Designing and Performance Testing an Automated Market Maker in BSE

Angus M. F. Parsonson

University of Bristol, Bristol BS8 1TL, UK
ap17691@bristol.ac.uk
https://angusparsonson.github.io

Abstract. In this paper, I will discuss an approach to implementing an automated market maker using a range of technical trading indicators in combination with Cliff's ZIP algorithm [1]. The goal will be to maximise profit (or net-worth) over a trading period in a simulation of a limit-order-book financial exchange. This trader will aptly be named ZIPMM; and the exchange simulation I will be using is BSE [2]. I will then present the results from various performance tests of ZIPMM, including with altering market price trends and altering trader agent populations, to best gauge its quality.

Keywords: Automated · Market Maker · ZIP.

1 Trader Design

1.1 Introduction

ZIPMM consists of two parts; the entrance and exit point algorithm, and the sales trader. Respectively, their jobs are to decide the best time in the market to buy or sell any number of stock (to maximise profit/net worth) and to choose at what price to buy or sell that stock (to maximise the profit margin). The entrance and exit point algorithm uses a number of technical trading indicators to make its choice. The sales trader algorithm uses Cliff's ZIP algorithm from Cliff '97, which (along with MGD) was among the first trader agents to be proven by IBM in 2001 to outperform human traders in LOB-based continuous double auction markets. This section will focus on outlining both of these parts in detail.

1.2 Technical Indicators

Stochastic Oscillator The most prominent indicator in ZIPMM is a stochastic oscillator. This is a momentum indicator which attempts to generate overbought and oversold signals in a market. ZIPMM looks to buy in an oversold market and sell in an overbought market. The following formula can be used to calculate the stochastic oscillator of a period:

$$\%K = \left(\frac{C - L14}{H14 - L14} \right) \times 100 \tag{1}$$

where:
C = The most recent trade price
L14 = The lowest price traded of the previous 14
H14 = The highest price traded of the previous 14
%K = The current value of the stochastic indicator

%K is known as the fast stochastic oscillator. The oscillator produces a value between 0 and 100 where values below 20 are often considered oversold, and values above 80 are considered overbought. A slow stochastic oscillator (%D) is also calculated as the 6-price moving average of the oscillator. When %D is below 20 and %K moves above %D, a buy signal is triggered as this indicates a reversal from an overbought market (see Fig. 1). Using a relatively short moving average period seemed to give the best results in testing. The candlestick chart visualises the values used for the oscillator calculation (see Fig. 2).

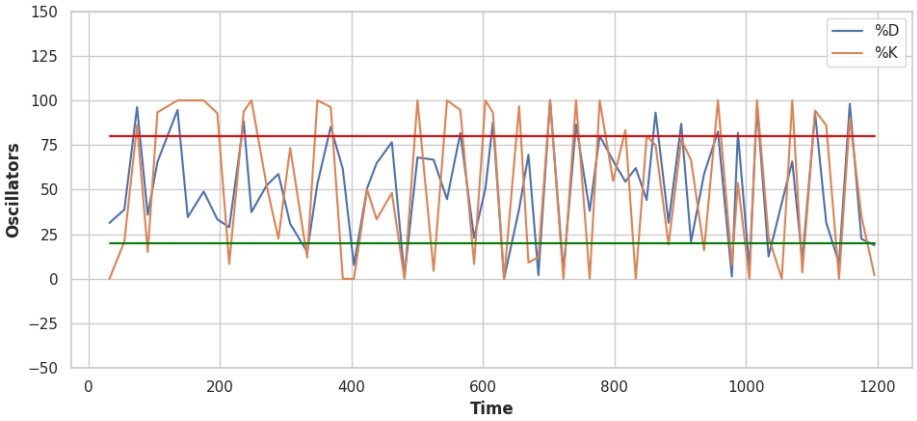

Fig. 1. Slow and fast stochastic oscillator values for every 15th 14-price period of a BSE market session. The buy line is shown in green and the sell line is shown in red.

