Deriving Insights and Strategies from Exchange Data Feeds: CFE VIX futures contracts

Y. Huang, W. Liu, J. Ru, P. Sojic, T. Tan, R. Wang

DV Trading - Uchicago Project Lab

December 12, 2024

Acknowledgements.

- We would like to give our special thanks to Alex Gerasev, our project sponsor, who gave us insightful lessons within the strategies used within high frequency trading.
- We would also like to thank our program advisor, Sebastien Donadio, for providing helpful advisory with regards to technical solutions and strategies within optimizing our insights.
- Lastly, thank you to everybody for brainstorming and implementing efficient solutions to the different problems and challenges this semester, and for being helpful partners.

Overview

Decode the Exchange Data Feed

Protocol Structure

- The CFE PITCH protocol is a low-latency, direct market data feed.
- It provides real-time updates on order book changes.
- Data is transmitted as a series of time-stamped binary messages.
- Each message corresponds to a specific event type (e.g., Add, Executed).
- Inverstors parse these messages to reconstruct and maintain the order book state.

Example: Visualizing an OrderExecuted Message

Offset (Bytes)	Field	Description
0	Length (1)	Length of the message in bytes: 27
1	Message Type (1)	Order Executed in this case
2–5	Time Offset (4)	Nanosecond offset from last unit timestamp
6–13	Order ID (8)	Unique ID of the order being executed
14–17	Executed Qty (4)	Number of contracts/shares executed
18–26	Execution ID (8)	CFE unique-to-day ID for the execution event
27	Trade Condition (1)	Normal, Opening, or Spread

Process Exchange Data Streams

Order Executed Struct Definition

```
// Struct Definition
struct [[gnu::packed]] OrderExecuted // 0x23
{
    uint32_t TimeOffset;
    uint64_t OrderId;
    uint32_t ExecutedQuantity;
    uint64_t ExecutionId;
    uint8_t TradeCondition;

    string txt(){
        // STRING METHOD
    }
};
static_assert(sizeof(OrderExecuted) == 25, "
    OrderExecuted struct must be 25 bytes");
```

*gnu::packed is mandatory for ensuring that struct padding does not occur and mess up packet processing when reading pcap.

Key Operations: Order Book Level

- **Add**: Introduces a new order into the book, for a given side, quantity, and price.
- **Reduce**: Reduces the given quantity of the given order. Can not fully empty order out, rendering it size 0 as per CFE specs.
- **OrderExecuted**: Executes a given amount of the order. Can be partially or fully executed.
- **Delete**: Completely removes an order from the order book
- **Modify**: This operation is complex:
  - Same Price, Lesser Quantity: like Reduce. Keeps priority.
  - Otherwise: Treated as a Delete, then Add. Loses priority.
  - Key: no-op (no change in anything) causes a loss of priority. Used by some MMs and HFTs for measurement.

Some Considerations about PITCH

- Sometimes we can get Long vs the normal Short messages. Ensure Long messages are handled effectively by using UINT_64T and UINT_32T.
- Unsigned integers are prone to overflow: Reducing a quantity below 0 results in very unexpected overflow behaviour: (i.e. why did our arbitrage earn 4 trillion dollars?)
- Sanity Check: If a reduce, execute, or modify results in a quantity that would be below 0, this indicates your order book has fallen out of sync with the exchange due to an erroneous operation. (Fatal Error)
- CFE is a FIFO Exchange: if you're not executing the first order in queue at the price level, this is another fatal error.

PITCH Protocol: Ticker Management

The exchange does not use a human-readable ticker format.

- For example: VXV4 is "000XIZ" within PITCH.

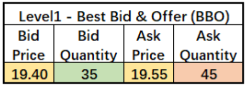
Solution: store ticker table when FuturesInstrumentDefinition messages come in.

- Allows us to store tick size and contract size.
- Useful also for constructing the opposite map: human-readable to exchange.
- The former allows easy strategy backtesting by not having to input exchange tickers when specifying which instruments to backtest.

