Università della Svizzera italiana

Prof. P. Gruber
P. Montemurro
2021-06-14

Programming for Finance II
Riccardo Cosenza, Jan Gobeli , Silvia Magni & Ivana Pallister

Algorithmic Trading
Assignment 2

Hand in before:
14 June, 2021
23:00 pm

1 Introduction

The main aim of this project is to establish a connection with the Interactive Brokers platform and implement a trading strategy which performs trades using tick data. In order to add to the project the possibility for the user to customize the trading strategy, we created two separate Python files, one which contains the backbone structure of the program and another in which we find the actual trading strategy, which can be modified. We nonetheless proposed a personal trading strategy based on a simple market making algorithm that waits on the bid and corrects its current level to a certain spread.

The minimum requirements of this project were indeed to:

- Implement a simple trading strategy in a robust way;
- involve in the strategy at least limit orders;
- verify everything twice, e.g. execution of limit orders, buying power before buying something, short-ability of a stock before shorting it etc.
- bonus: implement some sort of reporting and/or back-testing.

2 User Guide

2.1 Installation and Usage - README.md:

The dependencies for this project are (can be found in *requirements.txt*):

- *ib_insync:* a framework that simplifies the Interactive Brokers (IB) Native Python API, the interface which allows to trade algorithmically with IB. In particular, it offers a more familiar environment for Python programmers, an easier installation process, methods and syntax[1].
- *nest_asyncio:* asyncio is a library that provides infrastructure for writing concurrent code using the async/await syntax[2]. next-asyncio is a module which 'fixes' an intrinsic problem of asyncio, allowing nested use of the asyncio.run and loop.run_until_complete commands[3].
- *numpy:* a python open source library which adds support for multi-dimensional arrays and matrix data structure. It provides a huge collection of mathematical functions which can be used to operate on n-dimensional arrays[4].

[1] https://algotrading101.com/learn/ib_insync-interactive-brokers-api-guide/
[2] https://algotrading101.com/learn/ib_insync-interactive-brokers-api-guide/
[3] https://pypi.org/project/nest-asyncio/
[4] https://en.wikipedia.org/wiki/NumPy

- *tableprint* allows you to easily print tables of data.

The user can simply install all of these packages by running:

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

Some steps must be performed in order to run the IB_trader program:

1. The user needs to open a paper account on Interactive Brokers.
2. Download and install the IB Trader Workstation (TWS) and let it run in the background. The IB TWS is the platform which allows traders, investors and institutions to trade different instruments, ranging from stocks and options to futures and forex to funds and obligations, over more than a hundred markets around the world from a single account[5].
3. The user must now design a trading strategy by downloading data in the tests.ipynb file and then translate it into a fully fledged strategy.py file.
4. We provided an example of such a strategy in the test_strategy.py file: this is a simple market making algorithm that waits on the bid and corrects its current level to a certain spread.
5. Replace the standard algorithm with whichever strategy the user can imagine.
6. The user can finally see the profits from his or her trading!

3 Code Analysis

Our program is built over two Python files: *test_strategy.py* and *connect.py*. The *test_strategy.py* file contains what happens to incoming tick-data and the trading decisions made, while the *connect.py* file contains the functionalities to connect to IB and the implementations of sending orders and updating them. In addition a Jupyter Notebook file called *tests.ipynb* serves as space for code testing and strategy creation.