Simple Moving Average (SMA) Crossover The stochastic oscillator is a useful tool, however; often it can produce false signals. It also tended to give early buy signals, meaning ZIPMM would buy before a price drop and have to hold onto a number of falling stock for a considerable time before they moved back to a profitable position. In order to both slow and improve the signal, I added another indicator using simple moving averages. SMA was used as opposed to exponentially weighted moving averages because EMAs are more responsive to recent prices, meaning they wouldn't slow the signals as much as SMAs. An SMA is nothing more than the average price over a certain number of recent trade prices. To generate a buy signal, I used a long-term SMA over a 3-price period and a short-term SMA of the current price. The short period sizes were

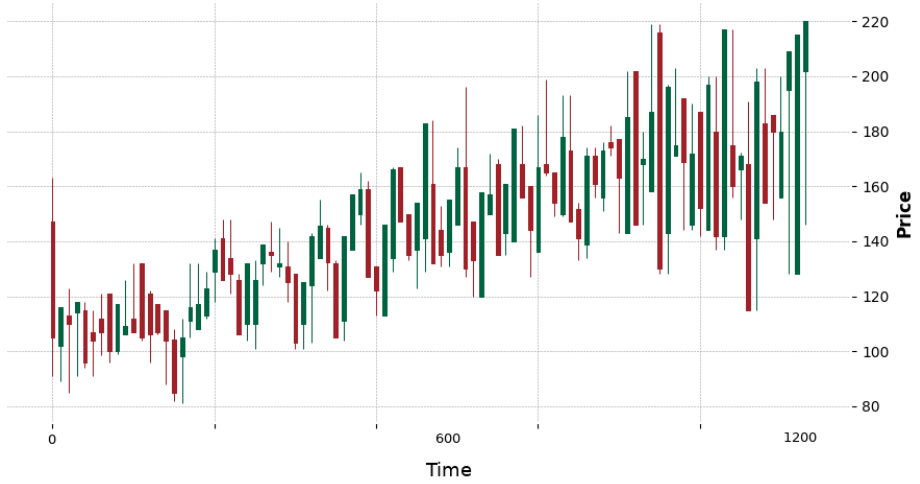

Fig. 2. Candlestick chart for every 15th 14-price period of a market session. This illustrates the start, end, highest and lowest prices of the period. All of these prices (with the exception of the start price) are used in the stochastic oscillator calculation.

used because I still wanted the stochastic oscillator to be the dominant indicator. When the short SMA moves above the long SMA, this indicates an up-trend in the market, and can be used as a buy signal.

Bollinger Bands When I implemented the functionality for ZIPMM to hold an inventory of multiple stock, a problem I faced was deciding when to buy and sell the stock which ZIPMM was currently holding. Both SMA and the stochastic oscillator seemed not to produce reliable sell signals from the *bookkeep* method. In order to help with this I used Bollinger Bands. These work by graphing three lines (bands) using trade prices (see Fig. 3). The middle band represents the 14-price period SMA of the recent trade prices, the upper and lower bands lie two standard deviations above and below the middle band respectively. Due to the relatively short price period, I took the outer bands to be one standard deviation from the middle. The theory is that prices tend to rebound back to their mean after they reach unusual highs or lows, a process known as mean reversion. In light of this, when ZIPMM is already holding one or more stock in its inventory, it will choose to buy more if the current trade price is equal to or below the lower band, and sell if not. This approach seemed to give a reasonable number of sell signals to ZIPMM, meaning it rarely holds on to stock for too long.

Exponential Moving Average When ZIPMM identified what it believed to be a good entry point to the market, it was originally using the most recent trade price as a buy limit price. This method is weak because the recent trade price is often not the best estimate of the current market price, so the limit

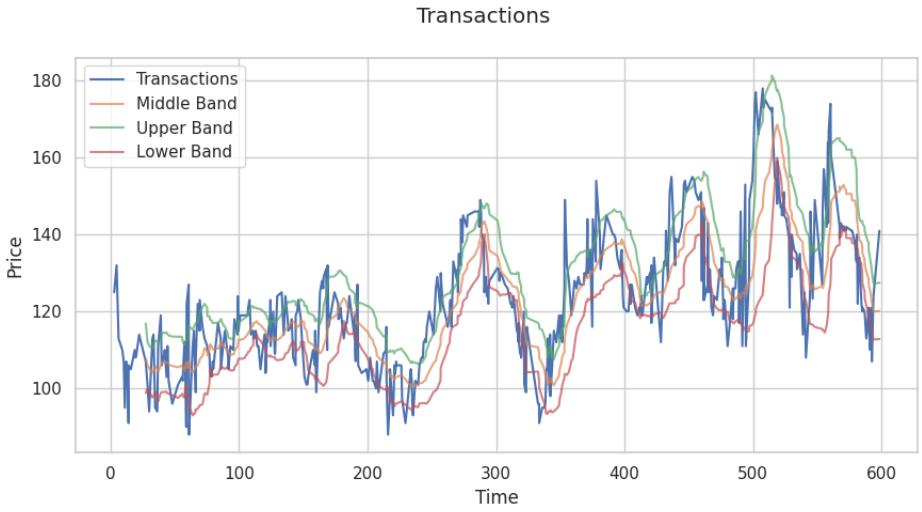

Fig. 3. All three Bollinger Bands for every trade price over a 600 time step period.

