ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

ANALYSIS OF CRYPTOCURRENCY MINIMUM VARIANCE PORTFOLIO

FIN-525 SEMESTER PROJECT

Ke Ma, Shufan Wang

27th January 2023

ABSTRACT

Cryptocurrency is an emerging type of asset, which is more volatile and risk-prone compared with traditional assets and stocks. In recent years, it has reached a trillion market capacity. It is appealing to investigate the MVP (Minimum Variance Portfolio) of cryptocurrency through different covariance cleaning strategies. In this project, we apply several well-studied filtering methods to clean the covariance matrix and retrieve the dependence of different crypto assets. We also apply a rolling window computation to calibrate the results. Our results show: 1) BAHC achieves the best performance of in-sample and out-of-sample volatility among all the methods. 2) Eigenvalue clipping and Linear shrinkage are sensitive to parameters and could lose important information. 3) Cryptocurrency is more volatile than traditional stocks, where big crashes can greatly influence the realized risks.

CHAPTER 1

MOTIVATION

1.1 INTRODUCTION

Cryptocurrency is a kind of high-volatility asset that was born since Bitcoin [1]. It represents a new type of currency that is minted strictly according to the transparent and non-modifiable code running in the blockchains. These cryptocurrencies have gained prominent attention because of the fast development of blockchain technology together with the flourishing of DeFi (Decentralized Finance). The past several years witnessed the explosive growth of cryptocurrency where some are well-designed and some are scams. Cryptocurrency still reaches a trillion capacity during the bear market nowadays where most high-quality assets survive.

As cryptocurrency naturally induces higher risk than traditional assets, it is appealing to investigate the MVP (Minimum Variance Portfolio) of cryptocurrencies. It is also unclear if the existing covariance cleaning approaches for stocks can apply to the crypto world. Besides, the MVP analysis will provide a reference to choose high-quality cryptocurrencies with minimum volatility in the bear market.

1.1.1 COVARIANCE MATRIX FILTERING

The covariance matrix is a square matrix, giving the covariance between each pair of elements of a given random vector. The covariance matrix of assets has been well-studied, which provides important information about the market price movement and reveals the correlation between different assets. This kind of matrix will become very noisy when the number of objects (i.e. assets) is similar to the number of features. In the financial market, since only the most recent history is likely to be relevant, this case appears very frequently. Thus, covariance matrix filtering is necessary to reduce estimation noise.

A popular approach is to obtain filtered covariance matrices from the corresponding correlation matrices. clustering, etc. Different approaches have been proposed to clean the covariance matrix. RMT (Random Matrix Theory) cleans the matrix with eigenvalues clipping. It regards the small eigenvalues as noise and only keeps the large eigenvalues as true relations. Another direction is Linear Shrinkage [2], which adjusts the overfitting covariance matrix by a linear combination with another underfitting estimator. There is also an approach to ansatz for the shape of the true correlation matrix. It imposes constraints on the structure of the eigenvectors and of the eigenvalues. One of the ansatzes is hierarchical clustering [3], an agglomerative algorithm that recursively clusters groups of objects according to a distance. However, in practice, it is hard to find statistically-validated hierarchical structures [4] when the fitted hierarchical structure is highly sensitive to small variations of data.

1.1.2 MINIMUM VARIANCE PORTFOLIOS

In risk management, a minimum variance portfolio is a collection of securities that combine to minimize the price volatility of the overall portfolio. The greater the volatility, the greater the fluctuations, and the higher the market risk. To minimize the risk, we should minimize the price ups and downs, for a greater chance of slow but steady returns over time. Ideally, objects with low correlation to each other should be chosen in the portfolio to minimize the variance. In the real world, investing in those that perform differently, compared to the market, is a good strategy with diversification.

1.2 AIM

In this project, we aim to find an MVP of cryptocurrencies with different approaches and analyze their in-sample risk and out-of-sample risk. We retrieved the hourly kline data of 394 USDT-based (USDT is a stablecoin pegged to the US Dollar) cryptocurrency prices from Binance since 2021. We adopted four strategies in the following.

