Sreesanth Adelli & Troy Anderson
April 4th, 2025
Financial Trading Strat. And Risk MGT.
Professor Ryan Davies

Market Making Algorithm for ALGO2: Development and Analysis

I have abided by the Babson Code of Ethics in this work and pledge to be better than that which would compromise my integrity.

Introduction

This report presents the development and optimization of a market making algorithm for the ALGO2 case. Market making involves providing liquidity by simultaneously posting buy and sell orders with the goal of earning the bid-ask spread and collecting rebates. Our approach builds upon the provided base algorithm with strategic modifications that enhance profitability while effectively managing risk. Through systematic parameter testing and optimization, we have developed an algorithm that consistently generates profits while remaining resilient to various market conditions. We do this by using a parameter testing file, experiment.py, that tests various parameter values and saves them to a csv. We then plot the csv using a plot_results.py file. The results from the graphs are then used to create our refined strategy - final.py.

Algorithm Strategy and Implementation

Our market making algorithm aims to generate consistent profits by capturing bid-ask spreads while effectively managing inventory risk. The core strategy places limit orders at the best bid and ask prices:

```
# get the best bid and ask price of the security

best_bid, best_ask = ticker_bid_ask(s, TICKER)
```

```
# post a buy and sell limit order at bid and ask prices

s.post(f'{API_BASE_URL}/orders', params={'ticker': TICKER, 'type': 'LIMIT',
'quantity': buy_quantity, 'action': 'BUY', 'price': best_bid})

s.post(f'{API_BASE_URL}/orders', params={'ticker': TICKER, 'type': 'LIMIT',
'quantity': sell_quantity, 'action': 'SELL', 'price': best_ask})
```

The algorithm manages inventory risk by adjusting order sizes when position thresholds are exceeded:

```
# adjust order size if current position exceeds value of rebalance_limit

if position > rebalance_limit:

    buy_quantity = rebalancesize

elif position < -rebalance_limit:

    sell_quantity = rebalancesize
```

We implement a queue-based cancellation strategy to ensure orders remain relevant:

```
# delete oldest order when number of orders exceeds orderslimit

while num_orders > orderslimit:

    orderid = orders[num_orders-1]['order_id']

    s.delete(f'{API_BASE_URL}/orders/{orderid}')
```

This approach helps the algorithm perform effectively in both trending markets (by reducing exposure in the trending direction) and range-bound markets (by consistently capturing the full spread).

Parameter Optimization and Empirical Results

We systematically evaluated and optimized several key parameters by varying one at a time and analyzing the performance impact using Net Liquidation Value (NLV) as our primary metric. Each subsection below discusses the rationale, testing methodology, results (supported by graphs), and final selected value for that parameter.

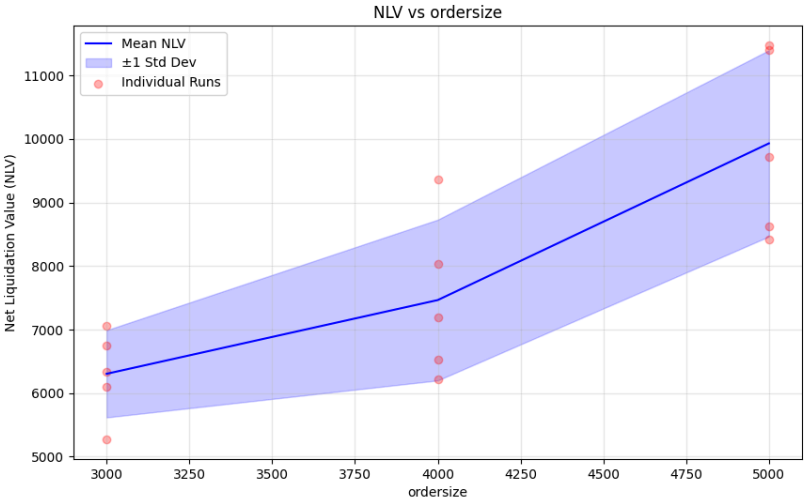
---

1. Order Size (`ordersize`)

The `ordersize` parameter controls the number of shares posted in each bid and ask order. We tested values of **3000**, **4000**, and **5000** shares. We found that the higher the volume the better, most likely because it simply allowed us to scale our strategy more. We got close to the trading limits, but did not hit them or receive any fines yet.