3.1 test_strategy.py

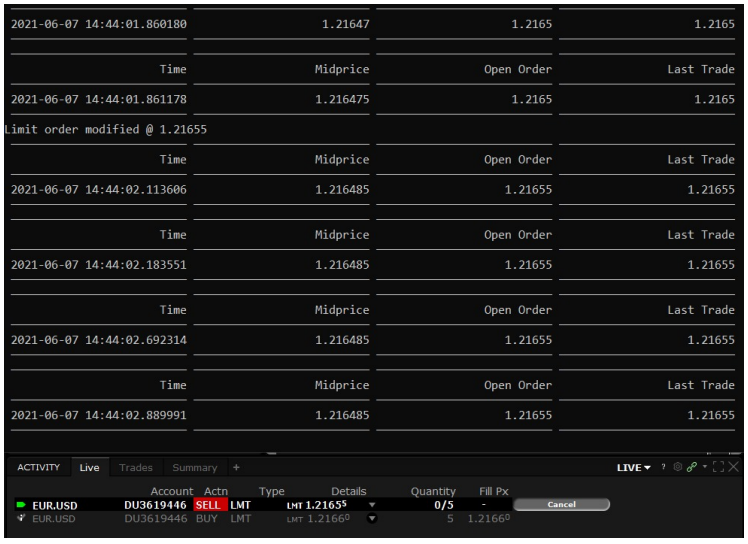

Figure 1: terminal output

In this file we provide an example of trading strategy implementation, which can be modified by the user. First of all, we need to import four modules: numpy to operate with n-dimensional arrays, *connect.py* which is the second file that will be used in order to create the actual connection with Interactive Brokers (recall IB must be running in the background), datetime to get the current time and tableprint to output tables in the terminal.
We define a number of parameters: amount as the quantity that will be traded on each signal, ticker as the trading ticker, last_buy, in_long as a flag to check if we are currently in a position, loopback as the amount of data

[5] https://www.interactivebrokers.eu/it/index.php?f=15900

needed for the trade logic, spread and trade_open. As anticipated, a connection with IB is created and opened. The ib_connect class, together with the open_ and close_connection definitions, is defined in the *connect.py* file.
Also at the beginning of the file, two print functions are defined to help the user better understand what the code is doing. The first one prints out a table containing the value received by IB along with a timestamp. The second is a print statement that notifies the user about a trade that has been submitted. Next we find three important functions defines:

- trade_logic(): the main function which performs the trading strategy itself. This function is called anytime a new tick arrives in order to check the trading condition. If, according to the trading condition, the decision is to trade, then the trades are directly executed using the functions defined in the *connect.py* file. """ In particular, we first check if a position is open: if yes, the position (whether long or short) is 'updated', while if we are currently not in a trade a new position must be created but only after checking the trading ability based on the account balance (through the balance_check function defined in the connect.py file). """
- process_ticks(tick): the function that processes incoming ticks and saves them to a local data-frame. To prevent the length of the data-frame from getting too big, it is then cut so that it will only contain the n most recent ticks necessary for the trading strategy.
- new_tick(tickers): the main loopback function which is automatically executed by IB whenever a new tick arrives. The tick is then added to the local data-frame using the process_ticks command and finally the trading logic is executed.

At the very end of this file we find the command with which the whole program starts running, that is the ib.start_stream(). This command links back to the other .py file which will be analyzed shortly.

3.2 connect.py

In this file we find the connection class which serves as the backbone of the trading strategy, as it contains the functionalities that a strategy needs to be executed.
The nest_asyncio module is loaded at the beginning of the code.
Next, the ib_connect class is defined, taking as initial parameters for the IB Connection the localhost, the port IB uses, and the client id. In addition a flag checks if there are open orders and another one takes the ticker of the strategy file. Just below this class we find the definitions which serve to open and close the connection to IB.
The start_stream function, which connects to IB and starts the stream of incoming tick data. It takes three parameters: an asset (forex or stock), a ticker (the ticker of the asset that will be traded) and a loopback function (which is used to process incoming ticks). Main testing has been performed on Forex pairs as tick data is freely available and the markets are open for longer times (24/5). If there is an event, an incoming tick, then the loopback function is run together with the new_tick(), and the tick is processed through the process_ticks function. The order_filled function is a routine to report trades' execution.
With the next two functions different kinds of orders, market and limit, are created:

- The create_marketorder function creates a market order and returns the order id or 'false' if not enough money is available. It takes two parameters: action (buy or sell) and amount (the amount of shares to trade). Hence, in this function we find a crucial verification, the one for buying power.
- The create_limitorder similarly creates a limit order and saves the order id. It takes the same inputs as the create_marketorder function, plus the limit parameter, which defines the limit price

at which the trade will be entered. Likewise the last function we defined also here we find a check for the balance on the account.

After these two, we find a list of functions defining actions concerning orders.

- modify_limit_order;
- cancel_order;
- check_order_status: if the order is still open, the program will wait for 5 seconds and check again if it is executed;
- balance_check: this function checks if there is enough money left in the account to execute the current trade. It takes the parameter amount, that is the number of shares that will be traded. It returns a boolean value: 'true' is enough money is available, 'false' otherwise;
- available_balance: it fetches the current available balance on the account;
- open_orders;
- sleep: it waits for 'time' (parameter) seconds while everything keeps processing in the background.

3.3 tests.ipynb

In this file various functions and commands are tested. For example, we find the commands used to establish the connection with IB, some line of code to download the last ticks and create a data-frame. Here the user can build and test his personal strategy which he can then implement in the *test_strategy.py* file. A simple backtesting function is already included in the file to let users backtest their ideas and approaches and see if they managed to turn a profit. This can then be analyzed using simple print functions or packages such as ffn.