Order Book Design

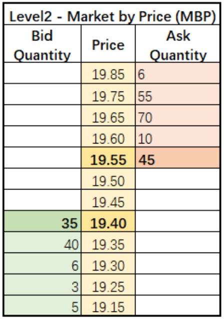
Order Book Level

- **Market By Price (MBP)**
  - Tracks the total quantities available at each price level.
  - When processing messages, modifies only the quantities at the affected price level.
  - Efficiently aggregates data without storing individual orders.
- **Market By Order (MBO)**
  - Stores individual orders for each price level in a FIFO queue.
  - Process messages by appending new orders to the queue(ADD) or adjusting specific orders within the queue; Need to sum them up to get the total quantities at a given price level



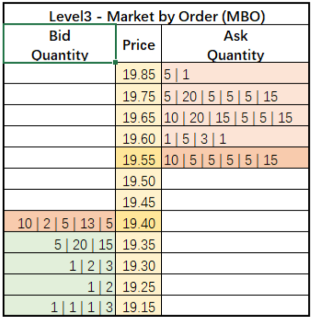
Level1 - Best Bid & Offer (BBO)

Bid Price	Bid Quantity	Ask Price	Ask Quantity
19.40	35	19.55	45



Level2 - Market by Price (MBP)

Bid Quantity	Price	Ask Quantity
	19.85	6
	19.75	55
	19.65	70
	19.60	10
	19.55	**45**
	19.50	
	19.45	
35	**19.40**	
40	19.35	
6	19.30	
3	19.25	
5	19.15	



Level3 - Market by Order (MBO)

Bid Quantity	Price	Ask Quantity
	19.85	5 | 1
	19.75	5 | 20 | 5 | 5 | 5 | 15
	19.65	10 | 20 | 15 | 5 | 5 | 15
	19.60	1 | 5 | 3 | 1
	19.55	10 | 5 | 5 | 5 | 5 | 15
	19.50	
	19.45	
10 | 2 | 5 | 13 | 5	19.40	
5 | 20 | 15	19.35	
1 | 2 | 3	19.30	
1 | 2	19.25	
1 | 1 | 1 | 3	19.15	

Pros and Cons of MBO

Pros	**Cons**
Provides granularity to check signals at order granular level	More overhead when updating instead of simple $O(logn)$ accesses at L2
Allows for visualizations to show queue and order position	Added complexity for finding simple arbitrages or signals
Data integrity cross checkable with order execution logic	More complex design slows signal testing development
more detailed, slower	**can be less detailed, faster**

Table: Comparison of Pros and Cons

Code/System Optimization

Optimization - Compiler Flags and Low Hanging Fruit

- GCC offers a series of flags for optimization
- -O2 and -O3 offer very small performance differences
- -march=native compiles for native architecture. Good to include.
- other options like -Ofast risk floating point computation inaccuracy and generally should not be used.
- observed no optimization when placing functions inline with -finline-functions.
- -O3 will enable most efficiency options that can be declared in this case.

Optimization - Multithreading

This solution is an **embarrassingly parallelizable** problem. However, parallelization on a single day that has sequenced transactions is difficult.

- Using locks to ensure proper sequence: slower due to blocking.
- Dividing the workflow between symbols requires knowing the frequency of activity to equally divide the work.

There are 5 trading days a week and 252 trading days a year.

- CPU has 5 threads $\Rightarrow$ you can split weekly data by day.
- CPU has 252 threads $\Rightarrow$ you can split data for a year by day.
- Parallelizing weekly workflow yields speed up from 19 seconds to around 5 seconds.

Types of HFT Strategies

High Frequency Basics

Here are a couple of strategies we can use at a high frequency level.

1. Sniping: When an opportunity like an arbitrage exists, exploit it by executing the trade.
2. Event-Driven: Given an external signal or other key event that happened, measure price movement to determine if a trade could be profitable.
3. Microstructure: Given flow of certain trades, identify potential signals within price movements post-signal.

All strategies are naturally reliant on low latency and strong backtesting.