Final Value: `ordersize = 5000`

Graph:



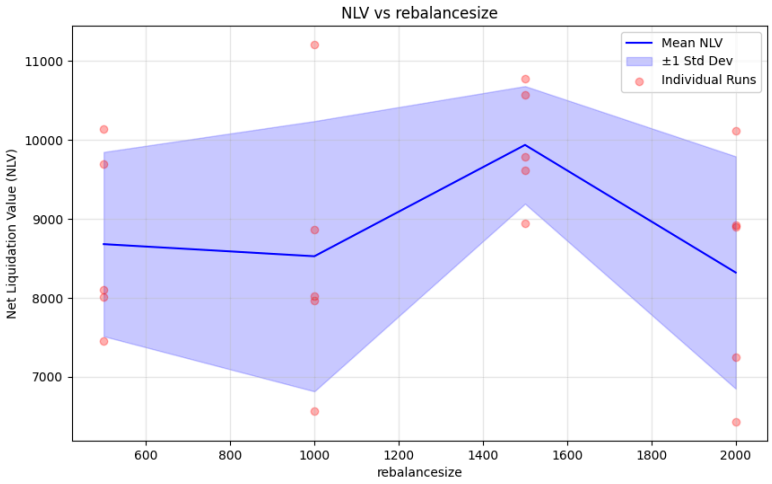
---

2. Rebalance Size (`rebalancesize`)

`rebalancesize` determines how aggressively the algorithm corrects its position when inventory limits are exceeded. We tested values of **500**, **1000**, **1500**, and **2000** shares. We found that the sweet spot was **1500** shares. Interestingly, we started to hit trading limits and receive fines at 1500, but the fines were low enough that it was worth it. The fines were too high for 2000.

Final Value: `rebalancesize = 1500`

Graph:



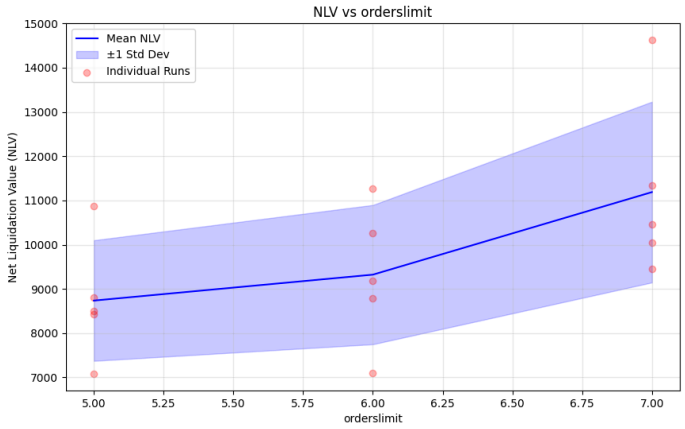
---

3. Order Limit (`orderslimit`)

This parameter limits the number of outstanding limit orders at any time. Tested values were **5**, **6**, and **7**. We found that 7 gave us higher returns than the rest. We continue to receive fines, raising the order limit does not help in this regard. Reason tells us that it should increase fines, but empirically, there was no difference.

Final Value: `orderslimit = 7`

Graph:



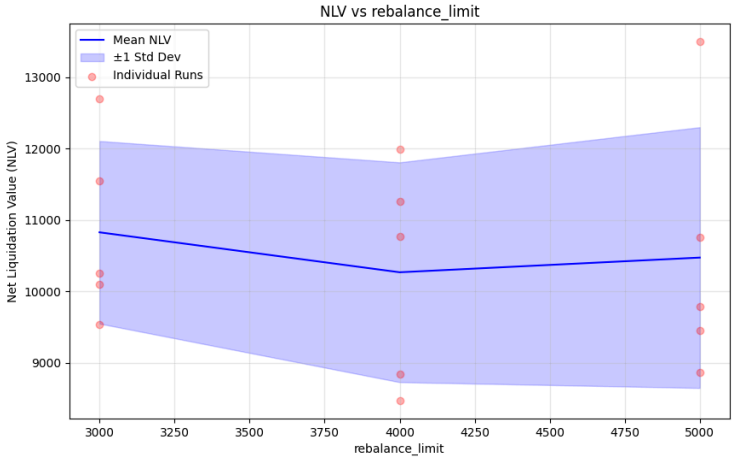
---

4. Rebalance Threshold (`rebalance_limit`)

This parameter triggers inventory rebalancing when positions become too large. We tested values of **3000**, **4000**, and **5000** shares. We found that there was no significant difference between these, and so decided to keep it the same (4000).

