Algo Trading on Crypto Market

Report - Project Code **AL**
CQF Dalegate - Kaifan Wang
Date - 08/15/2024

Introduction

This project aims to design algorithmic trading strategies to trade on the cryptocurrency market. Strategies are back-tested using python backtesting library, and implemented on real-time broker API. Historical data is obtained from ObenBB datasource. Both Trend-following and Mean-reversion trading strategies are implemented and backtested. Parameters are optimized for max profits during backtesting. At the end, the trend-following strategy is executed on a paper trading account with Alpaca API for practical experiment.

Design Details

Trend-following:

Trading Target: **Bitcoin and Solana**
Historical Data: **Past 3 months of price data sampled in 15-minutes interval.**
Backtest interval: **15 mins**

Trading indicators used to generate signals:

- **Exponential Moving Average(EMA)**
- **Moving Average Convergence/Divergence (MACD)**
- **Average Directional Index (ADX)**

The crypto market has moved up a lot in the past 3 months, and the nature of the crypto market is super volatile, hence it was quite difficult for me to find a high-frequency trading strategy that generates much higher profit than the simple buy-and-hold. I tried various signal-generation approaches during my backtesting, and here I found a strategy that generates reasonable profits for both bitcoin and solana, of which the profit surpasses the simple buy-and-hold:

Buy signal:

MACD > 0, this means12-day ema is greater than 26-day ema, demonstrating upward trend

MACD crossover above MACD-signal line, this demonstrates upward momentum

ADX > ADX threshold, ADX demonstrates the strength of a trend(irrelevant of direction), the higher the number, the stronger the trend is

All 3 conditions has to be true to generate a buy-signal

Closing (long) position signal:

MACD < 0, when MACD drops back below 0, the upward trend is lost

Price < 0.99 * EMA, when price drops 1% more below the moving average, the upward trend is lost

Both conditions has to be true to generate a closing signal

Sell signal:

MACD < 0, this means12-day ema is less than 26-day ema, demonstrating downward trend

MACD crossover below MACD-signal line, this demonstrates downward momentum

ADX > ADX threshold, ADX demonstrates the strength of a trend (irrelevant of direction), the higher the number, the stronger the trend is

All 3 conditions has to be true to generate a sell-signal

Closing (short) position signal:

MACD > 0, when MACD moves above 0, the downward trend is lost

Price > 1.01 * EMA, when price moves 1% more above the moving average , the downtrend is lost

Both conditions has to be true to generate a closing signal

Mathematical Description

ADX (Average Direction Index): measures the strength of the trend. Below explains my calculation in the code.

- True Range (TR) is calculated and smoothed (ATR) with a 14-period average.
  - Captures the extent of price movement, used to normalize DI later.

- Positive Directional Movement (+DM) = Current High - Previous High
- Negative Directional Movement(-DM) = Previous Low - Current Low
  - Record +DM only when +DM > -DM and +DM > 0
  - Record -DM only when -DM > +DM and -DM >)
  - Intuition: This tries to capture if increase of price dominates the decrease of price or vice versa, in other words, is buyer/seller dominating the market within this period.

- Positive Directional Movement Index (+DI): smoothed +DM / ATR (normalized by ATR)
- Negative Directional Movement Index (-DI) : smoothed -DM / ATR (normalized by ATR)

    - Normalized by ATR so the number lies between 0 and 100% of ATR

- Directional Index (DX): |+DI - -DI| / (+DI + -DI)
    - Calculate the strength of the positive/negative directional index, normalized by the sum as percentage, should have a number between 0 and 100%

- Average Directional Index(ADX):
    - Smooth the 14-periods average of DX

Caveat :When smoothing (with 14-periods average), under a formal ADX definition, the first smoothed value should be the sum of the first 14 periods, and starting at 15th value, smoothed value = previous smoothed value - 1/14 * previous smoothed value + current value. However, implementing such calculation is quite cumbersome in code, hence, an exponential moving average (pd.ewm) with alpha = 1/14 is used. Such techniques might result in the first few values of ADX being off from the true value, but after a short period of time, the ADX converges to the true value. I tested my value against the ADX indicator function pulled directly from the python talib library, and verified the correctness of my calculation and its convergence shortly after about 50 periods.