Exploit Arbitrage Opportunities

Mini Contract vs Normal Contract: Size Arb

Define the Size Arbitrage as between a contract of VXA, called X, and a contract of VXMA, called Y, where A is the month and year code of the contract, and the VXMA contract (X) is $\frac{1}{D}$ of the size of the VXA contract (Y) as the following static portfolio:

$$\hat{\Theta} = \begin{bmatrix} x & \text{(units X)} \\ -Dx & \text{(units Y)} \end{bmatrix}$$

Then the Arbitrage profit amount is determined by $\max(a), \Pi_{ARB} = aX_t\hat{\Theta} \geq 0$ where X_t is the price at time t for the given contracts. a is a weight that maximizes profit while the arbitrage is still profitable

Spread Contracts: Triangle Arb

Define the Triangle Arbitrage as between a spread contract of -CVXA+DVXB, called X, where C and D indicate leg size and A and B are different month and year codes, as a static portfolio as follows, and the individual contracts of VXA and VXB as Y and Z respectively:

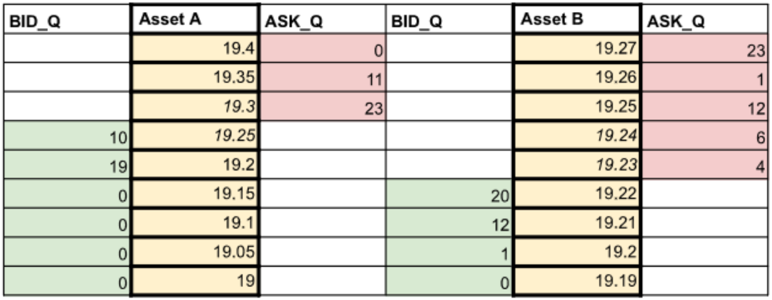
$$\hat{\Theta} = \begin{bmatrix} x \quad \text{(units X)} \\ Cx \quad \text{(units Y)} \\ -Dx \quad \text{(units Z)} \end{bmatrix}$$

Then the Arbitrage profit amount is determined by $\max(|a|), \Pi_{ARB} = aX_t\hat{\Theta} \geq 0$ where X_t is the price at time t for the given contracts.

Example

We must consider arbitrages over all price levels!

- Consider the scenario where the contract size of A is 100 and B is 1000, and the following order book state occurs:



BID_Q	Asset A	ASK_Q	BID_Q	Asset B	ASK_Q
	19.4	0		19.27	23
	19.35	11		19.26	1
	19.3	23		19.25	12
10	*19.25*			*19.24*	6
19	19.2			*19.23*	4
0	19.15		20	19.22	
0	19.1		12	19.21	
0	19.05		1	19.2	
0	19		0	19.19	

- Arbitrage: buy 4 A at 19.23 and 6 at 19.24; sell 1 B at 19.25
- Profit: $(0.02 \times 4 + 0.01 \times 6) \times 100 = 14\$$.

Arbitrage Backtesting

Challenge:

- Exploiting an arbitrage opportunity in a backtest removes it from the market.
- Real executions may occur, then how should we accurately reflect that in the order book?

Solution:

- Directly execute the order in the backtest and update the *main order book* accordingly.
- Create a *virtual order book* that performs the *opposite* of our executed trades.
- One advantage is, if we aggregate the main orderbook and the virtual order book, we can get the *real historical order book*.
- When real executions occur:
  - Attempt execution in the main order book first.
  - If quantities are insufficient, execute the remainder in the virtual order book.
- This approach previews the price dynamics of competing arbitrageurs ahead of time.

Arbitrage Backtesting Results

Within Week 41, we found the following amount of arbitrage:

- VX-VXM Arbitrage: $ 2826
- -1/+1 Spread Arbitrage: $ 9240

Generally, pure arbitrage is highly competitive: Most disappear within 100-200 microseconds.

