Real Time Trade Sim - Documentation

Overview

This Trade Simulator project is a high-performance system designed to simulate cryptocurrency trades using real-time market data from the OKX exchange. It estimates transaction costs including slippage, fees, and market impact using advanced models, processing live Level 2 (L2) orderbook data via WebSocket streams.

Architecture

The system is divided into modular components:

- **UI (Tkinter-based)**: Provides a user interface with input parameters on the left panel and processed output on the right panel. Inputs include Exchange, Asset, Order Type, Quantity, Fee Tier, and Volatility. Outputs include expected slippage, fees, market impact, net cost, maker/taker proportion, and latency.
- **WebSocket Client**: Connects asynchronously to OKX L2 orderbook WebSocket endpoints to receive real-time bid/ask data streams. It handles reconnection, error logging, and message parsing.
- **Data Processing & Models**: Processes incoming orderbook ticks to calculate:
  - **Slippage** using a regression model approximating price movement impact.
  - **Fees** using a rule-based fee calculation derived from fee tier and trade size.
  - **Market Impact** using the Almgren-Chriss model, a quantitative finance model estimating cost due to liquidity consumption.
  - **Maker/Taker Proportion** with a logistic regression predictor (simplified for demonstration).
  - **Latency** measuring system processing time per tick.
- **Logging and Error Handling**: A logger utility ensures clear, timestamped log messages for easier debugging and monitoring.

Key Modules

1. `main.py`

- Starts the WebSocket client in an asynchronous loop within a background thread.

- Defines `process_data` to receive incoming data, compute cost metrics, and print results.
- Manages concurrency to keep the UI responsive and data processing efficient.

2. `simulator_ui.py`

- Implements a Tkinter GUI with input forms and an output console.
- Runs the WebSocket client asynchronously, updating the output panel with processed results.

3. `websocket/l2_orderbook_client.py`

- Connects to `wss://ws.gomarket-cpp.goquant.io/ws/l2-orderbook/okx/{symbol}` WebSocket endpoint.
- Receives Level 2 orderbook updates (asks and bids).
- Parses JSON messages and triggers callbacks for data processing.

4. Models (`models/` folder)

- **Fee Model** (`fee_model.py`): Calculates fees based on quantity and fee tier.
- **Market Impact Model** (`market_impact.py`): Implements Almgren-Chriss model for impact cost.
- **Slippage Model** (`slippage_model.py`): Uses regression to estimate slippage.
- **Maker/Taker Model** (`maker_taker_model.py`): Predicts maker/taker trade ratio.

5. `utils/logger.py`

- Provides a consistent logging setup with configurable verbosity and formatting.

Algorithms and Models

Almgren-Chriss Market Impact Model

- Calculates temporary and permanent market impact costs when executing large orders.
- Takes into account volatility, average daily volume, and execution speed.
- Formula adapted for cryptocurrency spot markets.

Slippage Regression Model

- Trained or assumed linear/quartile regression on historical orderbook data.
- Estimates the price impact relative to order size.

Fee Model

- Simple percentage-based calculation derived from exchange fee tiers.

Maker/Taker Proportion

- Logistic regression estimating the ratio based on market activity and order flow.

Performance & Optimization

- **Latency Benchmarking**: Measures data processing time, UI update latency, and end-to-end loop duration.
- **Memory Management**: Uses efficient data structures for orderbook data.
- **Network Communication**: Employs asynchronous WebSocket connections to avoid blocking.
- **Thread Management**: Runs network I/O on separate threads, keeping UI thread free.
- **Regression Efficiency**: Preloads and caches regression parameters.

Setup Instructions

Prerequisites

- Python 3.9+
- `websockets`, `tkinter`, `numpy`, `scikit-learn` (for regression models), `asyncio`

Installation

1. Clone the repository:

```
git clone https://github.com/yourusername/trade-simulator.git
cd trade-simulator
```

2. Install dependencies:

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

3. Run the simulator:

```
python main.py
```

Usage

- Select the exchange (OKX) and the trading pair (default BTC-USDT-SWAP).
- Enter quantity in USD, select order type (market).
- The system connects to real-time L2 orderbook and outputs the cost estimates continuously.
- View expected slippage, fees, market impact, net cost, maker/taker ratio, and latency in real-time.

Future Enhancements

- Support more exchanges and assets dynamically.
- Add historical data playback mode.
- Incorporate advanced machine learning models for more accurate predictions.
- Develop a fully interactive UI with charting and order simulation.

References

- Almgren, R. and Chriss, N., "Optimal execution of portfolio transactions," *Journal of Risk*, 2000.
- OKX API Documentation: https://www.okx.com/docs/
- Regression and Logistic Regression techniques (scikit-learn documentation).
- Real-time WebSocket data processing best practices.

Output

```
PS C:\idk\Codes\real-time-trade-sim> python main.py
Trade Simulator started. Press Ctrl+C to stop.
[2025-05-16 01:00:44] INFO - Connected to wss://ws.gomarket-cpp.goquant.io/ws/l2-orderbook/okx/BTC-USDT-SWAP
Symbol: BTC-USDT-SWAP
Exchange: OKX
Order Type: market
Quantity (USD): 100.0
Expected Slippage: $0.10
Expected Fees: $0.10
Expected Market Impact: $0.00
Net Cost: $0.20
Maker/Taker Proportion: Maker: 40%, Taker: 60%
Tick Timestamp: 2025-05-15T19:30:44Z
--------------------------------------
```
