CASE 1: OPTIONS MARKET-MAKING

1. Introduction

This case tests the team's ability to adapt to changing volatility conditions and manage risk while continually making markets in multiple options on the same underlying product.

Each team will be competing in its *own* market in which it is the *sole* market-maker, and will be required to at all times post limit orders of quantity one *both* buy and sell options at each of five strikes. At evenly spaced time-intervals, a broker will submit a single immediate-or-cancel order of quantity one, which may be either a buy or a sell order at any of the five strikes. If the broker's order hits or lifts one of the market-maker's limit orders, a trade will occur. The market-maker will *only* know the broker's quote when a trade takes place. When you receive non-marketable quote or two, you should therefore probably widen the spread to search for current price level. The market-maker will have the opportunity to adjust her markets before the next broker order.

The underlying will not move over the course of the competition, nor will the time-to-expiration interest rate change. The changes in the prices offered by the broker will be primarily determined by the volatility parameter, $\sigma_t^{(b)}$, in the Black-Scholes formula, which will be generated from a stochastic process. The main tasks of each team's algorithm will be to attempt to track $\sigma_t^{(b)}$ and adjust its quotes accordingly to make profit while managing inventory risk.

This case will consist of three rounds with 100 broker orders each. Teams will have time between rounds to make changes to their parameters used in the algorithms if they choose.

2. The Broker Orders

Each broker order will be randomly generated from the distribution described below. Each order is a triple $O_t = (price_t, strike_t, direction_t)$, i.e., ($\omega$, 90, -1) would mean a broker order to *sell* the 90 strike at ω. The underlying price is fixed at $X = 100$, the five available strikes are {80, 90, 100, 110, 120}, interest rate is fixed at $r = 0.01$, time-to-expiration is fixed at $L = 1$, and the distributions of the elements of O_t are:

$$direction \overset{iid}{\sim} Uniform(\{-1, 1\})$$

$$strike \overset{iid}{\sim} Uniform(\{80, 90, 100, 110, 120\})$$

$$price_t = BlackScholes^{call}(X, strike_t, r, L, \sigma_t^{(b)}) \cdot e_t \cdot \epsilon_t$$

where e_t reflects the 5% edge the broker is willing to give to the market-maker and $e_t = 1.05$ or 0.95 in the case of a broker buy or sell order, respectively. ϵ_t serves as the noise parameter drew from $N(1, 0.05^2)$. $\{\sigma_t^{(b)}\}_1^T$ is a set of discrete draws from a geometric Brownian motion with zero drift, volatility 0.05, and initial value of 0.3.

In each round, all teams will face the same realization of the broker order process.

3. Penalty

To prevent excessive risk-taking, teams will be required to adhere to vega limits throughout all rounds of the case.

The vega limit will be 5 times the calculated vega of the 100-strike option with $\sigma = 0.3$. You will be charged one hundred dollars times the amount by which your position vega exceeds the vega limit. Note that although the vega limit is fixed, your position vega is calculated based on the current true $\sigma_t^{(b)}$ which you have to track. Further, the penalty will continue to apply until your position vega is below the limit.

4. Scoring

Participants' scores in each round of the case will be determined based on PnL and any penalties incurred, where PnL is defined to include profits and losses from closed positions, as well as profits and losses from any open positions at the end of the round marked to $price_{100}$ of that round.

The weighting of the scores from each round to calculate the final score of the case will be announced prior to the start of the competition.

5. Case Objects & Interface

The following Java class objects are implemented in the util package and should be used in your program based on the interface defined below:

```
public static class Quote {
    public Quote(double bid, double offer)
}

public static class QuoteList {
    public QuoteList(Quote quoteEighty, Quote quoteNinety,
                     Quote quoteHundred, Quote QuoteHundredTen, quoteHundredTwenty)
}
```

You will be writing implementations in Java for the following methods:

```
public QuoteList getCurrentQuotes();
```

The implementation of this method should be ready to provide current market quotes at any point in time. Broker request submitted at each time interval will cause this method to be called. The name of the `Quote` variable within `QuoteList` indicates the strike. Therefore, please provide your price with the correct corresponding `Quote` object.

The brokers orders will be compared to the quotes returned from the above method and any trades will be notified through the following two functions.

```
public void newFill(int strike, int direction, double price);

public void noBrokerFills();
```

If the QuoteList object returned in `getCurrentQuotes` matches against any broker orders, the `newFill` method will be invoked with trade details. The `strike` argument indicates which option was filled. Definition of `direction` will stay consistent as before in reference to the broker, i.e., -1 would be mean that you have *bought* one lot with a broker *sell* order.

If the QuoteList object returned in `getCurrentQuotes` does not result in a trade, the `noBrokerFills` method will be invoked so that your implementation can make any necessary adjustments.

Any penalty incurred will be announced through the following function.

```
public void penaltyNotice(double amount);
```

The `amount` argument reflects the monetary amount of the penalty that the participants have incurred in that particular round.

In addition, we have also provided you with the Black-Scholes option pricing function and vega calculation function in `OptionsMathUtils.java`:

```
public static double theoValue(double strike, double vol)
public static double calculateVega(double strike, double vol).
```