- No covariance cleaning
- Eigenvalue clipping
- RIE (Rotationally Invariant Estimators)
- BAHC (Bootstrapped Average Hierarchical Clustering)

We compare the realized risks among all the strategies and analyze their advantages and disadvantages. Our code implementation could be found by this link: https://github.com/Uniblake/fin525/tree/master

CHAPTER 2

DATA SET

2.1 DATA RETRIEVING

We retrieve all the data in this project from Binance, which is the biggest centralized exchange of cryptocurrency all over the world. Binance already filtered most scams so that most assets (also called tokens in the following) we used are of high quality. Binance provides the K-line spot data in different granularity such as hourly, daily, and monthly basis. The K-line spot data contains the exchange rate of two tokens such as Ether to Bitcoin but not the traditional price in US dollars. Each data point contains the *Open*, *Close*, *High*, and *Low* exchange rates together with the Unix timestamp in milliseconds, which we transformed into a human-readable time. An example of the data is shown in Figure. 2.1.

	Open time	Open	High	Low	Close	Volume	Close time	Quote asset volume	Number of trades	Taker buy base asset volume	Taker buy quote asset volume	Ignore
time												
2023-01-25 00:00:00	1674604800000	1555.97	1557.00	1541.53	1548.88	42208.8517	1674608399999	6.541696e+07	49567	20460.9587	3.170906e+07	0
2023-01-25 01:00:00	1674608400000	1548.89	1550.47	1518.00	1535.00	67088.1066	1674611999999	1.028002e+08	65171	32969.2298	5.051652e+07	0
2023-01-25 02:00:00	1674612000000	1534.99	1543.06	1534.21	1540.87	13866.2509	1674615599999	2.134667e+07	21444	8061.2293	1.240901e+07	0
2023-01-25 03:00:00	1674615600000	1540.87	1546.88	1540.28	1546.63	12655.7856	1674619199999	1.954109e+07	16082	7778.4693	1.201070e+07	0
2023-01-25 04:00:00	1674619200000	1546.63	1549.01	1545.83	1546.17	9355.4083	1674622799999	1.447565e+07	14624	5263.4895	8.144337e+06	0
2023-01-25 05:00:00	1674622800000	1546.17	1551.74	1545.20	1549.14	10814.5467	1674626399999	1.673896e+07	15854	6048.4757	9.361519e+06	0
2023-01-25 06:00:00	1674626400000	1549.14	1557.03	1546.27	1555.16	16870.0489	1674629999999	2.617494e+07	27321	9610.9224	1.491241e+07	0
2023-01-25 07:00:00	1674630000000	1555.16	1558.91	1553.42	1553.81	12984.0760	1674633599999	2.020488e+07	20337	6508.8196	1.012887e+07	0

FIGURE 2.1
Example of ETH-USDT on 25.01.2023 after transforming the Unix timestamp to human readable time.

In order to evaluate different tokens equally, we choose the exchange rate of all these tokens to USDT (a stablecoin pegged to 1 US dollar). There are $N_0 = 394$ pairs in Binance's data that meet our criteria. Binance only provides the data since 1st March 2021. Therefore, we download the hourly K-line data for all these tokens from 1st March 2021 to 15th January 2023. It's worth mentioning that not all the tokens have exchange data during this period. Some tokens did not exist at the beginning, and some exited the market before the end. Each token has at most 16464 data points as the period contains $T_0 = 686$ days, namely $T_0 = 16464$ hours. The price change of a subset of assets is shown in Figure. 2.2, where most assets' prices are going down in this period.