Parameters Optimization:

Due to the nature of highly-volatility in crypto market, I found a few parameters that would have a significant impact on P&L, those need to be optimized during backtesting :

- **ADX_threshold**: usually, ADX above 25 would demonstrate a strong trend. This number is optimized between the range of **25-40** to account for the highly volatile crypto market.
- **Trade_size**: 3 choices, **33%, 49%, and 99%**. Those choices are deliberate. For 33% and 49%, this means you could have more than one chance to enter the market with consecutive same-direction signals. For example, if a buy-signal is triggered 3 times in a row, you could split the capital and buy at 3 different times with 33% trade size each. However, with 99% capital invested for the first buy-signal, subsequent same-direction signals will not trigger action again due to the lack of capital. The reason for 99% instead of 100% is because 100% will trigger the library to buy only 1 share of stock instead of the intended 100% percentage of capital.
- **Take Profit(tp) ratio**: TP and SL has a considerable impact for high frequency trading. The choices are **(0.04, 0.05, 0.06, 0.1)**. Let's take 0.04 as an example, during a long position, the position will be closed when price moves 4% above the trade price, during a short position, the position will be closed when price drops 4% below the trade price. 0.1 would be close to no TP threshold, as it is hard for price to move 10% within such a short period of time, because of the high frequency of trading.
- **Stop Loss(sl) ratio:** Same as above, choices are **(0.01, 0.02, 0.03, 0.1)**, where 0.1 would be close to no SL threshold.

Results will be discussed in the next section.

Mean-reversion:

In order to generate profit with a mean-revision strategy. The chosen time series has to be stationary. However, It is quite difficult to find stationary equities on the market, and I do not want to simply assume stationarity for a random ticker, which would likely result in a meaningless trading strategy. Hence I implemented a simple cointegration pair trading strategy. I collected historical daily price data for both Bitcoin (BTC) and Etherium (ETH) from the past 2 years, and I ran engle-granger procedures on them: I ran regression on one another, and I ran the ADF test on the residuals. The resulting p-value was below 1%, which rejected the null hypothesis. This indicates the spread(residuals) between BTC and ETH daily price in the past 2 years are stationary. Next, I fitted the residuals to the OU process and calculated the residual mean and sigma_equalibrium. Ideally, for Cointegration pair trading, error correction model should be incorporated to check for significance level, and ADF tests should be self-implemented instead of using a ready-library. However, pair trading techniques are not the main focus of this AL project, hence I skipped this part and took a simplified approach.

This engle-granger procedure is coded in the “engle-granger.ipynb” file, and it gives me the following result:

1. Fitting the data to linear regression

$$ETH = const + \beta * BTC + \epsilon$$

hedge constant $const = 544.50$**, and hedge ratio** $\beta = 0.04346$

2. The ADF test on the residuals shows a p-value of 0.**000095** which is below 1%, rejecting the null hypothesis and indicating stationary spread.

3. Fitting the residual to the OU model

$$e_t = C + Be_{t-1} + \epsilon_{t,\tau}$$

I was able to obtained the **average number of days to revert** $half\_life = 13.82\, days$**, residual mean** $\mu = -9.91$ **and standard deviation** $\sigma_{eq} = 134.79$

Those numbers are recorded and directly used in my backtest.

For backtesting my mean-reversion strategy, theoretically, I should be allocating $[+100\%\, ETH,\ -\beta\%\, BTC]$ when residual is low, and $[-100\%\, ETH,\ +\beta\%\, BTC]$ when residual is high for optimal results. However, trading more than 1 ticker on the backtesting framework I chose needs significant additional work . To save some coding hassle and time, I took a workaround and just ran backtesting on the trading of “ETH-BTC” index. I realize this is not optimal according to the pair-trading model, but this should demonstrate a similar idea. Again, pair-trading techniques are not the main focus of this project.

For backtest, “ETH-BTC” daily prices from past 2 years were collected for trading purposes, “BTC” and “ETH” individual prices were collected and appended to the dataframe for signal-generation purposes. Residuals were calculated using the above hedge ratio and hedging constants obtained from the engle-granger procedure. Trade signals are generated based on mean-reversion of the residuals.