Develop and Evaluate Trading Signals

Trading Signals

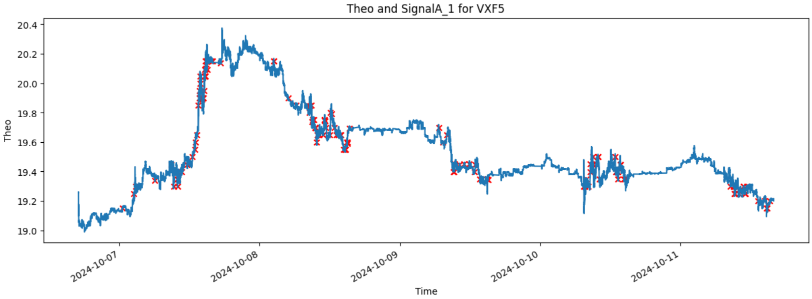
- A signal is defined as a binary indicator. It is considered significant if, at each occurrence at time t, and for a given timeframe ΔT, it consistently precedes a substantial and statistically significant theoretical price movement.
- The term "theoretical price movement" refers to the weighted price that reflects the likely price level, calculated as follows:

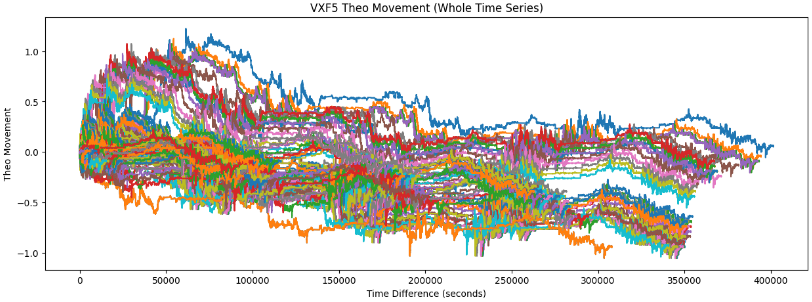
$$P_{Theo} = \frac{P_{Bid} \times Q_{Ask} + P_{Ask} \times Q_{Bid}}{Q_{Ask} + Q_{Bid}}$$

Evaluation Process

1. Identify and record all occurrences of the signals $\mathbf{T} = [T_{s_0}, T_{s_1}, \dots, T_{s_n}]$, where each T_{s_i} marks the time a signal s_i is triggered.
2. For the series of start times $\mathbf{T}$, measure the price movement following each signal. Specifically, for each each T_{s_i}, construct the corresponding price time series $\mathbf{P}_{s_i} = [p_{T_{s_i}}, p_{T_{s_i}+1}, \dots, p_{T_{\text{END}}}]$, where $p_{T_{\text{END}}}$ represents the price at the end of the trading period.
3. Given these time series, realign the starts of the time series to each instance of the beginning of the signal triggering, and measure overall price level movement. This creates a series of paths starting at 0, but diverging as the signal instances perform differently over time.
4. Slice the timeframes at the given intervals, for example, $100ms$, $1s$, $1H$, and conduct statistical testing on the significance of the movements.

Evaluation Process





T-Test on Signals

The t-test evaluates whether the theo movements show a significant deviation from the hypothesized mean, which is 0.

- Null Hypothesis: $H_0 : \bar{X} = 0$
- Test Statistic: $t = \frac{\bar{X}}{\frac{s}{\sqrt{n}}}$
- P-Value and Decision Rule:
  - **Two-Tailed Test**: $p_{\text{two-tailed}} = P(T \geq |t_{\text{obs}}|) + P(T \leq -|t_{\text{obs}}|)$;
    Reject H_0 if $p < \alpha$.
  - **Upper-Tail Test**: $p_{\text{upper-tail}} = P(T \geq t_{\text{obs}})$;
    Reject H_0 if $p < \alpha$.
  - **Lower-Tail Test**: $p_{\text{lower-tail}} = P(T \leq t_{\text{obs}})$;
    Reject H_0 if $p < \alpha$.

Binomial-Test on Signals

The binomial test is used to determine whether the proportion of positive movements differs significantly from 50%.

