Option Chain Fetcher Usage Guide

Overview

The `option_chain_fetcher.py` script fetches SPX option chain data including Greeks (delta, gamma, theta, vega, rho) and bid/ask quotes, then stores it in a SQLite database with timestamps.

Important: When fetching SPX data, the script automatically retrieves **BOTH** `SPX` (monthly/quarterly expirations) and `SPXW` (weekly expirations) option chains. This ensures complete coverage of all available SPX options.

Prerequisites

1. Ensure your TastyTrade credentials are configured in `.env` file:
   - `TT_API_CLIENT_SECRET`
   - `TT_REFRESH_TOKEN`
2. Install dependencies:

```
pip install -r requirements.txt
```

Usage

Basic Command

```
python option_chain_fetcher.py --max_dte <days> --min_strike <price> --
max_strike <price>
```

Parameters

- `--max_dte`: Maximum days to expiration (required for data fetching)
- `--min_strike`: Minimum strike price to fetch (required for data fetching)
- `--max_strike`: Maximum strike price to fetch (required for data fetching)
- `--symbol`: Underlying symbol (optional, default: SPX)
- `--db_path`: Path to SQLite database file (optional, default: database/option_chain_data.db)
- `--clear_db`: Clear all data from the database and exit (optional flag)

Examples

Fetch 0-7 DTE options with strikes between 5800 and 6000

```
python option_chain_fetcher.py --max_dte 7 --min_strike 5800 --max_strike
6000
```

Fetch 0-30 DTE options with strikes between 5500 and 6200, custom database

```
python option_chain_fetcher.py \
    --max_dte 30 \
    --min_strike 5500 \
    --max_strike 6200 \
    --db_path my_option_data.db
```

Fetch options for a different symbol

```
python option_chain_fetcher.py \
    --max_dte 14 \
    --min_strike 500 \
    --max_strike 550 \
    --symbol SPY
```

Clear the database

```
# Clear all data from the default database
python option_chain_fetcher.py --clear_db

# Clear all data from a specific database
python option_chain_fetcher.py --clear_db --db_path my_option_data.db
```

SPX/SPXW Symbol Handling

Automatic Combined Fetching

When you specify `--symbol SPX` or `--symbol SPXW`, the script automatically fetches **BOTH**:

- **SPX**: Monthly and quarterly expirations (3rd Friday of the month)
- **SPXW**: Weekly expirations (every Monday, Wednesday, Friday)

This ensures you get complete market coverage without needing to run the script twice.

Example

```
# This fetches BOTH SPX and SPXW chains
python option_chain_fetcher.py --max_dte 7 --min_strike 5800 --max_strike
6000 --symbol SPX

# This ALSO fetches BOTH SPX and SPXW chains
```

```
python option_chain_fetcher.py --max_dte 7 --min_strike 5800 --max_strike
6000 --symbol SPXW
```

Identifying Symbol Types in Results

You can identify which symbol type an option belongs to by examining the option symbol in the database:

- SPX options: Symbol will be like `SPX250117C5900` (no 'W')
- SPXW options: Symbol will be like `SPXW250110C5900` (contains 'W')

Database Schema

The script creates a SQLite database with the following schema:

Table: `option_chain_data`

Column	Type	Description
id	INTEGER	Primary key (auto-increment)
fetch_timestamp	TEXT	ISO timestamp when data was fetched
symbol	TEXT	Option symbol (e.g., SPXW250107C5950)
expiration_date	TEXT	Option expiration date (ISO format)
strike_price	REAL	Strike price
option_type	TEXT	'CALL' or 'PUT'
bid_price	REAL	Current bid price
ask_price	REAL	Current ask price
mid_price	REAL	Calculated mid price (bid + ask) / 2
delta	REAL	Option delta
gamma	REAL	Option gamma
theta	REAL	Option theta
vega	REAL	Option vega
rho	REAL	Option rho
days_to_expiration	INTEGER	Days until expiration

Table: `underlying_prices`

Column	Type	Description
id	INTEGER	Primary key (auto-increment)
fetch_timestamp	TEXT	ISO timestamp when data was fetched

Column	Type	Description
symbol	TEXT	Underlying symbol (e.g., SPX)
price	REAL	Mid price of underlying
bid_price	REAL	Bid price of underlying
ask_price	REAL	Ask price of underlying

Note: For SPX/SPXW options, the underlying price is always recorded as SPX (not SPXW), as both option types reference the same underlying index.

Indexes

- `idx_fetch_timestamp`: For querying option data by fetch time
- `idx_expiration_date`: For querying option data by expiration date
- `idx_underlying_fetch_timestamp`: For querying underlying prices by fetch time

Querying the Database

You can query the database using any SQLite client or Python:

```
import sqlite3

conn = sqlite3.connect('option_chain_data.db')
cursor = conn.cursor()

# Get all data from latest fetch
cursor.execute("""
    SELECT * FROM option_chain_data
    WHERE fetch_timestamp = (
        SELECT MAX(fetch_timestamp) FROM option_chain_data
    )
""")

# Get all calls with delta between 0.25 and 0.35
cursor.execute("""
    SELECT symbol, strike_price, delta, bid_price, ask_price
    FROM option_chain_data
    WHERE option_type = 'CALL'
    AND delta BETWEEN 0.25 AND 0.35
    ORDER BY expiration_date, strike_price
""")

conn.close()
```

Logging

The script logs to both:

- Console (stdout)
- File: `option_chain_fetcher.log`

Log levels include INFO, WARNING, and ERROR messages for monitoring the fetch process.

Note: The log file is automatically cleared at the start of each run to prevent unbounded growth. Each run creates a fresh log file.

Notes

- The script fetches data for all expirations from 0 up to `max_dte` days
- Both CALL and PUT options are fetched for each strike in the range
- The `fetch_timestamp` field allows you to track when each dataset was collected
- Unique constraint on (fetch_timestamp, symbol) prevents duplicate entries
- Market data streamer needs a few seconds to initialize before fetching data