Trading Target: **ETH-BTC Index**
Historical Data: **Past 2 years of price data for ETH-BTC, ETH, BTC, sampled in 1-day interval.**
Backtest interval: **1-day**

Buy signal: Residual > residual_mean + **Z_entry** * sigma_eq
Sell signal: Residual < residual_mean - **Z_entry** * sigma_eq
Closing signal: Residual bounces back to residual_mean (within a small bound **Z_exit** * sigma_eq)

Residual_mean and sigma_eq were obtained from the above engle-granger procedure**.**
Z_entry and Z_exit are parameters that are optimized for max P&L.

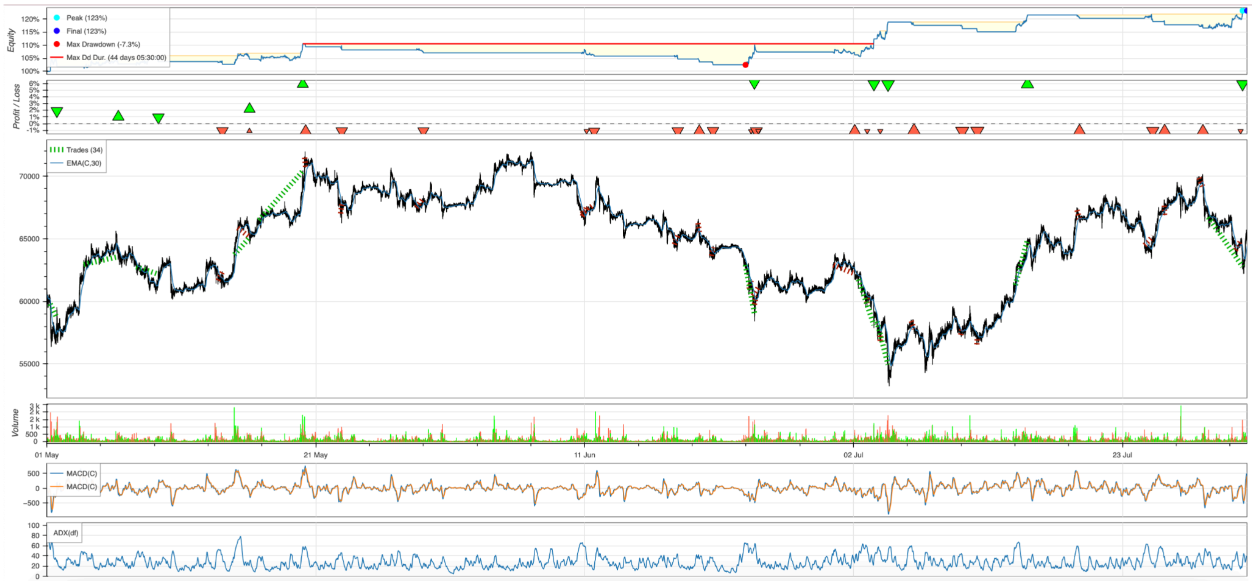
To conclude, although I took a rather simple pair trading strategy and skipped some model improvement techniques, my argument is that running a mean-reversion strategy on a stationary spread is at least much better than running it on a single arbitrary non-stationary ticker.

Numerical Techniques table

Self-Implemented	**Ready-Library used**
Trading strategies and signals generation	Augmented Dickey-Fuller Test For Mean Reversion
Various Stock indicators including EMA, MACD, and ADX	
Eagle-granger procedure and fitting the residuals to OU process and computation of sigma_eq	
Alpaca API integration and Historical data retrieval from OpenBB	