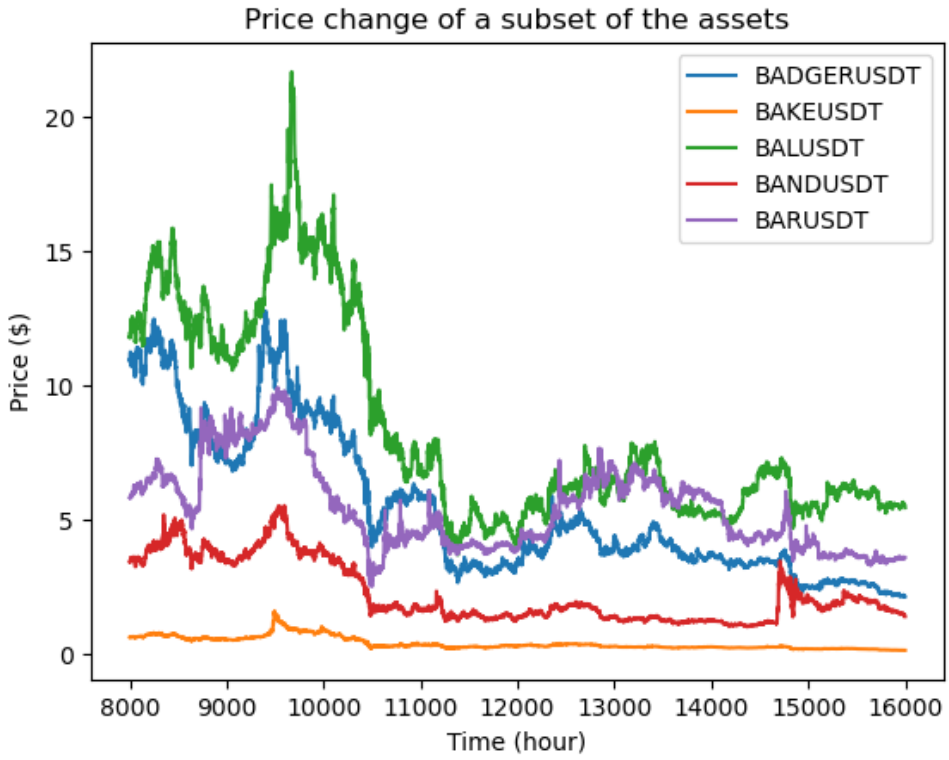

FIGURE 2.2
Close price of a subset of assets. During this period, all the assets price generally went down.

2.2 DATA PREPROCESSING

Binance provides a csv file for each token each day. For each token, we first merge all the 686 days' files into one data frame. Next, we merge the data frames for all tokens into one data frame.

We transform the Unix timestamp to the date and time format. Only the *Close* price is taken as the price data for the project and other unused data is dropped. Then we compute the log return of all the tokens. We obtain the raw data frame with 16452 rows and 394 columns. We take 8000 samples from 8000-16000 rows and remove the columns with NaN. Finally, we got a data frame with shape $(8000, 253)$ representing 8000 rows of data for 253 tokens as shown in Figure. 2.3.

	AAVEUSDT	ACAUSDT	ACHUSDT	ACMUSDT	ADADOWNUSDT	ADAUPUSDT	ADAUSDT	ADXUSDT	AIONUSDT	AKROUSDT	...	XTZUSDT	XVGUSDT
8000	0.016944	0.037596	0.010463	0.008332	-0.015881	0.018140	0.009597	0.009128	0.010351	0.020501	...	0.010782	0.013205
8001	0.002685	0.008717	0.009066	-0.013926	-0.001256	0.002657	0.000000	0.000491	0.003427	0.012102	...	0.012323	-0.003032
8002	-0.010782	-0.020038	-0.002582	-0.001531	0.004077	-0.007145	-0.002869	0.000736	-0.002283	0.000802	...	-0.010649	-0.005074
8003	0.009440	0.023133	0.003613	0.001786	-0.008329	0.011086	0.003824	0.006844	0.002283	0.013530	...	0.012303	0.003047
8004	0.004019	-0.003095	0.005395	0.000000	-0.003794	0.002833	0.001907	0.003647	0.003415	0.014129	...	0.004287	0.005058
...	...	...	...	...	...	...	...	...	...	...	...	...	...
15995	0.005231	-0.000821	0.002488	0.004274	0.001206	0.004728	0.000384	0.000000	0.002977	-0.012012	...	0.003889	0.000000
15996	-0.008734	-0.002468	-0.002488	-0.003417	0.007924	-0.014252	-0.004621	-0.001693	-0.005216	0.023882	...	-0.003889	-0.007092
15997	-0.003515	-0.004127	-0.006246	-0.001713	0.004891	0.000000	-0.004255	0.000000	0.008555	-0.008889	...	-0.005208	0.007092
15998	-0.005296	-0.004144	0.013690	0.002996	0.004749	-0.014458	-0.003106	0.001693	-0.000741	0.005935	...	-0.006549	0.003527
15999	0.001768	-0.004161	-0.009938	-0.003854	0.006846	-0.004866	-0.005849	-0.001693	0.001852	-0.002963	...	-0.009241	-0.010620