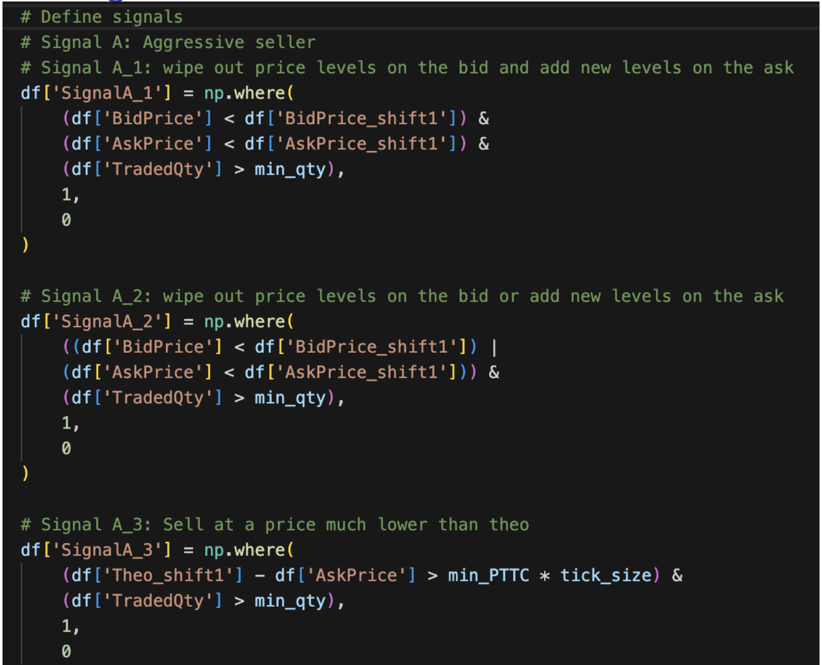
- Advantages:
  - Does not assume normality of the data.
  - Useful when the distribution of theo movements is skewed or contains outliers.
- Null Hypothesis: $H_0 : p = 0.5$
- Binomial Distribution: $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$
- P-Value and Decision Rule:
  - **Upper-Tailed Test**:
    $p = P(X \geq k_{\text{obs}}) = \sum_{k=k_{\text{obs}}}^{n} \binom{n}{k} p^k (1-p)^{n-k}$;
    Reject H_0 if $p < \alpha$.
  - **Lower-Tailed Test**:
    $p = P(X \leq k_{\text{obs}}) = \sum_{k=0}^{k_{\text{obs}}} \binom{n}{k} p^k (1-p)^{n-k}$;
    Reject H_0 if $p < \alpha$.

Explore Signals at the BBO Level

- **SignalA - Aggressive Seller**
  - SignalA_1: Indicates a scenario where both the bid and ask prices move down in a single transaction.
  - SignalA_2: Indicates a scenario where at least the bid price or the ask price moves down.
  - SignalA_3: Indicates a scenario where there's a trader selling at a price "much" lower than `Theo`.(The threshold here refers to how many ticks it crosses below the pre-trade Theo)
- **SignalB - Aggressive Buyer**
  - SignalB_1: Indicates a scenario where both the bid and ask prices move up in a single transaction.
  - SignalB_2: Indicates a scenario where at least the bid price or the ask price moves up.
  - SignalB_3: Indicates a scenario where there's a trader buying at a price "much" higher than `Theo`.(The threshold here refers to how many ticks it crosses above the pre-trade Theo)

Additionally, the definition of signals can also take into account constraints, such as traded quantities, trading time, and more.