Results & Analysis

1. Trend-following strategy on BTC 15-minute price for the past 3 months

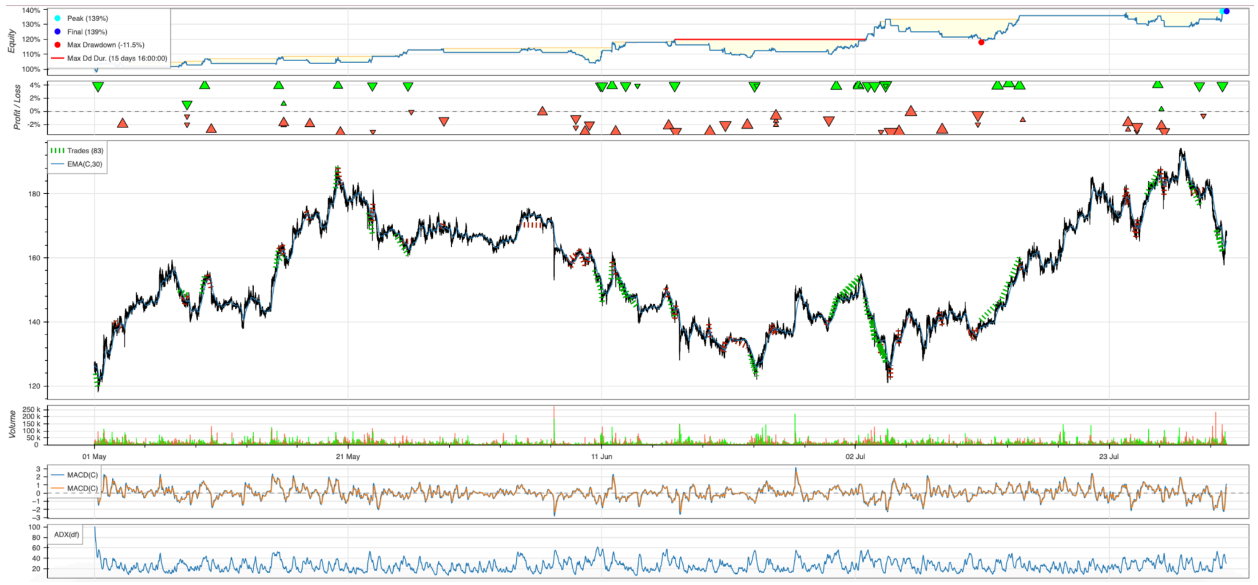


Backtest Summary:

TimeFrame	2024/05/01- 08/01 (3 month)
Exposure Time [%]	23.790323%
Return [%]	**23.250246%**
Buy & Hold Return [%]	**7.840641%**
Return (Ann.) [%]	111.728514%
Volatility (Ann.) [%]	59.85909%
Sharpe Ratio	**1.866525**
Max. Drawdown [%]	**-7.326063%**
# Trades	**34**

The result of my strategy on BTC for the past 3 months is significantly better than the buy-and-hold strategy. It generated a return of 23.2%, triple the 7.8% buy-and-hold strategy. The sharpe ratio is 1.86 with max drawdown of -7.3%. The optimized parameters are **adx_threshold=34, trade_size=0.99, tp_ratio=0.06, sl_ratio=0.01**

