Berkeley Option Database Modernization

15th October 2024

OVERVIEW

The Berkeley Option Database (BODB) contains option data from before 1996. This data is stored in long files of fixed length strings, which is not easily queryable or readable without significant computational effort. The goal of this project is to extract the data from these BODB files and convert them to a modern format, stored in a modern database which will allow interested parties to query for their purposes.

This project will be worked on as part of the Computer Science 4470Y capstone course for software engineering. The length of this course is one year, and the students working on this project also have a full course load, there are time limitations.

Because of the time constraints on this project, an iterative approach can be taken to enable flexibility and future modification. This project will be split into three main phases: data extraction, data cleaning/extension, and web service. The data extraction phase will be the main focus since it is the required backbone for the additional phases.

Each phase will be defined by a list of requirements and features which can be iterated upon as time allows.

MILESTONES

BODB Data Extraction

The main goal of the first stage is to extract BODB data into a modern database and transform it into a queryable and readable form. This will allow the data to be further manipulated, cleaned, and extended by this project, or other projects in the future. The quality of the extracted data will be a top priority as it will be the base of all further features.

The encoded strings must be decoded. The data format can be found within the bodb_guide.pdf file. Challenges will include decoding the ticker symbols, decoding the date, interpreting the expiry date given the month, determining order type, and decoding the price fields.

Data Manipulation, Cleaning, and Extension

The second goal is to analyze the data, and clean it of imperfections. This will require some knowledge about financial modeling. There are some well known mistakes and inaccuracies within the BODB. For example, some records may have an incorrectly encoded price for the underlying asset. Identifying these issues, and developing strategies to clean or correct them will make up a large portion of the work for this phase.

Once the data is cleaned, it can be extended to fit the existing OptionMetrics data, such as adding “end of day” data for price and volume. Access to OptionMetrics can be requested through Western Libraries.

Hosted Web Service

The stretch goal for this project would be to host the cleaned and extended data. The wider internet would have the ability to query the data. More details about this implementation will need to be discovered and ironed out. However, given the time constraints of this project, the details for this milestone can be worked on as progress is made.

REQUIREMENTS

BODB Data Extraction

1. The BODB date should be converted to yyyy-mm-dd SQL date format
2. The BODB time should be converted to HH:MI:SS SQL time format
3. The BODB expiration month should be converted to yyyy-mm-dd SQL date format
4. Using the Appendix provided in bodb_guide.pdf, convert the alternate ticker symbols to the original ticker symbol.
5. Extract the order type and encode it in a queryable format
    a. This includes identifying between put and call orders
6. Extract the prices and convert it into a readable number with a decimal
7. Volume needs to be extracted and recorded (if applicable)

Technical Details

Testing: A high testing standard will be maintained to ensure the validity of the data being extracted. We will aim for 100% branch and statement coverage. Test cases will be selected to cover known cases, and more may be added as issues are discovered.

Multiple Environments: “QA” and “Production”. Nomaan will take on the responsibility of managing and setting up these environments.

Language: Python 3. Chosen due to its popularity with researchers, which will make our work more easily extensible and modifiable in the future by others.

Python has its own idiomatic code style. It will be important to follow this style to maintain consistency throughout the project. The following provides a great overview on how to achieve this: https://intermediate-and-advanced-software-carpentry.readthedocs.io/en/latest/idiomatic-python.html.

Source Control: Github will be used to store the code for this project. The link to the repository on Github can be found here: https://github.com/riversheher/BODB-Extractor. The main branch will be protected, which means code cannot be committed directly to the main branch. Work will be completed on new branches. Pull requests will be created when the development of a new feature is complete, and an approval will be required from another developer. To approve a pull request, the change must satisfy the testing requirements outlined in this document.

Documentation: Documenting the code is an important part of this project's modifiability and extensibility. This will allow current and future developers to understand the code that is written, debug, and change it to suit their needs. We will follow pydoc standards, and comment each method definition with a multi-line docstring that explains the purpose, and interface. More can be found here :https://www.datacamp.com/tutorial/docstrings-python

Other Points: Use of a virtual environment is encouraged to maintain the consistency of environment across systems: https://docs.python.org/3/library/venv.html.

Extractor Components

extractor.py

This is where the main() method will live. This script will be the orchestrator which makes sure all extraction steps are followed in sequence. The resulting decoded information from the secondary scripts will then be used to create an object depending on the type of order: quote_record, or trade_record, with the subtypes call_record and put_record.

Possible Issues: More data types may be required as the BODB includes many more types of records. However the vast majority of records will be "quotes" or "trades". This will need to be investigated further, since the majority of the work will be done for quotes and trades.

How these records will be stored into a database will also need to be considered. The database structure may end up changing through the course of this project depending on issues we find, or considerations we create.

file_reader.py

The file_reader.py python file will contain the code which will extract each line within the BODB file provided. This file will also contain any extra methods written for the purpose of reading the BODB files.

Input: File path for the BODB resource file such as "./res17"

Output: A list of strings, where each string is one line from the provided BODB file.

This script will accept a file path as input, open the file, read the file, and extract each line into a string, which will be stored in a list and returned.

Possible issues: A file may be too big, and may need to be split in order to maintain the performance and resource constraints of the program. This may need to be investigated further.

date_time.py

This file contains the methods used to convert the date and time encoded in the BODB into a modern format that can be easily inserted into an SQL database, and easily read.

Input: Substring [6-17] from the encoded BODB line.

Output: An object representing the date/time, easily readable and insertable into SQL databases. (potentially a date/time object, or string object)

expiration_month.py

This file contains the methods used to convert the expiration month encoded in the BODB into a specific date that can be easily inserted into an SQL database and easily read.

Input: Substring [18-19] from the encoded BODB line.

Output: Date of expiration, year, month, and day.

Information about how to calculate this can be found in the bodb_guide.pdf file.

ticker_conversion.py

The ticker symbols on the BODB may be concatenated. For example IBM may be encoded as IBL for a leap (just an example). These alternate symbols only exist for the purpose of encoding in the BODB since it was limited to 3 characters. This will need to be converted back to the original ticker.

Input: Substring [3-5] from the encoded BODB line.

Output: Original ticker, and its description (ex. full company name).

record_type.py

This will determine the type of record being encoded.

Input: Substring [1-2] from the encoded BODB line.

Output: quote, trade, or another record type (find encoding in bodb_guide.pdf appendix.

The main focus should be quotes and trades. The other record types can be discarded initially, and worked on after the quotes and trades if they are hard to manage.

trade_detail.py

This determines the trade price, volume, strike price, and put/call order type for trade records.

Input: record object, and Substring [20-35]

Output: trade_record with decoded trade price, volume.

The *trade_record* object encodes call or put trades. The *record* object will be passed in with fields that pertain to date, time, ticker, expiration month, and will be used to construct a child trade_record object which also contains the price and volume details of the trade.

Technical details will be ironed out in diagrams to come.

quote_detail.py

This determines the ask/bid prices for quote records, and any additional details regarding it.

Input: record object, and Substring [20-35]

Output: quote_record with decoded bid/ask price fields decoded.

asset_price.py

This determines the price of an asset based on the encoded BODB line.

Input Substring [36-40]

Output: float with the price decoded and decimal point added to the correct location.

Data Types

record.py

+datetime

+ticker:string

+expiration_date

+strike_price:float

+underlying_asset_price:float

quote_record.py

+bid_price:float

+ask_price:float

trade_record.py

+type:enum (call or put)

+trade_price:float

+volume:integer