price may be higher than it needs to be, or too low to succeed. To address this issue of deciding a buy limit price for the ZIP algorithm, I used an estimate of the current market equilibrium price - similar to an Adaptive-Aggressive trader. Like AA, this was calculated using an exponential moving average, which I took over the last 14 trade prices. EMA gives a more accurate depiction of the current equilibrium than SMA because it gives more influence to the most recent trades. It is calculated using the following equation (with a smoothing factor of two):

$$EMA = \left(P_{today} \times \frac{smoothing}{1 + window}\right) + EMA_{yesterday} \times \left(1 - \left(\frac{smoothing}{1 + window}\right)\right) \tag{2}$$

1.3 ZIP

DIMM01 used a constant bid delta to decide its price margin (difference between buy/sell price and the limit price). This meant it had no functionality to adapt its margin according to market price conditions in order to make a successful trade. This is a clear weakness which I felt needed addressing. The ZIP algorithm helps to mitigate this weakness by adapting its profit margin according to the behaviour of other traders. For example, if you have an item to sell at price P and either; sellers are accepting bids below P or, sellers are making offers below P, then decrease the profit margin M. If trades are happening above P, then increase M. Buyers will do the inverse of this. The profit margin is adapted towards the target price using the Widrow-Hoff learning rule with momentum.

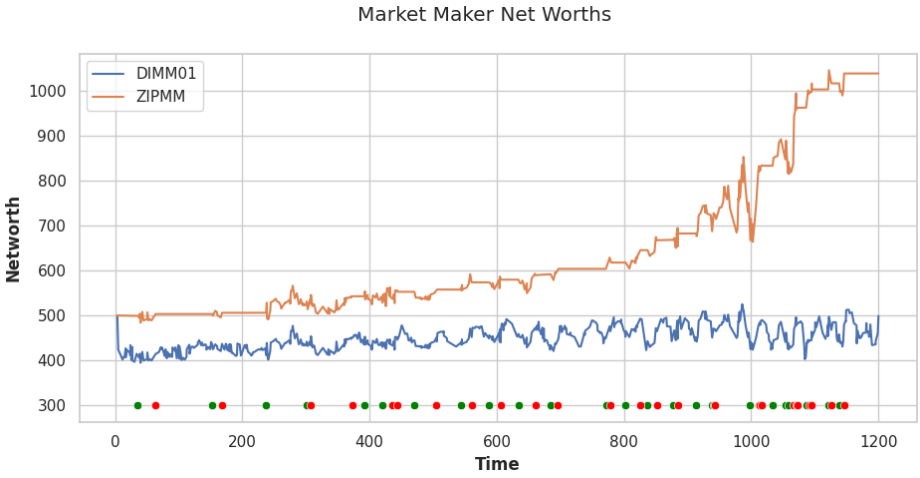

Fig. 4. The net worth of ZIPMM and DIMM01 over a 1200 time step trading period. The net worths are calculated by summing a trader's balance and the number of stock it has at the current market price (most recently traded price). The green points represent a successful buy trade for ZIPMM and the red points represent a sell.

2 Performance Experiments

2.1 Overview

In order to test ZIPMM's performance, I ran a number of experiments in BSE. These included changing market price trends and sales trader agent populations. I used these experiments throughout development to test the incremental changes I was making to the algorithm, often a technique that seemed good to me on paper would underperform in reality and have to be scrapped. Doing this also allowed me to tweak the various parameters required for ZIPMM's algorithms (price period lengths etc.), in order to achieve the best performance. In an ideal scenario, these experiments would include direct comparisons to other automated market makers, for example one using AA or MGD. However, due to the limited time available to me in this coursework, I thought it best to work on one trader and therefore didn't have any other market maker implementations to test against.

2.2 Experiments

2.3 Altering Market Price Trends

For the experiments shown in Table 1. - market equilibria are altered by changing how sales trader limit prices are selected (changing the supply and demand curves). I defined a ranging market as one where the prices are selected randomly from a uniform distribution. In the bull/bear markets, the price equilibrium

is steadily rising/falling using a sine wave offset function at time t, the wave grows in amplitude and reduces its wavelength over time. The shock markets incorporate shock changes to the equilibrium of either a sharp rise (U) or sharp fall (D). The drunkards walk is a bull market with a random uniform addition or subtraction from the wavelength, amplitude and gradient of the offset function.