8000 rows × 253 columns

FIGURE 2.3
Log return of $N_0 = 253$ assets with $T_1 = 8000$ hours after pre-processing

CHAPTER 3

METHOD

3.1 COVARIANCE MATRIX CLEANING

Ideally, one can optimize the variance portfolio by the covariance matrix to maximize the interest and minimize the risk at the same time. However, the correlation of the assets cannot be trivially captured from the matrix as it is computed from a finite window of noisy financial data, which results in significant errors. In order to capture more robust correlations among the assets and build a profitable portfolio while hedging the risk, one needs to remove the noise in the covariance matrix. We used eigenvalue clipping, Rotationally Invariant Estimators (RIE), and Bootstrapped Average Hierarchical Clustering (BAHC) to clean the matrices.

3.1.1 EIGENVALUE CLIPPING

The main idea of eigenvalue clipping is that only the largest eigenvalues of the matrix give valuable information about the cross-correlations, while the remaining are random noise and can be discarded. Because the remaining smaller eigenvalues represent deviations from the true joint moments of the data. The shrinkage replaces the overfitting matrix by a linear combination of that covariance matrix with another underfitting estimator and this is achieved with a combining coefficient which helps to adjust the model structure to the data. This method is based on the assumption that returns follow a Gaussian distribution. With the time series of length T, assets amount N, normalized return matrix X ($X \sim N(0,1)$), the correlation matrix can be denoted as

$$C = \frac{1}{T} X'X \tag{3.1}$$

Assume $N, T \to \infty, q = N/T$, then the probability density of λ under the null hypothesis that $C = I$ is

$$P_0(\lambda) = \frac{1/q}{2\pi} \frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{\lambda}, \lambda_\pm = (1 \pm \sqrt{q})^2 \tag{3.2}$$

Replace all eigenvalues such that $\lambda_i \leq \lambda_+ \to \delta$, where $\delta = \frac{1}{N_{clip}} \sum_{\lambda_i \leq \lambda_+} \lambda_i$. Define $\Lambda^{clip} = (\lambda_i^{clip})$, V is the matrix of eigenvectors, then the correlation matrix after clipping is

$$C^{clip} = V' diag(\Lambda^{clip}) V \tag{3.3}$$

3.1.2 ROTATIONALLY INVARIANT ESTIMATORS

An RIE is a type of estimator that is not affected by rotations in the data. Even when the structure of the empirical correlation matrix C is not known, Cleaned eigenvalue $\xi(\lambda_i)$ of the true correlation matrix $C^{(RIE)}$ can still be computed by using RIE. This is achieved by removing the noise or estimation error present in C, and then computing the eigenvalues of the cleaned matrix to build $C^{(RIE)}$. According to [2], $\xi(\lambda_i)$ is denoted as

$$\xi(\lambda_i) = \frac{\lambda_i}{(1 - q + qz_i s)^2} \tag{3.4}$$

where s is the resolvent of E in terms of z_i, z_i is the complex number $\lambda_i - 1j/\sqrt{N}$.

Since N is not very large in the real world, a systematic downward bias affects $\xi(\lambda_i)$ for small eigenvalues of C. This bias can be heuristically corrected by

$$\hat{\xi}(\lambda_i) = \xi(\lambda_i) * max(1, \gamma) \tag{3.5}$$

where

$$\gamma = \frac{(1 - q + qz_i g_i)^2 \sigma^2}{\lambda_i}$$
$$g_i = \frac{z_i + \sigma^2(q-1) - \sqrt{(z_i - \lambda_0)(z_i - \lambda_+)}}{2q\sigma^2 z_i}$$
$$\sigma^2 = \frac{\lambda_0}{(1 - \sqrt{q})^2}$$
$$\lambda_+ = \frac{(1 + \sqrt{q})^2}{(1 - \sqrt{q})^2} * \lambda_0$$

Finally, we can construct the filtered correlation matrix by

$$C^{(RIE)} = V' diag(\left(\hat{\xi}(\lambda_i)\right))V \tag{3.6}$$

3.1.3 BOOTSTRAPPED AVERAGE HIERARCHICAL CLUSTERING

BAHC rests on multiple hierarchical structures weighted by their frequency. A single hierarchical structure will only emerge if all the bootstrap realizations lead to the same dendrogram. Thus, data matrix bootstrap resampling of the feature indices can be applied, which better accounts for the influence of randomness on the inferred structure [5].

