FNCE30010

Algorithmic Trading

Semester 2, 2020

Project 1 - Task 1 (Induced demand-supply)

Due **11:59 pm on Monday, 7 September 2020**

Nitin Yadav and Peter Bossaerts

Brain, Mind & Markets Lab.

Department of Finance.

Faculty of Business and Economics.

The University of Melbourne.

Administrative Arrangements

Due date:	**11:59 pm on Monday, 7 September 2020**
Late assignments:	Late submission assignments may attract a penalty unless an extension has been granted. All extension requests must be made prior to the assessment due date and supported with appropriate documentation. Unless an extension has been granted, penalties to the assessment will be applied. For assignments submitted after the due date, the mark awarded will be reduced by 10% for each day the work is late. For assignments submitted later than 5 working days after the due date will not be marked and will receive no marks.
Where to submit:	Assignment submission is via the LMS Assignment Submission link for all written assignments. Please refer to the LMS Student Guide: Turnitin Assignments for detailed submission instructions, if needed.
Group size:	This is an **individual** assignment.
Submission Metadata:	You should include your details as part of your submission. **Subject Code and Name** **Student Number** **Full Name** **Assignment Name**
Word limit:	FBE usually as a word limit. This assignment is for computer code, however, and hence word limits do not apply.
Marks:	This assignment counts 15% towards the final mark in this subject.

The assignment must be your own work. Students are encouraged to discuss the assignment and to share information sources. However, the writing of each group's assignment must be conducted separately and independently.

Assignment Instructions

Trading Setting (Identical to “Instructions” for Induced Demand/Supply Manual Trading)

You will be asked to trade in a marketplace to trade Widgets.

When you log in, you will see that you have two markets, one **public** market and one **private** market. You will be given a certain amount of equal securities in both markets and some cash. A trading session will last for 20 minutes.

All your offers in the public market are displayed in a common book (and visible as red and blue entries in the book). The common book in the public market is visible to all traders.

You are not allowed to initiate offers in the private market, instead you can only respond to existing orders in the private market. When the marketplace opens you will get an order in the private market (either a Buy order or a Sell order). This order is not visible to other traders. *The private orders can be refreshed multiple times during a session.*

Here is how your profit will be measured.

If you get a Buy order for $X in the private market, then you make money by buying a widget for less than $X in the public market and selling that widget in the private market.

If you get a Sell order for $X in the private market, then you make money by selling a widget for more than $X in the public market and buying back that widget in the private market.

In addition, you are allowed to “speculate” which is to buy and subsequently sell; you keep the earnings if the purchase price is below the sales price, otherwise the loss will be subtracted from your earnings. Conversely, you may be able to sell high and buy back at a lower price. The corresponding gains are added to your earnings.

Your profit is calculated based on the following formula:

$P = C_T - C_0 + P_M(min(0, \Delta H))$ where:

- C_T is the cash at the end of the session,
- C_0 is the initial cash,
- P_M is the maximum price in the public market, and
- ΔH is the difference of the widget holdings between the end of session and the start of session. It is calculated as $\Delta H = (h_r + h_u) - (h^*_r + h^*_u)$ where,
    - h_r = final quantity of widgets in the private market
    - h_u = final quantity of widgets in the public market
    - h^*_r = initial quantity of widgets in the private market
    - h^*_u = initial quantity of widgets in the public market

For example, if you start with a total of 20 widgets (10 in the public market and 10 in the market market) and:

- hold 15 widgets when the session closes, ΔH = 15 - 20 = -5, $min(0,\ \Delta H) = -5$ and $P = C_T\ -\ C_0 - 5\,P_M$ .That is you will be penalised by $5P_M$.
- hold 25 widgets when the session closes, ΔH = 25 - 20 = 5, $min(0,\ \Delta H) = 0$ and $P = C_T\ -\ C_0$
- hold 20 widgets when the session closes, ΔH = 20 - 20 = 0, $min(0,\ \Delta H) = 0$ and $P = C_T\ -\ C_0$

You should aim at ΔH = 0. Note that If ΔH is positive (i.e., $\Delta H >$ 0) then you would have bought more widgets than required, and hence you will end up with less cash and therefore lower performance.

Task Specification

For this task, please use the robot class template DSBot provided on the LMS. If you add new functions in the given file, please make sure you prefix your functions with an underscore ("_"). You should not remove any function from the provided template.

1. Trading and bot parameters .... [1 mark]
    a. Profit margin. Create a global variable called `PROFIT_MARGIN` that stores the minimum amount of required profit (in cents) per trade. For example, if there is a private order to buy at $5 and the profit margin is 10 then the bot should only buy widgets in the public market for at most $4.90.
    b. Trading account details. Change the variables that start with `FM_` prefix (e.g., `FM_ACCOUNT`) to your Flexemarkets details.
    c. Your details. Change the `SUBMISSION` variable to your student details. This variable is a dictionary (e.g., `SUBMISSION = {"number": "123", "name": "Name 1"}`).
2. Role of the bot .... [2 marks]
   The template defines an enumeration called `Role` that has two possible values, `BUYER` and `SELLER`. You should infer the role from orders that you get in the private market (see task description), and accordingly update the `_role` variable in the DSBot class.
3. Type of the bot .... [2 marks]
   The template defines an enumeration called BotType with two types. The bot can either react to orders in the order book (type `REACTIVE`) or can proactively send orders to the market (`MARKET_MAKER`). Add an extra argument called `bot_type` to the constructor of the class DSBot and store its value in a class variable called `_bot_type`. In a session, the bot can only either be of type market maker or reactive (but not both). You should provide correct implementation for both of these types.

4. Identification of profitable trade opportunities .... [3 marks]
   The bot should identify ALL profitable trade opportunities (irrespective of weather it responds to those opportunities). Trade opportunities should be identified based on the best bid and best ask prices. The bot should print the details of the profit opportunity and if it is responding to the trade opportunity by sending an order. Additionally, a reason should be printed if the bot is unable to respond (e.g., a buyer bot has ran out of cash). To print the trade opportunity please use the provided `_print_trade_opportunity` function.
5. Order management .... [5 marks]
   a. Market Id. The DSBot class has two variable named `_public_market_id` and `_private_market_id` to store the id of the public and private market, respectively. When the bot is initialised, you should correctly store the market ids. You should use these ids for order management (e.g., sending a new order).
   b. Maximum number of open orders
      At any point in time the bot should have only 1 pending order in the public order book. Each order should be for only 1 unit of widget. Note: Implementing this will require waiting for the server to accept/reject sent orders. Please refer to the bot built during the second tutorial for help.
   c. Order validity
      Before sending an order the bot should check for order validity (e.g., does it have enough cash to buy). The bot should not send an order that will decrease its profitability.
6. Code quality .... [2 marks]
   Your submitted code must be clear to understand and well commented. Please read the following links for suggestions:
   – https://docs.python-guide.org/writing/style/
   – https://www.python.org/dev/peps/pep-0008/