Table 1. The average net worth of ZIPMM compared to DIMM01 for different types of price trend. These results were taken over ten market sessions each lasting 1000 time steps. The average net worth was taken every five time steps. Shock trends consisted of two shocks in a particular direction, either up (U) or down (D).

	DIMM01			**ZIPMM**		
Trend	Start	Peak	End	Start	Peak	End
Ranging	500.00	500.00	**431.14**	500.00	622.00	**615.55**
Bull	500.00	534.99	**499.54**	500.00	843.27	**829.06**
Bear	500.00	500.00	**381.31**	500.00	757.10	**757.10**
Shock UD	500.00	556.37	**356.13**	500.00	694.45	**402.46**
Shock UU	500.00	611.70	**565.97**	500.00	747.91	**716.29**
Shock DD	500.00	500.00	**516.57**	500.00	516.56	**258.40**
Drunkards Walk	500.00	500.00	**425.00**	500.00	644.01	**642.33**
		Mean	453.67		**Mean**	776.17
		σ	70.46		σ	31.95

ZIPMM performs better in a market where the price is constantly fluctuating between highs and lows in a regular fashion; no matter the overall price trend. This is evident in Table 1. - ZIPMM accumulates a higher net worth in a bear market than in a ranging one. Furthermore, the difference between the bull and bear results aren't as large as one might expect. These results make sense in the context of the trading indicators I used. Bollinger Bands will perform well in this kind of market due to the regular movement of the market price to and from the equilibrium. This movement makes it easier to spot peaks using the standard deviation, and a reversion to the equilibrium allows for profit to be made from this. Stochastic oscillators tend to perform well in range-bound markets (as opposed to RSI which tends to perform well in trending markets), however; in this case the trending markets showed better profits. A possible explanation for this is the inclusion of the SMA indicators. Stochastic oscillators are often well paired with trend indicators. Again, the regular oscillations about the equilibrium, create good conditions for the stochastic oscillator to identify peaks and troughs in the market. The larger the amplitude of the price oscillations, the more profit ZIPMM is able to make, this explains the increasing gradient towards the end of the bear market session in Fig. 5.

ZIPMM appears to generate more profit with cyclic trends, supported by the comparatively low profit it was able to generate in a bullish drunkards walk

market vs a standard bull market. Most trading indicators are correlated to the cyclical movements of a market, so this makes sense. It may be the most telling indicator of how ZIPMM would perform in a real market, despite it still being a huge simplification. The additional randomness should misalign the technical indicators, causing the signals they produce to be out of phase, something which would be a regular occurrence in a real complex market.

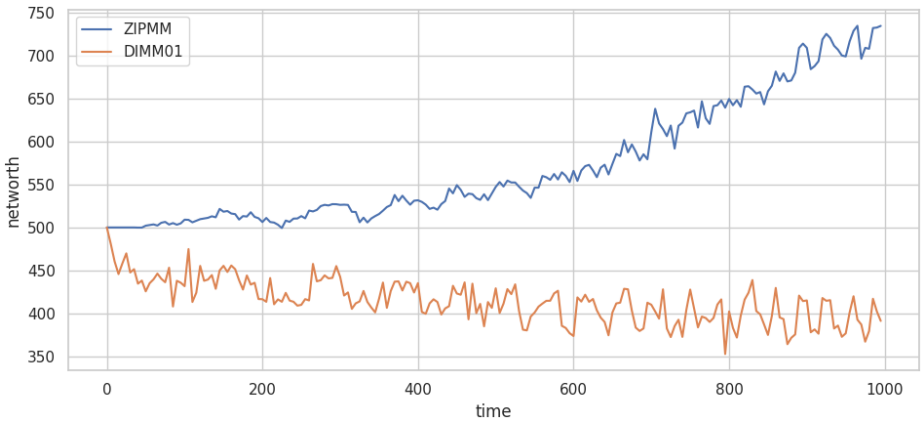

Fig. 5. The average net worth taken every 5 time steps (for a total of 1000) over 10 market sessions for ZIPMM and DIMM01 under bear market price conditions.

ZIPMM performs badly when shock changes are introduced to the equilibrium. This is due to the fact that ZIPMM cannot react quickly enough to a sudden drop in market price. Often it will be holding a number of stock when the market drops, resulting in an amplified decrease in net worth. There is also no stop-loss functionality for ZIPMM, which explains why it performs much better in steadier markets. Once ZIPMM is holding stock worth significantly less than the price it purchased it for, there is no mechanism for it to sell. This is a floor which would need addressing if ZIPMM is to perform well in shock markets. Occasionally ZIPMM will get lucky and make a large amount of profit in a shock market, this is through no skill of its own however.