The hierarchical clustering algorithm defines the distance between different assets by $d_{i,j} = 1 - c_{i,j}$, where $c_{i,j}$ is the entry of the Pearson correlation matrix. Next, it leverages the average linkage between clusters to define the distance among clusters. For cluster $\mathfrak{C}_p, \mathfrak{C}_q$, the average linkage is defined as

$$\rho_{pq} = \frac{\sum_{i \in \mathfrak{C}_p} \sum_{j \in \mathfrak{C}_q} (1 - c_{ij})}{|\mathfrak{C}_p||\mathfrak{C}_q|} \tag{3.7}$$

Initially, all assets have their own cluster. The hierarchical clustering algorithm merges the pair of clusters with minimum distance into a new cluster. The algorithm recursively joins a pair of clusters until all nodes fall into a single unique cluster. Then it builds a matrix $C^<$ where

$$c_{ij}^< = c_{ji}^< = 1 - \rho_{pq} \tag{3.8}$$

It is believed that the limitation of HCAL is that it does not consider the presence of overlap between clusters [5].

BAHC is built upon HCAL by applying HCAL to m bootstrap copies of the data matrix and obtaining m HCAL filtered matrx $C^{(m)<}$. Finally, the filtered Pearson correlation matrix c_{ij}^{BAHC} can be constructed as

$$c_{ij}^{BAHC} = \sum_{b=1}^{m} \frac{c_{ij}^{(b)<}}{m} \tag{3.9}$$

3.2 ROLLING WINDOW

As the financial system is a strongly non-stationary system, we employ rolling window computation. We first try different in-sample window sizes. For each in-sample window, we move the window by a $stepsize = 5$ hour for around 900 trials and average the in-sample risk and the out-of-sample risk.

3.3 IN- AND OUT-OF-SAMPLE RISK

The optimal weights w of MVP are computed by minimizing the variance at fixed gains G. The cost function is shown below.

$$F_{cost} = w^{'} C w + \lambda (w^{'} g - G) \tag{3.10}$$

And the solution is:

$$w^{*} = G \frac{C^{-1} g}{g^{t} C^{-1} g} \tag{3.11}$$

For different strategies, we apply the formula above to the cleaned covariance matrix and compute w. We compute the in-sample and out-of-sample risk given the MVP weights we obtained by different strategies. The risk is defined as:

$$R_{in}^{2} = w_{in}^{*'} C_{in} w_{in}^{*} \tag{3.12}$$

$$R_{out}^{2} = w_{in}^{*'} C_{out} w_{in}^{*} \tag{3.13}$$

where w_{in} represents the optimal weights computed from the in-sample. C_{in} and C_{out} represent the covariance matrices of in-sample and out-of-sample respectively.

CHAPTER 4

RESULT

4.1 RETURNS AND VARIANCE

We take the matrix with no covariance cleaning as the baseline and compare the performance of eigenvalue cleaning, NLS, and BAHC. We first compute the optimal weights by each method on the in-sample window, then apply the weights to a longer out-of-sample window and display the cumulative returns of each asset. As shown in Figure. 4.1(b), BAHC achieves the best performance regarding cumulative returns. However, eigenvalue clipping and NLS only lead to better performance on in-sample as shown in Figure. 4.1(a), while causing more loss on out-of-sample. It is believed that overfitting is caused in this case, and the dependence they extracted does not generalize well.