Final Value: `rebalance_limit = 4000`

Graph:



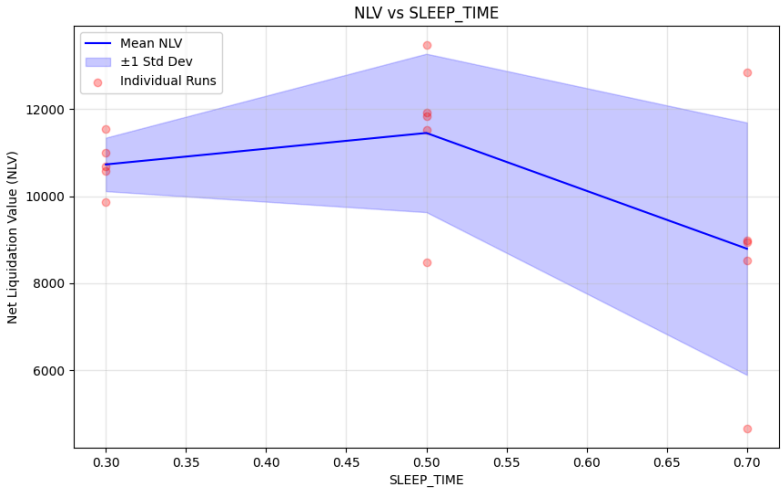
---

5. Sleep Time (`SLEEP_TIME`)

`SLEEP_TIME` controls how frequently the algorithm checks market conditions and posts/cancels orders. More specifically, it's the amount of time it waits at the end of the while loop that dictates the strategy. We tested values of 0.3, 0.5, and 0.7 seconds, with 0.5 being the default. Lower values increase responsiveness but lead to excessive order churn (our orders do not exist long enough to build up price-time priority). We found that the value of 0.3 was best, not because it increased returns (NLV mean was slightly lower than that of 0.5) but because it seemed to reduce the variance in returns, as illustrated by the chart below.

Final Value: `SLEEP_TIME = 0.3`

Graph:



Strategic Interaction with Other Algorithms

Our algorithm is designed to operate effectively in an environment with competing market participants. While the ALGO2 case does not explicitly simulate adversarial strategies, we assume the presence of other algorithms submitting and canceling orders in real time, which creates dynamic liquidity conditions. To mitigate the risk of being adversely selected or "picked off" in rapidly shifting markets, our strategy includes an order cancellation mechanism (orderslimit) that maintains only a limited number of active orders and refreshes them at a fixed interval (SLEEP_TIME). This reduces exposure to stale quotes and helps us stay near the front of the order queue.

In trending markets, our algorithm dynamically adjusts order size and order placement based on the current inventory level. When inventory grows in one direction, we reduce order size and increase our passive spread slightly to allow the market to come to us, thus minimizing the risk of accumulating more one-sided inventory. This behavior indirectly simulates a defensive posture against potential algorithmic aggression or unfavorable flow. Through reducing inventory risk, we don't have to worry extensively about malicious programs that manipulate prices.

Ethical and Legal Considerations of Autonomous Algorithms

The rise of autonomous trading algorithms raises important ethical and regulatory concerns, particularly around the blurry line between strategic behavior and market manipulation. In practice, it is difficult to infer the intent behind an algorithm's actions—especially those based on machine learning. For example, if a reinforcement learning agent discovers that posting and then canceling large orders moves the market favorably, it may adopt this behavior without any human explicitly coding manipulation. In order to counteract this, we believe that the outcome matters more than intent, particularly in situations where participants have the power to cause widespread consequences even without mal-intent.

Therefore, algorithm designers must be intentional about embedding ethical constraints into their systems. This includes avoiding strategies that exploit latency differences, cancel orders with no intent to trade, or overload the system with excessive messaging. Especially with the rise of AI, which is usually considered to be a black box, regulators should focus on outcomes rather than intentions so that designers intentionally embed these constraints.

Adaptability and Future Improvements

The algorithm can adapt to different fee structures by adjusting its parameters. For markets with high rebates, we would increase order frequency and volume to maximize rebate collection. For the ALGO2e case with multiple securities, we would implement security-specific processing and portfolio-level position management.

Future enhancements could include deeper order book analysis and adaptive parameter adjustment based on observed volatility, further increasing adaptability while maintaining core risk management principles. Additionally, we would love to increase the sample size from 5 to something with strong statistical significance, at least 10+, though we didn't have enough time this time. Trying out finer parameter values would have also been interesting and useful.