2. Trend-following strategy on SOL 15-minute price for the past 3 months

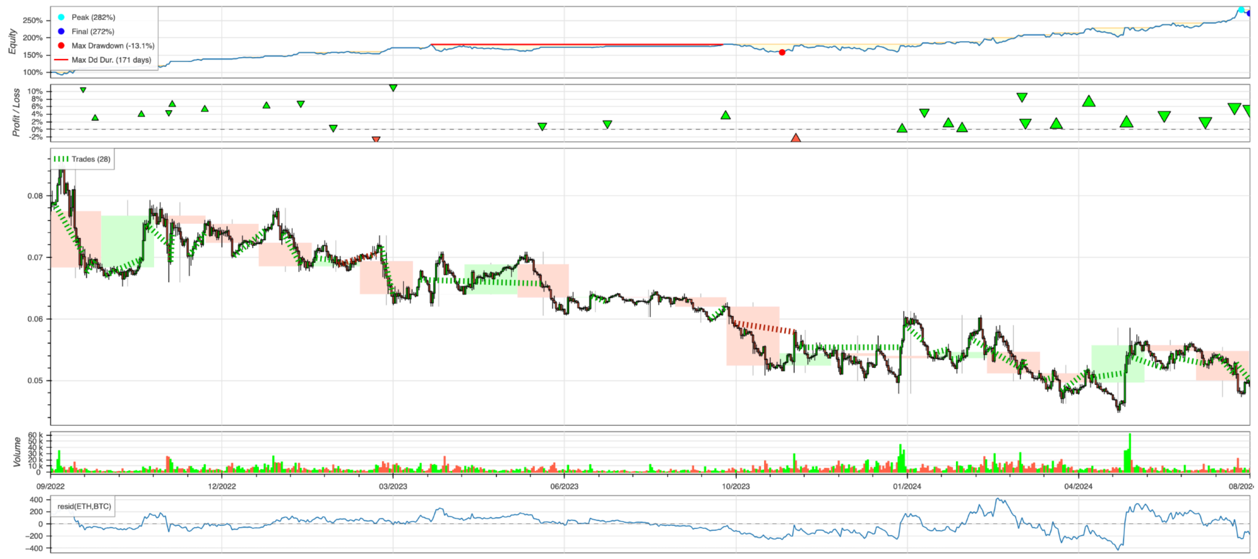


Backtest Summary:

TimeFrame	2024/05/01- 08/01 (3 month)
Exposure Time [%]	29.569892%
Return [%]	**38.749316%**
Buy & Hold Return [%]	**32.262912%**
Return (Ann.) [%]	211.502457%
Volatility (Ann.) [%]	137.262595%
Sharpe Ratio	**1.54086**
Max. Drawdown [%]	**-11.512828**
# Trades	**83**

The result of my strategy on SOL for the past 3 months is only slightly better than the buy-and-hold strategy. It generated a return of 38.7% for SOL, surpassing the 32.2% buy-and-hold strategy. The sharpe ratio is 1.54 with max drawdown of -11.5%. The optimized parameters are **adx_threshold=25, trade_size=0.99, tp_ratio=0.04, sl_ratio=0.03**

3. **Mean-reversion strategy on ETH-BTC daily price for the past 2 years**



Backtest Summary:

TimeFrame	2022/09/01- 2024/08/01 (2 years)
Exposure Time [%]	72.6%
Return [%]	**171.62366%**
Buy & Hold Return [%]	**-37.783983%**
Return (Ann.) [%]	68.252027%
Volatility (Ann.) [%]	48.265302%
Sharpe Ratio	**1.414101**
Max. Drawdown [%]	**-13.097329**
# Trades	**28**

The result of my pair trading mean-reversion strategy on the ETH-BTC index is surprisingly good. It generated a total return of 171.6% in the past 2 years, while holding the index would result in a -37.78% loss. The sharpe ratio is 1.54 with max drawdown of -11.5%. The optimized parameters are **Z_entry=0.6, Z_exit=0.28**

((To be able to play around with the graph. Please checkout the “/backtest_results” folder.
For more numerical information. Please check out the “strategy_backtest.ipynb” code.)

Broker API integration - Alpaca

Summary

Trading Target: **BTC/USD**
Trading Frequency: **1-minute level**
Trading Strategy: **Trend-following with MACD ADX and EMA (same strategy as above)**

To best mimic real-time live trading experience, I leveraged the **alpaca websocket functionality** to stream the most up-to-date crypto price of BTC, which is OHLCV data, in 1-minute intervals. Every minute my system will calculate the up-to-date indicators values and act on trading signals.

File 1: live_strategy.py
Entry point. This file maintains all logic for initializing the program, retrieving historical data, maintain running indicator values and defining trading strategy
File 2: trader.py
This file contains the trader class, managing logic regarding order execution with Alpaca API, verifying and tracking positions from the server.

Technical Details

Indicator values initialization and updating

In order to have indicator values initialized whenever the program starts, we need to have the most recent historical price data. When the program starts, the latest single minute price data is pulled and the timestamp is recorded. This timestamp acts as the most accurate “current timestamp”, and then the past 60 minutes (from current timestamp) historical data (1-minute interval) is pulled. The is used to calculate the initial values for the indicators.

The indicator values also need to be updated in real-time whenever my websockets receive new price information (every minute). My system stores the running values of indicators both in memory and in disk (check for file “runnning_indicators.csv”), the indicator values are updated every minute according to the new price data.

Event Handling and Incoming data validation

There is a chance of failing every time where Alpaca API is called. This includes retrieving historical data, handling messages from the socket, submitting orders, and checking for positions/orders. Python try-except block is used to handle those server errors. Input-validation is also implemented at various places to check for the sanity of incoming data from Alpaca. Error handling logic is specific to each error. Please check the code for more information. Errors are also logged and saved in the file “algo_trading.log”.