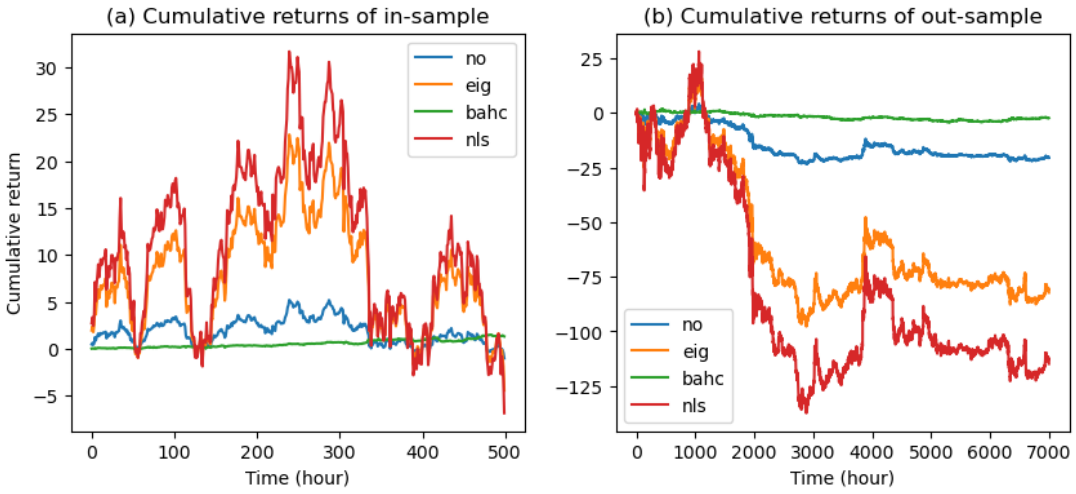

FIGURE 4.1
Comparison of cumulative price returns of the portfolio among different methods. **no**: without covariance cleaning. **eig**: with eigenvalue clipping. **bahc**: with BAHC. **nls**: with optimal shrinkage.

4.2 ROLLING WINDOW CALIBRATION

We applied the rolling window calibration to our final results of MVP risks. We choose the in-sample window size from 300 hours to 1400 hours, and the out-of-sample window size fixed at 2500 hours. For rolling, we make the start of the in-sample window from 0 to 4000, with $stepsize = 5$ (hours). The out-of-sample is selected by just following the in-sample window. Besides, the realized risks decrease

TABLE 4.1
Relative difference (%) between out-of-sample risk and in-sample risk of four methods.

window size	no clipping	nls	bahc	eigenvalue clipping
300	178.236388	-0.099847	0.396855	-0.107134
400	37.110558	-0.116142	0.231552	-0.123758
500	20.552604	-0.131628	0.139052	-0.140268
600	14.422998	-0.145655	0.073188	-0.155756
700	11.437105	-0.155661	0.028601	-0.167199
800	9.445718	-0.165799	-0.012401	-0.177870
900	7.754334	-0.177208	-0.045468	-0.191510
1000	6.511232	-0.191469	-0.075566	-0.207827
1100	5.508842	-0.207617	-0.103203	-0.225544
1200	4.754395	-0.225349	-0.128876	-0.244597
1300	4.146155	-0.244819	-0.153864	-0.264327
1400	3.648680	-0.265041	-0.177337	-0.284915

when the in-sample window size increases. As shown in Figure. 4.2, the BAHC's realized risk is the smallest among all methods. In addition, we also calculated the difference between out-of-sample risk and in-sample risk of four methods as shown in Table. 4.1. Once again, we see that BAHC has the smallest percentage difference most of the time. Thus, it can be concluded that BAHC has the best performance in covariance matrix filtering and MVP.