Explore Signals at the BBO Level



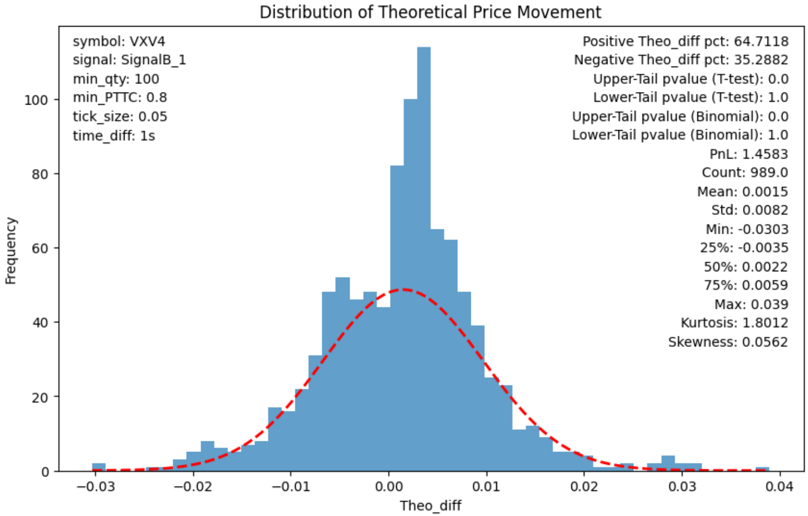
```
# Define signals
# Signal A: Aggressive seller
# Signal A_1: wipe out price levels on the bid and add new levels on the ask
df['SignalA_1'] = np.where(
    (df['BidPrice'] < df['BidPrice_shift1']) &
    (df['AskPrice'] < df['AskPrice_shift1']) &
    (df['TradedQty'] > min_qty),
    1,
    0
)

# Signal A_2: wipe out price levels on the bid or add new levels on the ask
df['SignalA_2'] = np.where(
    ((df['BidPrice'] < df['BidPrice_shift1']) |
    (df['AskPrice'] < df['AskPrice_shift1'])) &
    (df['TradedQty'] > min_qty),
    1,
    0
)

# Signal A_3: Sell at a price much lower than theo
df['SignalA_3'] = np.where(
    (df['Theo_shift1'] - df['AskPrice'] > min_PTTC * tick_size) &
    (df['TradedQty'] > min_qty),
    1,
    0
```

Signal Evaluation

- **Parameters to Test:**
  - **Time Difference (ΔT):** Ranges from 500 ms to 1 hour.
  - **Traded Quantity:** Ranges from 100 to 3000.
  - **Pre-trade Theo Cross (in ticks):** Between 0.8 and 1.0.
  - **Trading Time:** Active vs. inactive trading hours.
- **Performance Matrix:**

Statistic	**Description**
Distribution	count, mean, std, min, 25th percentile, median, 75th percentile, max, kurtosis, and skewness.
Positive (negative) theo_diff pct	The percentage of positive (negative) theoretical price movements. Larger deviations from 50% indicate better performance.
Upper (Lower) tail p-value - t test	If $p < 0.05$, the mean value is significantly higher (lower) than zero.
Upper (Lower) tail p-value - binomial	If $p < 0.05$, the percentage of positive (negative) theoretical price movements is significantly higher (lower) than 50%.
PnL	mean $\times$ count.

Signal Evaluation - An Example



Signal Performance Across Parameters

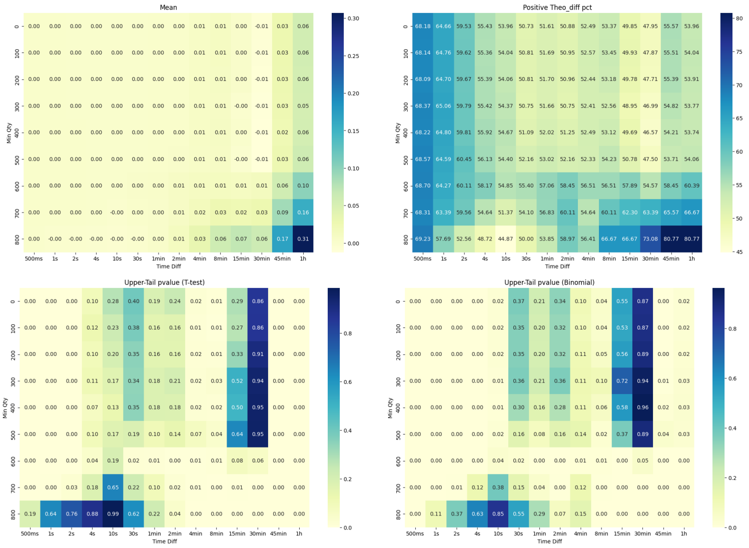


Figure: Signal Performance across time differences and traded quantities for Signal B1, VXV4, 7:00 - 14:00

