Derivatives Case

Cornell Trading Competition

In this game, teams will analyze evolving data in real-time to execute profitable trades on derivative contracts. The underlying that all derivatives will be derived from will be some random dice rolls (from an atypical die).

1. Game Overview

There will be n rounds in this case, where n is some positive integer. Before each round begins, participants will receive unique datasets of **2,000 dice rolls** drawn from the underlying distribution for the round's die (**historical/training)**. The dice used in each round of the game will follow the same distribution, but the distribution will change between rounds. Strategies should not depend on the value of n.

Each round will have 10 subrounds. There will be 2,000 dice rolls per subround, so there are 20,000 total dice in consideration when computing the final fair value of each derivative. In each subround, participants can trade European calls, European puts, and futures, which all expire at the end of the current round (i.e., the 10th subround). A European option is an option that can **only** be exercised at **settlement**.

The underlying value of all rounds will be the **sum** of all **20,000 dice** rolled within the round. Note that the final fair value will be confirmed before the strategies execute in the 10th subround.

The task is to submit bid-ask spreads (markets) on the fair value of the underlying at the end of the round to maximize PnL (profit and loss) and make markets that are easily matched by others' markets. A team's markets will be matched by other teams' markets, or by preprogrammed bots' markets. Markets must be well-formed; that is, a market's bid must be smaller than its ask (example: (10, 20) is allowed, but (20, 10) is not). All products will be executed at mid price (abiding by a typical mid-price execution). This means that the midpoint between the bid (buy) and the ask (sell) points between two teams will be what the price of the trade.

Example: A has a market of (3, 7), B has a market of (8, 10). B buys from A at 7.5.

Here is an example of how this game will play out:

```
Teams are given 10,000 dice rolls from a distribution
Round 1 Begins
        Subround 1.1 Begins
                2,000 more dice are rolled
                Strategies are executed
                Results are computed and outputted
        Subround 1.1 Ends
        Subround 1.2 Begins
                2,000 more dice are rolled
                Strategies are executed
                Results are computed and outputted
        Subround 1.2 Ends
        .
        .
        .
        Subround 1.10 Begins
                2,000 more dice are rolled
                Strategies are executed
                Results are computed and outputted
        Subround 1.10 Ends
Round 1 Ends
Teams are given 10,000 dice rolls from a different distribution
Round 2 Begins
        Subround 2.1 Begins
                .
                .
                .
        Subround 2.10 Ends
Round 2 Ends
.
.
.
Teams are given 10,000 dice from a final distribution
Round n Begins
        .
```

.

.

Round n Ends

2. Logistics

In each subround, teams can only trade at most 1 market per instrument. In other words, in each subround, teams can only trade at most 1 market on any combination of strike price and product. If s is the number of strikes, the maximum number of markets a team can submit in a subround is 2s+1 (up to s calls, up to s puts, and up to 1 future). After each subround, all teams will be able to see prior trade prices and overall positions. Teams will also be able to see their positions and PnL after each subround.

3. Evaluation

Each team's performance will be scored based on two criteria:

1. **Profit & Loss (PnL):** How profitable the team's strategy was across rounds.
2. **Market Engagement:** How many trades (markets matched) the team was able to execute across rounds.

The goal is to reward strategies that are both profitable (maximize PnL) and liquid (maximize number of markets matched), balancing aggressive trading with good market-making.

We will score participants based on the following formula:
Let p_r represent the team's rank against all participants (excluding bots) in PnL, in a given round r.

- $p_r = 1$: The team achieved the highest PnL in the round (best possible outcome).
- $p_r = N$: The team achieved the lowest PnL in the round (worst possible outcome), where N is the total number of teams.

Let m_r represent the team's rank against all participants (excluding bots) in the number of markets matched, in a given round.

- $m_r = 1$: The team's markets matched the most trades in the round (best possible outcome).
- $m_r = N$: The team's markets matched the fewest trades in the round (worst possible outcome), where N is the total number of teams.

$$\text{Final Score} = \frac{3}{4} \cdot \left(\frac{1}{10} \sum_{r=1}^{10} p_r \right) + \frac{1}{4} \cdot \left(\frac{1}{10} \sum_{r=1}^{10} m_r \right)$$

Essentially, your team's objective is to submit at most 1 market for each instrument that maximizes the PnL generated and the number of markets you can match per round.

4. Deliverables

Each team must submit a Python file titled **`strategy.py`** that implements the provided **`AbstractTradingStrategy`** class via Gradescope. Please refer to the gradescope course entry code and sample submission file below.

The sample submission contains an example trading strategy and includes an example of a trading strategy, implementing the abstract methods of **`AbstractTradingStrategy`**. See the end of this document for some additional details on the underlying dataclasses used in the codebase as well as documentation for the abstract class you need to submit.

Gradescope:
Course ID: 1157932
Entry Code: X2WYR4

Example File:
EXAMPLE TEAM SUBMISSION

Warning: Your trading strategy must not take more than **2 seconds** to run per round. That is, your **`make_markets`** method (which is given the marketplace, training rolls, current rolls, your trades in the round, and the round data) must not take longer than 2s to return all of your markets. If it takes longer than that, you will not trade in that round.

Additionally, all other methods are expected to take no longer than **1s** to run (**on_game_start, on_round_end, on_game_end**). **Be aware of the scales of data mentioned above for the dice rolls.** Additionally, we have around 150 competitors in teams of 4 that you can trade with, as well as a smaller number of bots you can trade with.

