FIN 566
Problem Set #6

Adam D. Clark-Joseph

Due 11/6/2017

University of Illinois Urbana-Champaign

This problem set introduces a matching engine with full entry, cancellation, and modification capabilities. In the first part of the problem set, I ask you to construct some market-making algorithms that make use of these new functionalities. In the second part of the problem set, I introduce a template that allows multiple non-background-trader algorithms to interact, and I ask you to explore some simple strategic competition.

Part I

For this part (Part I), use "Enter_cancel_mod_matching_subscript.m",
and "FIN566_PS6a_2017_main_script_template".

Unlike in PS#5, whether or not robot_1 gets the chance to participate in a given period is being by the simulation template now. Also, robot_1 deterministically gets the chance to participate once every num_bgt periods, instead of just participating that often on average.

Unless otherwise stated, instructions about what robot_1 should do refer to what he should do *in periods when he gets the chance to participate.* Also in this part (Part I), unless otherwise stated, assume that robot_1 only enters orders of size 1.

1 Warm-up to Working with the New Matching Engine

(a) Write an algorithm for robot_1 so that he places passive orders of size 1 at the best on a randomly chosen side of the book. This is exactly like the market-making algorithms from PS#3, except that now the code must be a little different to work with the new matching engine.

Hint: remember to include the"message_type" variable.

(b) Run the simulation and report robot_1's total trading volume, total number of orders placed, and number of orders resting in the orderbook at the end of the simulation.

2 Cancellations and Inventory Control

(a) Write an algorithm for robot_1 so that:

- If his net inventory position is positive, he cancels all of his resting buy orders, then enters a passive sell order at the best ask

- If his net inventory position is negative, he cancels all of his resting sell orders, then enters a passive buy order at the best bid
- If his net inventory position is zero, he randomly chooses a side of the book, and enters a passive order there at the best.

Please turn in your code and a plot of robot_1's net inventory position (in shares)

(b) Write an algorithm for robot_1 so that he always enters a passive order at the best on one side of the book or the other, but his net inventory position never goes above +3 shares, or below -3 shares.

Please turn in your code and a plot of robot_1's net inventory position (in shares).

3 Upward Quantity Modifications

(a) Change the algorithm you wrote for question 1(a) so that if robot_1 already has an order resting at the best on the side of the book he randomly chose, he modifies that existing resting order by increasing its quantity by one share. Please turn in your code. *Hint: you need to do a downward modification followed by a new entry.*

(b) Observe that permanent price-impact is linear in aggressive-order size. Explain why this means that the algorithm in part (a) will tend to earn lower profits per share traded than will the original version from question 1.

4 Conditioning MM Strategy on Resting Depth

(a) Change the algorithm you wrote for question 1(a) so that if resting depth at the best on either side of the book is greater than 23, robot_1 does not enter an order. Please turn in your code.

(b) Run the simulation and report robot_1's total trading volume, total number of orders placed, and number of orders resting in the orderbook at the end of the simulation.

Part II

For this part (Part II), use "Enter_cancel_mod_matching_subscript_battlebots.m", "FIN566_PS6b_main_script_template.m", and "FIN566_PS6b_meta_script_template.m"

5 Warm Up

(This question should take you about 4 minutes. The purpose is just to make sure you understand how the different algorithms get called and their activity stored.)

- The script "robot1_algo_mm_PS6b_template_2017.m" is the simplest market-making algorithm that we have seen; it just places a passive order at the best price on a randomly chosen side of the book every time it gets a chance to do something.
- The script "robot2_algo_aggressor_PS6b_template_2017.m" is the first algorithm we have seen that resembles a proprietary aggressive trading algorithm (as opposed to an agency/execution algorithm). This algorithm uses information that was previously forbidden to us, namely the "last_order_price." However, this algorithm uses that information in about the simplest possible way.

(a) Run the meta script with two different "robot1_algo_mm_PS6b_template_2017" robots (but no other algos except background traders) participating at the same time. Report the output table that the meta script produces.

(b) Run the meta script with two different "robot2_algo_aggressor_PS6b_template_2017" robots (but no other algos except background traders)participating at the same time. Report the output table that the meta script produces.

(c) Run the meta script with three different "robot1_algo_mm_PS6b_template_2017" robots and three different "robot2_algo_aggressor_PS6b_template_2017" robots all participating at the same time. Report the output table that the meta script produces. Indicate which robots in the output table are the aggressor algorithms.

6 Better Smart Robots

(a) Change the "robot1_algo_mm_PS6b_template_2017" algorithm so that it includes inventory control, and uses information about past aggressive orders in some way that increases its profits. DO NOT use information about last_order_price, but otherwise try to make the algorithm about as profitable as you can. (Try hard, but you don't literally need to figure out a provably optimal strategy.)

Please briefly explain what you changed (you do not need to turn in your code). Run the meta script with your new algorithm as the only smart robot, and report the output table that the meta script produces.

(b) Change the "robot2_algo_aggressor_PS6b_template_2017" algorithm so that it includes inventory control, and uses information about last_order_price in some way that is more profitable than the original way. You don't need to make the algorithm perfectly optimal, but try hard to make it very profitable.

Please briefly explain what you changed (you do not need to turn in your code). Run the meta script with your new algorithm as the only smart robot, and report the output table that the meta script produces.

7 Battlebots (A.K.A., Cyborg Kyle)

*****Nothing to turn in, but please do the question!*****

(a) Run the meta script with both of the algorithms you wrote in Question 2 participating. Look at the output table that the meta script produces; keep track of which robot in the output table is the aggressor algorithm.

(b) Change the market-making algorithm in some way (still DO NOT use information about last_order_price) to better compete against the aggressor algorithm. Please think about how to briefly explain what you changed, and repeat part (a) using your new market-making algorithm and the same aggressor algorithm.

(c) Suppose that maximum order size for the aggressor algorithm is 32. Change the aggressor algorithm to obey this new constraint, then change the algorithm to better compete with the market-making algorithm from part (b) of this question. Please think about how to briefly explain what you changed, and repeat part (a) using your your two new algorithms.

8 Don't Build Skynet ***Extremely Important!***

(a) Look up this reference (Google "don't build skynet," and look at http://xkcd.com/534/).

(b) Don't build Skynet.