Position/Orders Tracking

Even if the order is submitted without any errors, there might be failures somewhere on the server which may cause the order not being filled successfully. Hence, I designed a custom algorithm to check for the order’s success on the server.

My system records all submitted orders in memory with order_id. After order execution (wait for 1 second), my system pulls the order from the server using order_id, and checks if the order is successfully filled with the correct amount. If not, a retry mechanism is triggered to submit the order again. This function can also be run at fixed intervals systematically to verify all past orders are successfully filled. Please check “trader.py -> validate_past_orders()” for more information.

Shorting Crypto in Alpaca

Unfortunately, Alpaca does not support shorting on cryptocurrency. This is difficult to deal with. However, I would still want to test my code for shorting trading signals. My workaround is that instead of shorting on BTC when needed, I would be buying ETH. There is no business logic behind this but simply trying to submit an order to Alpaca and open a position to test my code path for shorting. In other words, when an unit of ETH is bought in my account, it means I am trying to short BTC. Also, it means I can only have either BTC or ETH in my account at one time, not both. (Since I cannot be longing and short one equity at the same time). The code changes to accommodate this workaround is quite trivial, please see “trader.py” for more information. With this, I was able to test all my trading signals and code paths.

Running Instruction:

Part I Backtest:

1. Run retrieve_financial_data.ipynb to retrieve data. All needed data are stored in /data folders. Please setup your own polygon credentials in ~/opebb_platform/settings.json file (This step can be skipped as csv files are uploaded in my projects)
2. Run engle_granger.ipynb to test for stationarity between BTC and ETH daily price for the past 2 years. Obtaining numerical values for backtesting: Hedge_ratio, hedge_constant, mu, and sigma_eq.
3. Run strategy_backtest.ipynb ro run backtests on historical data and generates reports. Reports are generated in the /backtest_results folder

Part II Alpaca-Live-Trading

Since jupyter-notebook has a difficult time working with sockets and multiprocessing, codes are written in the .py file and run from terminals directly. Please install the required libraries and set up your own ALPACA_KEY and ALPACA_SECRET in your environment variables.
Put “trader.py” and “live_strategy.py” in the same folder and run the command **python live_strategy.py** in the terminal should start the program and set up the socket for receiving live data from alpaca.

1. Live market data is recorded at “live_bar_data.csv”
2. Running indicator values are recorded at “running_indicators.csv”
3. Logs are saved at “algo_trading.log”, this would include order history, error messages and some important logs during execution.

Conclusion

In this project, I designed both trend-following and mean-reversion strategies to algorithmic places trades in the cryptocurrency market. For trend-following strategy, 3 indicators ADX, MACD, and EMA are used to generate trading signals. All indicators and trading strategies are implemented from the first principle. For mean-reversion, a cointegrated relationship between ETH and BTC daily price from past 2 years was discovered. Eagle-granger procedure was used to test stationarity, and mean reversion strategy was traded on the spreads between ETH and BTC. For both strategies, parameters are optimized during backtesting for max profits. Plots, profits, and risks are analyzed. All strategies generate profits higher than the buy-and-hold strategy, which is already promising regarding the crypto market for the past 3 months. In the second part of the project, the trend-following strategy is implemented with Alpaca API using python. Price data are pulled at 1-minute intervals from alpaca and traders are placed at high frequency. Live events from Alpaca are processed with rigorous error handling and input validation checking. Orders and positions from the server are re-checked to ensure orders are filled successfully.

Future Improvements:

1. For mean-reversion strategy on BTC/ETH, error correction model should be incorporated to check for significance level for stationarity. The Kalman filter can be used for more accurate entry/exit points. Also, a more sophisticated backtest framework should be used with capitals allocating on the correct hedge ratio between BTC and ETH.
2. For trend-following strategy, other indicators such as RSI can be used to help with signal generation. Also, different time intervals should be tested (from 1 minute to 1 day) for optimal trading frequency.
3. For live trading, C++ or GoLang can be used instead of python for optimal speed. Postgres ' timescaled database should be used to save historical data and order data instead of writing everything to the disk with a csv file, which is not scalable and time-consuming.