There will be **two submission deadlines**:

- **Early Deadline:** October 12, 2025, at 11:59 PM

    - Strategies submitted by this deadline will be run locally, and teams will receive compilation feedback on October 13, 202,5 at 1:00 PM.
    - This feedback is intended solely as a **sanity check**. If a strategy compiles successfully on Gradescope, it is essentially guaranteed to compile when we run it (≈99% certainty).

- **Final Deadline:** October 15, 2025, at 11:59 PM
    - Strategies submitted after the early deadline but before the final deadline will still be accepted for the competition, but **no feedback will be provided**.

Teams may submit multiple times to Gradescope, and we will always use the most recent submission received before the final deadline.

5. Terminology

- **Underlying Value:** The total sum of all dice rolls in a round. This is the asset on which all derivatives (calls, puts, and futures) are based.
- **European Call Option (Call):** An option to buy an asset at a predetermined price and on a predetermined date (or equivalently, subround).
- **European Put Option (Put):** An option to sell an asset at a predetermined price and on a predetermined date.
- **Future:** An agreement to buy or sell an asset at a predetermined price and on a predetermined date.
- **Expiration:** The point at which a derivative contract settles. In this game, all contracts expire at the end of the round (after the 10th subround).
- **Premium (Option Price):** The value of an option (call or put) at the time of trade, determined by the market. In this game, options are traded via bid/ask spreads and executed at the midpoint.
- **Strike Price:** The strike price of the option contract.
- **Bid/Ask Prices:** The best available prices at which a market participant is willing to buy or sell.
- **Liquidity / Market Matching:** A measure of how frequently a team's posted markets are matched against other participants or bots.
- **Well-Formed Market:** A market whose bid is strictly lower than its corresponding ask.
- **PnL (Profit and Loss):** The net financial outcome of all trades made, calculated at the end of each round once the underlying value is revealed.

6. Competition Day

On the day of the competition, you will be given an opportunity to improve your strategy with additional information provided at every round. This will be explained and clarified on the actual competition day.

Appendix: Data Classes & Documentation:

The starter python code should give you enough to get going, but if you want some more details on some of the underlying data classes, here are the classes for Product and Trades for clarity, as well as the abstract class `AbstractTradingStrategy`.

Note: you are welcome to add helper functions / classes if you so desire to your implementation of `AbstractTradingStrategy`, but make sure you are aware of what is expected to be implemented.

```python
class Trade:
   def __init__(self, product_id: str, quantity: float, price: float,
                round_traded: int, buyer_id: str, seller_id: str):
       self.product_id = product_id
       self.quantity = quantity
       self.price = price
       self.round_traded = round_traded
       self.buyer_id = buyer_id
       self.seller_id = seller_id
@dataclass
class Product:
   """Naming convention for product id:
       Settlement, Future or Call/Put Option, Settlement (Expiry) Round, Price

       Settlement = S or P => Sum or Product

       Examples:
       - Future: S,F,5
       - Call Option: S,C,5,55
       - Put Option:  S,P,5,45
   """
   product_id: str


class AbstractTradingStrategy(ABC):
```

```python
"""
Abstract trading strategy that students must implement.

Students will subclass this and implement the required methods.
The autograder will pass live game objects at runtime via dependency injection.
"""

@abstractmethod
def on_game_start(self, config: Dict[str, Any]) -> None:
    """
    Called once at the beginning of each game.

    Args:
        config: Game configuration including:
            - seed: Random seed for reproducibility
            - num_rounds: Total number of rounds in the game
            - team_name: Your team's identifier
            - dice_sides: Number of sides on the dice
    """
    pass

@abstractmethod
def make_market(
    self,
    *,
    marketplace: Any,
    training_rolls: Any,
    my_trades: Any,
    current_rolls: Any,
    round_info: Any
) -> Dict[str, Tuple[float, float]]:
    """
    Make trading decisions for the current sub-round.

    This is the main method where students implement their trading logic.

    Args:
        marketplace: Live marketplace object with methods like:
            - get_products() -> List[Product]: Get all tradeable products
            - get_die_num_faces() -> int: Get number of dice faces
        training_rolls: Historical dice rolls for learning (10,000)
        my_trades: Your current trading positions and history (List[Trade])
```

```python
            current_rolls: Dice rolls in the current round so far
            round_info: Information about current round state:
                - current_sub_round: Current sub-round number
                - num_sub_rounds: Total sub-rounds in this round
                - round_id: Identifier for this round

        Returns:
            Dict mapping product_id -> (bid_price, ask_price)
            Example: {"S,F,6": (95.5, 104.5), "S,C,100,2": (2.0, 8.0)}

            Empty dict {} means no quotes for any products
        """
        pass

    @abstractmethod
    def on_round_end(self, result: Dict[str, Any]) -> None:
        """
        Called at the end of each round with settlement results.

        Args:
            result: Round results including:
                - dice_rolls: Final dice rolls for the round
                - settlements: Settlement values for each product
                - your_pnl: Your profit/loss for this round
                - your_trades: Summary of your trades this round
        """
        pass

    @abstractmethod
    def on_game_end(self, summary: Dict[str, Any]) -> None:
        """
        Called at the end of the entire game.

        Args:
            summary: Final game summary including:
                - total_pnl: Your total profit/loss across all rounds
                - final_score: Your final performance score
                - num_trades: Total number of trades you made
                - performance_metrics: Additional performance statistics
        """
        pass
```