Signal Performance Across Parameters

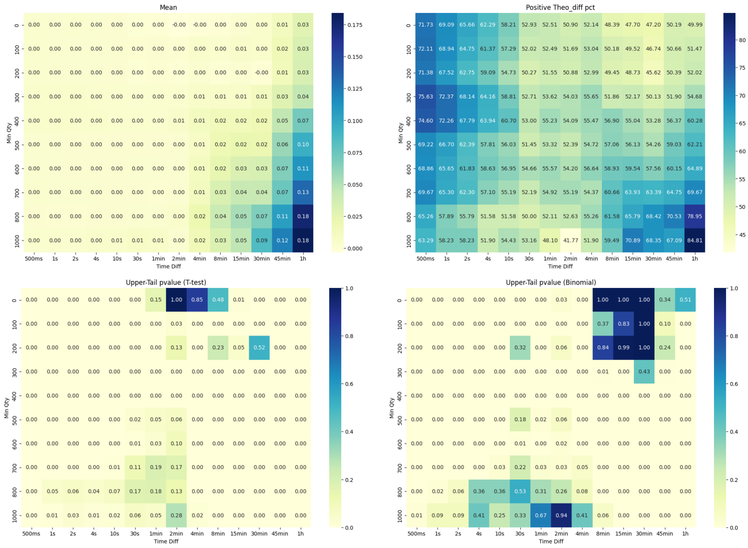


Figure: Signal Performance across time differences and traded quantities for Signal B2, VXV4, 7:00 - 14:00

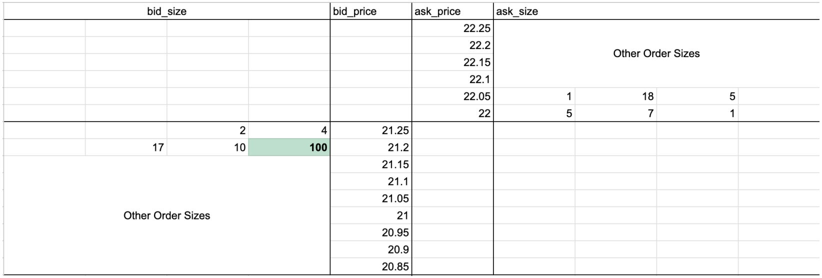
Future Exploration

Improving Efficiency of Code

1. Sniping arbitrages are best calculated by simply using MBP level data. MBO level sniping is slow and space intensive.
2. Use MBO data stream to track microstructure-level signals by analyzing transactions with current MBO/MBP book state.
3. Further Parallelize the data: analyze relative frequency of market activity for each symbol, divide workload equally where each threads get assigned certain symbols.
4. Further optimize order book design. Minimize use of lists and dynamic data structures by using fixed-size alternatives.
5. Tune hash functions,

Explore Signals at the MBP/MBO Level

1. Abnormal bid/ask size that was placed not at the best bid/ask price



bid_size				bid_price	ask_price	ask_size			
					22.25	Other Order Sizes			
					22.2				
					22.15				
					22.1				
					22.05	1	18	5	
					22	5	7	1	
		2	4	21.25					
	17	10	**100**	21.2					
Other Order Sizes				21.15					
				21.1					
				21.05					
				21					
				20.95					
				20.9					
				20.85					

2. More types of order: e.g. Stop Limit Order and Spread Order. Ref: https://cdn.cboe.com/resources/regulation/rule$_b$*ook/cfe − rule − book.pdf*, *P*52

Signal ideas

1. Large ADD/DELETE orders deep in the book (+- 5 levels from BBO)
2. Partial/complete wipe-out at a fractionned BBO
3. Large modify from one level to the other

Future Roadmap

1. Redesign and improve efficiency of Order Book and test for improvements through common backtest framework.
2. Redesign sniping matching for multiple instruments. Build efficient signal generation and backtesting frameworks.
3. Build in snapshots. For example: for a trading day, have a snapshot for each hour and store transactions for the next hour. Enables us to recreate any order book state given day, last hour state, and transactions up to that timestamp.
4. Examine visualizations: using certain visualization tools and snapshots, build efficient visualizations akin to BookLens.
5. Examine infrastructure level changes: disable syscalls, reduce interrupts, kernel level changes. (Credit: DataBento)

Thank you!