order = None
        self.ask_order = None
        self.latest = 0
        self.own = 0
        self.last_purchase_price = None
        self.last_sell_price = None
        self.net_worth = 0

    def respond(self, time, lob, trade, verbose):
        mvaShort = 0
        countShort = 0
        mvaLong = 0
        countLong = 0

        for t in reversed(lob["tape"]):
            if t['type'] == 'Trade':
                if countShort == 0:
                    self.latest = t['price']

                if countShort < self.short:
                    if self.ma_type == 'W':
                        mvaShort += t['price']*(self.short-countShort)
                    elif self.ma_type == 'S':
                        mvaShort += t['price']
                    countShort += 1

                if countLong < self.long:
                    if self.ma_type == 'W':
                        mvaLong += t['price']*(self.long-countLong)
                    elif self.ma_type == 'S':
                        mvaLong += t['price']
                    countLong += 1
```

```
            else:
                break

        #if no trades happened set averages to None, don't send order
        if countShort == 0:
            mvaShort = None
            mvaLong = None
            # we don't have enough information to buy or to sell
            # wait for more trades to gain inside, then trade
        else:
            #find moving averages
            if self.ma_type == 'E':
                if self.mem == (0, 0): #first trade
                    mvaShort = self.latest
                    mvaLong = self.latest
                else:
                    facShort = 2/(1+self.short)
                    facLong = 2/(1+self.long)
                    mvaShort = self.latest*facShort +
                         self.mem[0]*(1-facShort)
                    mvaLong = self.latest*facLong + self.mem[1]*(1-facLong)
            elif self.ma_type == 'W':
                denShort = (countShort*(countShort+1))/2
                denLong = (countLong*(countLong+1))/2
                mvaShort = mvaShort/denShort
                mvaLong = mvaLong/denLong
            elif self.ma_type == 'S':
                mvaShort = mvaShort/countShort
                mvaLong = mvaLong/countLong
            else:
                sys.exit('Wrong moving average type.')

            if self.mem != (mvaShort, mvaLong): #if averages have changed
                 (trade happened)
                mvaShortPr = self.mem[0] #store previous value of short
                mvaLongPr = self.mem[1] #store previous value of long
                self.mem = (mvaShort, mvaLong) #update values

                #crossover method
                if mvaShort > mvaLong: #check if crossing upwards
                    if mvaShortPr <= mvaLongPr:
                        #since price is crossing upwards, then it's good
                             time to buy
                        if self.type == 1: #type 1 works like shaver
                            if lob['bids']['best'] == None:
                                p = 1
                            else:
                                p = lob['bids']['best'] + 1
                        elif self.type == 2: #type 2 orders at the current
                             average
```

```
                    p = mvaLong
                elif self.type == 3: #type 3 just buys at market
                    price
                    if lob['asks']['best'] == None:
                        p = 1
                    else:
                        p = lob['asks']['best'] + 1
                else:
                    sys.exit('Wrong trade type.')

                if not self.last_sell_price == None:
                    if p > self.last_sell_price: #make sure we
                        don't lose money
                        p = self.last_sell_price

                if self.balance >= p:
                    oprice = p
                else: #if you can't make a better bid than the
                    best, make the best you can
                    oprice = self.balance
                self.bid_order = Order(self.tid, 'Bid', oprice, 1,
                    time, lob['QID'])

                if self.ask_order == None:
                    self.orders = [self.bid_order]
                else:
                    self.orders = [self.bid_order, self.ask_order]
            elif mvaShort < mvaLong: #check if crossing downwards
                if mvaShortPr >= mvaLongPr:
                    #since price is crossing downwards, then it's a
                        good time to sell
                    if self.own >= 1:
                        if self.type == 1: #type 1 makes a better ask
                            than the current best
                            if lob['asks']['best'] == None:
                                p = 999
                            else:
                                p = lob['asks']['best'] - 1
                        elif self.type == 2: #type 2 orders at the
                            current average
                            p = mvaShort
                        elif self.type == 3: #type 3 makes a market
                            order
                            if lob['bids']['best'] == None:
                                p = 999
                            else:
                                p = lob['bids']['best']
                        else:
                            sys.exit('Wrong trade type.')
```

```
                        if not self.last_purchase_price == None:
                            if p < self.last_purchase_price: #make sure
                                we don't lose money
                                p = self.last_purchase_price
                        self.ask_order = Order(self.tid, 'Ask', p, 1,
                            time, lob['QID'])

                        if self.bid_order == None:
                            self.orders = [self.ask_order]
                        else:
                            self.orders = [self.ask_order,
                                self.bid_order]
                elif self.own > 0: #sell if you have enough profit/loss
                    if not lob['bids']['best'] == None:
                        if self.last_purchase_price*self.profit >=
                            lob['bids']['best']:
                            self.ask_order = Order(self.tid, 'Ask',
                                lob['bids']['best'], 1, time, lob['QID'])
                            if self.bid_order == None:
                                self.orders = [self.ask_order]
                            else:
                                self.orders = [self.ask_order,
                                    self.bid_order]
                        elif self.last_purchase_price*self.loss >=
                            mvaShort:
                            self.ask_order = Order(self.tid, 'Ask',
                                lob['bids']['best'], 1, time, lob['QID'])
                            if self.bid_order == None:
                                self.orders = [self.ask_order]
                            else:
                                self.orders = [self.ask_order,
                                    self.bid_order]

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

    def bookkeep(self, trade, order, verbose, time):
        self.blotter.append(trade) # add trade record to trader's blotter
        #find order in my orders
        if len(self.orders) == 1: #only one order
            i = 0
        elif order.tid == self.tid: #this is my order
            if order.otype == 'Bid':
                if self.orders[0].otype == 'Bid': i = 0
```

```
            else: i = 1
        elif order.otype == 'Ask':
            if self.orders[0].otype == 'Ask': i = 0
            else: i = 1
    else: #it's someone else order
        if order.otype == 'Bid':
            if self.orders[0].otype == 'Ask': i = 0
            else: i = 1
        elif order.otype == 'Ask':
            if self.orders[0].otype == 'Bid': i = 0
            else: i = 1

    if self.orders[i].otype == 'Bid':
        self.bid_order = None
        self.balance -= trade['price']
        self.last_purchase_price = trade['price']
        self.own += 1
    elif self.orders[i].otype == 'Ask':
        self.ask_order = None
        self.balance += trade['price']
        self.last_sell_price = trade['price']
        self.own -= 1

    self.n_trades += 1

    self.net_worth = self.balance + self.last_purchase_price *
        self.own

    if verbose:
        outstr = ""
        for order in self.orders:
            outstr = outstr + str(order)
        if self.orders[i].otype == 'Bid': # We bought some shares
            outcome = "Bght"
        else: # We sold some shares
            outcome = "Sold"
        print('%s, %s=%d; Qty=%d; Balance=%d, NetWorth=%d'
            %(outstr, outcome, trade['price'], self.own,
                self.balance, self.net_worth))
    self.del_order(self.orders[i]) # delete the order
    #print(self.orders)

def del_order(self, order):
    self.orders.remove(order)
```