2.4 Altering Trader Agent Population

The experiments shown in Table 2. were conducted with a bullish market trend, but with different proportions of each sales trader in the market.

As can be seen from the results, ZIPMM appears to perform relatively well across the board. Even when ZIP agents are the only traders present in the market, ZIPMM still manages to accumulate a reasonable net worth by the end of session.

Table 2. The average net worth of ZIPMM compared to DIMM01 for different trader agent populations. These results were taken over ten market sessions each lasting 1000 time steps. Buy and sell populations are symmetric. The ratio is of the form GVWY:SHVR:ZIC:ZIP

	DIMM01			**ZIPMM**		
Population	Start	Peak	End	Start	Peak	End
10:10:10:10	500.00	534.99	**499.54**	500.00	843.27	**829.06**
0:10:10:10	500.00	500.00	**403.34**	500.00	760.45	**760.45**
10:0:10:10	500.00	517.82	**473.14**	500.00	842.56	**821.82**
10:10:0:10	500.00	538.68	**518.39**	500.00	764.99	**739.12**
10:10:10:0	500.00	500.00	**398.00**	500.00	787.76	**776.63**
0:0:0:40	500.00	537.33	**523.43**	500.00	748.08	**729.92**
		Mean	469.31		**Mean**	776.17
		σ	51.1		σ	31.95

3 Conclusion and Improvements

It would be useful to make further comparisons of ZIPMM to other market makers operating under different algorithms, and this would make for some interesting future work. For now, comparing with DIMM01 - in the first set of experiments ZIPMM's mean final net worth is 1.3 times larger than DIMM01's (1.6 times not including shock experiments) and 1.7 times larger in the second. DIMM01 also appears to maintain a more volatile net worth, which can be seen by its relatively high standard deviation compared to that of ZIPMM. This can be explained by the fact that DIMM01's net worth tends to fluctuate heavily in tune with the market price, perhaps another reason ZIPMM is able to outperform DIMM01. The superiority of ZIPMM over DIMM01 is evident from my results in every single market experiment with the exception of the double-drop shock market. As was pointed out in a previous section, this highlights a major floor of ZIPMM. Firstly, the inability to react quickly to very sharp market changes and secondly, the absence of a suitable stop-loss functionality.

Quicker reaction speeds could be obtained by allowing ZIPMM to be in a state of buy and sell at the same time when it is holding multiple stock. A stop-loss could be implemented by waiting for a suitably large market drop and then selling at the equilibrium price, although this is difficult to implement without ruining profits in other market types.

References

1. D. Cliff: Minimal-Intelligence Agents for Bargaining Behaviors in Market-Based Environments. Technical Report, HP Labs. (1997)
2. BSE: A LOB-Based Financial Exchange Simulator, Github open-source repository (code & documentation), https://bit.ly/2XdW184.

```
class Trader_ZIPMM(Trader):

    # Given that BSE is in one big file, this class is designed to
        ↪ be pasted
    # into the BSE.py file rather than to remain in a standalone
        ↪ file. It will
    # not compile unless in the BSE.py file.
    def __init__(self, ttype, tid, balance, time):
        Trader.__init__(self, ttype, tid, balance, time)
        self.job = 'Buy'
        self.willing = 1
        self.able = 1
        self.active = False # gets switched to True while actively
            ↪ working an order
        self.prev_change = 0 # this was called last_d in Cliff'97
        self.beta = 0.1 + 0.4 * random.random()
        self.momntm = 0.1 * random.random()
        self.ca = 0.05 # self.ca & .cr were hard-coded in '97 but
            ↪ parameterised later
        self.cr = 0.05
        self.margin = None # this was called profit in Cliff'97
        self.margin_buy = -1.0 * (0.05 + 0.3 * random.random())
        self.margin_sell = 0.05 + 0.3 * random.random()
        self.price = None
        self.limit = None
        # memory of best price & quantity of best bid and ask, on
            ↪ LOB on previous update
        self.prev_best_bid_p = None
        self.prev_best_bid_q = None
        self.prev_best_ask_p = None
        self.prev_best_ask_q = None
        self.recent_trades = deque()
        self.long_term_window = 14
        self.short_term_window = 6
        self.long_term_moving_avg = None
        self.short_term_moving_avg = None
        self.boll_mid = None
        self.boll_upper = None
        self.boll_lower = None
        self.sma_long_window = 3
        self.sma_short_window = 1

        self.oscillations = deque()
        self.k = None
        self.d = None
```

```
        self.highest_price = None
        self.lowest_price = None

        self.buy_times = []
        self.sell_times = []
        self.inventory = []
        self.max_inventory = 3
        self.ema_param = 2 / float(self.long_term_window + 1)
        self.eqlbm = None
        self.stock_qty = 0
        self.inventory = []

    def getorder(self, time, countdown, lob):
        '''Equivalent to GVWY's getoder method'''

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


    def get_moving_avg(self, q, start):
        '''Returns the simple moving average from the latest trade
            ↪  back to the
        start index'''
        total = 0
        for i in range (start, len(q)):
            total += q[i]
        return total / (len(q)-start)

    def update_boll_bands(self):
        '''Updates the bollinger bands based on the recent trades
            ↪ '''
        self.boll_mid = self.get_moving_avg([item['price'] for
            ↪ item in self.recent_trades], 0)

        standard_dev = 0
        for item in self.recent_trades:
```

```
            standard_dev += (item['price'] - self.boll_mid)**2
        standard_dev = math.sqrt(standard_dev / self.
            ↪ long_term_window)

        self.boll_lower = self.boll_mid - (1 * standard_dev)
        self.boll_upper = self.boll_mid + (1 * standard_dev)

    def updateEq(self, price):
        '''Updates the equilibrium price estimate using EMA'''
        if self.eqlbm == None: self.eqlbm = price
        else: self.eqlbm = self.ema_param * price + (1 - self.
            ↪ ema_param) * self.eqlbm

    # update margin on basis of what happened in market
    def respond(self, time, lob, trade, verbose):
        if (trade != None):
            self.recent_trades.append(trade)
            if (len(self.recent_trades) > self.long_term_window):
                self.recent_trades.popleft()

                # Calculate K for stochastic oscillator
                self.highest_price = max([item['price'] for item in
                    ↪  self.recent_trades])
                self.lowest_price = min([item['price'] for item in
                    ↪ self.recent_trades])
                self.k = (trade['price'] - self.lowest_price) * 100
                    ↪  / (self.highest_price - self.lowest_price)
                self.oscillations.append(self.k)

                self.long_term_moving_avg = self.get_moving_avg([
                    ↪ item['price'] for item in self.recent_trades
                    ↪ ], self.long_term_window - self.
                    ↪ sma_long_window)
                self.short_term_moving_avg = self.get_moving_avg([
                    ↪ item['price'] for item in self.recent_trades
                    ↪ ], self.long_term_window - self.
                    ↪ sma_short_window)

                # if (self.short_term_moving_avg > self.
                    ↪ long_term_moving_avg):
                    # print(str(self.long_term_moving_avg) + " " +
                        ↪ str(self.short_term_moving_avg))
                # Calculate D for stochastic oscillator
                self.update_boll_bands()
```

```
        if (len(self.oscillations) > self.short_term_window
            ↪ ):
            self.oscillations.popleft()
            self.d = self.get_moving_avg(self.oscillations,
                ↪ 0)

        self.updateEq(trade['price'])

# ZIP trader responds to market events, altering its
    ↪ margin
# does this whether it currently has an order to work or
    ↪ not
def target_up(price):
    # generate a higher target price by randomly perturbing
        ↪  given price
    ptrb_abs = self.ca * random.random() # absolute shift
    ptrb_rel = price * (1.0 + (self.cr * random.random()))
        ↪ # relative shift
    target = int(round(ptrb_rel + ptrb_abs, 0))
    # # print('TargetUp: %d %d\n' % (price,target))
    return target

def target_down(price):
    # generate a lower target price by randomly perturbing
        ↪ given price
    ptrb_abs = self.ca * random.random() # absolute shift
    ptrb_rel = price * (1.0 - (self.cr * random.random()))
        ↪ # relative shift
    target = int(round(ptrb_rel - ptrb_abs, 0))
    # # print('TargetDn: %d %d\n' % (price,target))
    return target

def willing_to_trade(price):
    # am I willing to trade at this price?
    willing = False
    if self.job == 'Buy' and self.active and self.price >=
        ↪ price:
        willing = True
    if self.job == 'Sell' and self.active and self.price <=
        ↪  price:
        willing = True
    return willing

def profit_alter(price):
    tmp_price = 0
```

```
        # set the price from limit and profit-margin
        if (self.job == 'Buy'):
            if (trade == None):
                tmp_price = lob['bids']['best']
            elif (self.eqlbm == None):
                tmp_price = trade['price']
            else:
                tmp_price = self.eqlbm
        else:
            tmp_price = self.last_purchase_price

        oldprice = self.price
        diff = price - oldprice
        change = ((1.0 - self.momntm) * (self.beta * diff)) + (
            ↪ self.momntm * self.prev_change)
        self.prev_change = change
        newmargin = ((self.price + change) / tmp_price) - 1.0

        if self.job == 'Buy':
            if newmargin < 0.0:
                self.margin_buy = newmargin
                self.margin = newmargin
        else:
            if newmargin > 0.0:
                self.margin_sell = newmargin
                self.margin = newmargin


        self.price = int(round(tmp_price * (1.0 + self.margin),
            ↪  0))

    # # print('old=%d diff=%d change=%d price = %d\n' % (
        ↪ oldprice, diff, change, self.price))
    if (self.price == None):
        if (self.job == 'Buy'):
            if (trade == None):
                self.price = lob['bids']['best']
            elif (self.eqlbm == None):
                self.price = trade['price']
            else:
                self.price = self.eqlbm
        else:
            self.price = self.last_purchase_price

    # what, if anything, has happened on the bid LOB?
```

```
        bid_improved = False
        bid_hit = False
        lob_best_bid_p = lob['bids']['best']
        lob_best_bid_q = None
        if lob_best_bid_p is not None:
            # non-empty bid LOB
            lob_best_bid_q = lob['bids']['lob'][-1][1]
            if (self.prev_best_bid_p is not None) and (self.
                ↪ prev_best_bid_p < lob_best_bid_p):
                # best bid has improved
                # NB doesn't check if the improvement was by self
                bid_improved = True
            elif trade is not None and ((self.prev_best_bid_p >
                ↪ lob_best_bid_p) or (
                    (self.prev_best_bid_p == lob_best_bid_p) and (
                        ↪ self.prev_best_bid_q > lob_best_bid_q))):
                # previous best bid was hit
                bid_hit = True
        elif self.prev_best_bid_p is not None:
            # the bid LOB has been emptied: was it cancelled or hit
                ↪ ?
            last_tape_item = lob['tape'][-1]
            if last_tape_item['type'] == 'Cancel':
                bid_hit = False
            else:
                bid_hit = True

        # what, if anything, has happened on the ask LOB?
        ask_improved = False
        ask_lifted = False
        lob_best_ask_p = lob['asks']['best']
        lob_best_ask_q = None
        if lob_best_ask_p is not None:
            # non-empty ask LOB
            lob_best_ask_q = lob['asks']['lob'][0][1]
            if (self.prev_best_ask_p is not None) and (self.
                ↪ prev_best_ask_p > lob_best_ask_p):
                # best ask has improved -- NB doesn't check if the
                    ↪ improvement was by self
                ask_improved = True
            elif trade is not None and ((self.prev_best_ask_p <
                ↪ lob_best_ask_p) or (
                    (self.prev_best_ask_p == lob_best_ask_p) and (
                        ↪ self.prev_best_ask_q > lob_best_ask_q))):
```

```
        # trade happened and best ask price has got worse,
            ↪ or stayed same but quantity reduced
        # -- assume previous best ask was lifted
        ask_lifted = True
elif self.prev_best_ask_p is not None:
    # the ask LOB is empty now but was not previously:
        ↪ canceled or lifted?
    last_tape_item = lob['tape'][-1]
    if last_tape_item['type'] == 'Cancel':
        ask_lifted = False
    else:
        ask_lifted = True

if verbose and (bid_improved or bid_hit or ask_improved or
    ↪  ask_lifted):
    print('B_improved', bid_improved, 'B_hit', bid_hit, '
        ↪ A_improved', ask_improved, 'A_lifted',
        ↪ ask_lifted)

deal = bid_hit or ask_lifted

if self.job == 'Sell':
    # seller
    for item in self.inventory:
        self.last_purchase_price = item
        if deal:
            tradeprice = trade['price']
            if self.price <= tradeprice:
                # could sell for more? raise margin
                target_price = target_up(tradeprice)
                profit_alter(target_price)
            elif ask_lifted and self.active and not
                ↪ willing_to_trade(tradeprice):
                # wouldnt have got this deal, still working
                    ↪ order, so reduce margin
                target_price = target_down(tradeprice)
                profit_alter(target_price)
        else:
            # no deal: aim for a target price higher than
                ↪ best bid
            if ask_improved and self.price > lob_best_ask_p:
                if lob_best_bid_p is not None:
                    target_price = target_up(lob_best_bid_p)
                else:
```

```
                    target_price = lob['asks']['worst'] #
                        ↪ stub quote
                profit_alter(target_price)

        # Main sell logic, taking limit price to be the
            ↪ purchase price
        if (lob['bids']['n'] > 0):
            bestbid = lob['bids']['best']
            self.margin = self.margin_sell
            askprice = self.last_purchase_price * (1 + self.
                ↪ margin)
            # askprice = self.eqlbm * (1 + self.margin)
            if askprice < bestbid:
                order=Order(self.tid, 'Ask', askprice, 1,
                    ↪ time, lob['QID'])
                self.orders=[order]
                break

if self.job == 'Buy':
    if self.margin == None:
        self.margin = self.margin_buy
    # buyer
    if deal:
        tradeprice = trade['price']
        if self.price >= tradeprice:
            # could buy for less? raise margin (i.e. cut the
                ↪  price)
            target_price = target_down(tradeprice)
            profit_alter(target_price)
        elif bid_hit and self.active and not
            ↪ willing_to_trade(tradeprice):
            # wouldnt have got this deal, still working
                ↪ order, so reduce margin
            target_price = target_up(tradeprice)
            profit_alter(target_price)
    else:
        # no deal: aim for target price lower than best ask
        if bid_improved and self.price < lob_best_bid_p:
            if lob_best_ask_p is not None:
                target_price = target_down(lob_best_ask_p)
            else:
                target_price = lob['bids']['worst'] # stub
                    ↪ quote
            profit_alter(target_price)
```

```
            # Main buy logic, includes use of stochastic
                ↪ oscillators and SMA
            # crossover, limit price is determined by EMA
            if (self.d != None and self.d < 20 and self.k > self.d
                ↪ and self.short_term_moving_avg > self.
                ↪ long_term_moving_avg):
                self.margin = self.margin_buy
                bidprice = self.eqlbm * (1 + self.margin)
                if bidprice < self.balance:
                    order=Order(self.tid, 'Bid', bidprice, 1, time,
                        ↪ lob['QID'])
                    self.orders=[order]

        # remember the best LOB data ready for next response
        self.prev_best_bid_p = lob_best_bid_p
        self.prev_best_bid_q = lob_best_bid_q
        self.prev_best_ask_p = lob_best_ask_p
        self.prev_best_ask_q = lob_best_ask_q

    def bookkeep(self, trade, order, verbose, time):
        outstr = ""
        for order in self.orders:
            outstr = outstr + str(order)

        self.blotter.append(trade) # add trade record to trader's
            ↪ blotter
        # NB What follows is **LAZY** -- it assumes all orders are
            ↪  quantity=1
        transactionprice = trade['price']

        bidTrans = True #did I buy? (for output logging only)
        if self.orders[0].otype == 'Bid':
            # Bid order succeeded, remember the price and adjust
                ↪ the balance
            self.stock_qty += 1
            self.buy_times.append(trade['time'])
            self.balance -= transactionprice
            self.last_purchase_price = transactionprice
            self.job = 'Sell' # now try to sell it for a profit
            self.inventory.append(transactionprice)
        elif self.orders[0].otype == 'Ask':
            bidTrans = False # we made a sale (for output logging
                ↪ only)
            # Sold! put the money in the bank
            self.stock_qty -= 1
```

```
            self.sell_times.append(trade['time'])
            self.balance += transactionprice
            # self.last_purchase_price = 0
            self.job = 'Buy' # now go back and buy another one
            self.inventory.remove(self.last_purchase_price)
        else:
            sys.exit('FATAL: ZIPMM doesn\'t know .otype %s\n' %
                ↪ self.orders[0].otype)

        if (len(self.inventory) == self.max_inventory):
            self.job = 'Sell'
        elif(len(self.inventory) == 0):
            self.job = 'Buy'
        else:
            # Use Bollinger bands to decide if we should buy more
                ↪ stock or sell
            # our currently owned stock
            if (trade['price'] <= (self.boll_lower)):
                self.job = 'Buy'
            else:
                self.job = 'Sell'

        self.n_trades += 1

        verbose = True # We will log to output

        if verbose: # The following is for logging output to
            ↪ terminal
            if bidTrans: # We bought some shares
                outcome = "Bght"
                owned = 1
            else: # We sold some shares
                outcome = "Sold"
                owned = 0
            net_worth = self.balance + self.last_purchase_price
            print('%s, %s=%d; Qty=%d; Balance=%d, NetWorth=%d' %
            (outstr, outcome, transactionprice, owned, self.balance
                ↪ , net_worth))

        self.del_order(order) # delete the order
```