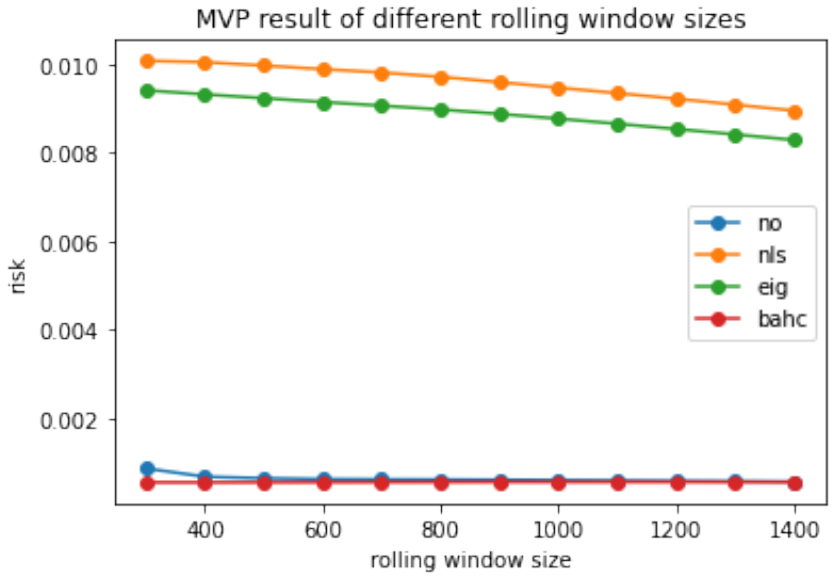

FIGURE 4.2
Comparison of realized risks of four methods after calibration

We also observed an abnormal phenomenon in our results. As shown in Figure. 4.3, except for the results of no clipping, the other three methods of filtering all lead to the out-of-sample risk larger than the in-sample risk, which is not reasonable at all in our case. We did lots of research to find out the issue, and the possible answer is as follows.

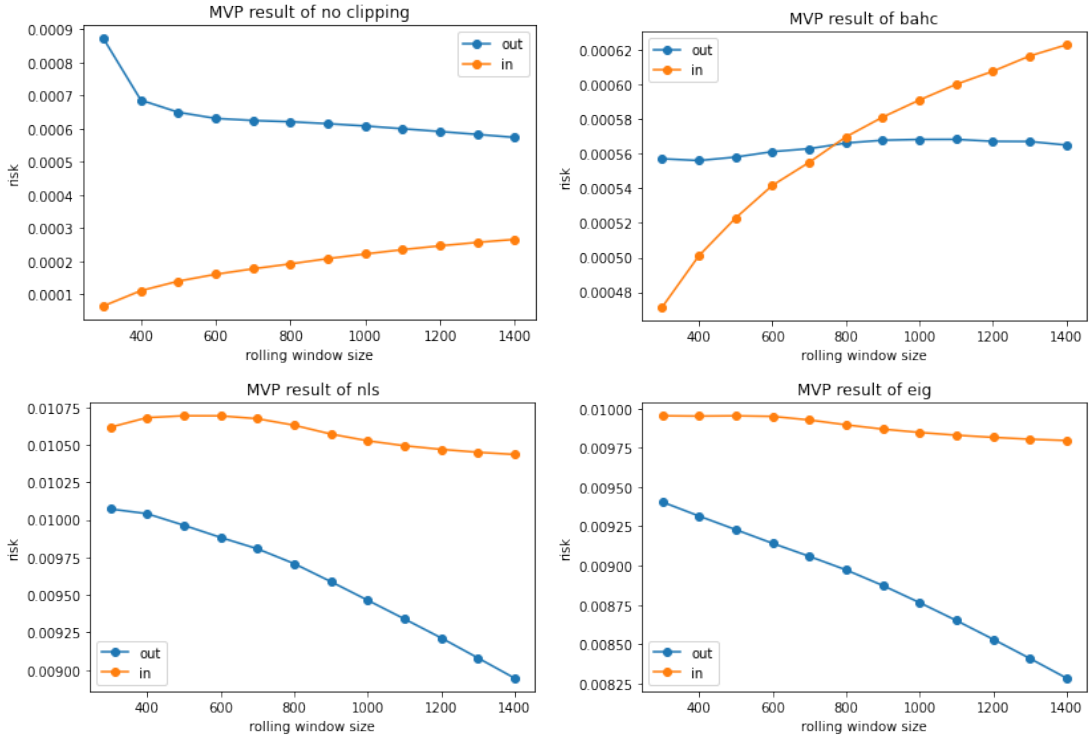

FIGURE 4.3
Comparison of in-sample and out-of-sample realized risks of four methods after calibration

As shown in Figure. 4.4, the realized risk rises an order of magnitude higher between 1500 to 2500 (hour), with a rolling window size of 800 hours. Thus, there must be a huge fluctuation during this time, compared to the remaining time span. Since the actual time of 0 hour is on 14-03-2022, we can deduce that the huge fluctuation happened from 15-05-2022 to 26-06-2022. We make a snapshot of the capitalization of all crypto assets from 14-03-2022 to 15-01-2023 (i.e. 0 to 8000 hours) as shown in Figure. 4.5. We can see that there are truly two huge crashes in the cryptocurrency market. They both happened between May and July of 2022. As the out-of-sample window falls in the big crashes, it cannot represent a typical market situation. These crashes could lead to a decrease in the realized risk as everything is crashing. A much longer out-of-sample window may be better for the high volatile cryptocurrency.

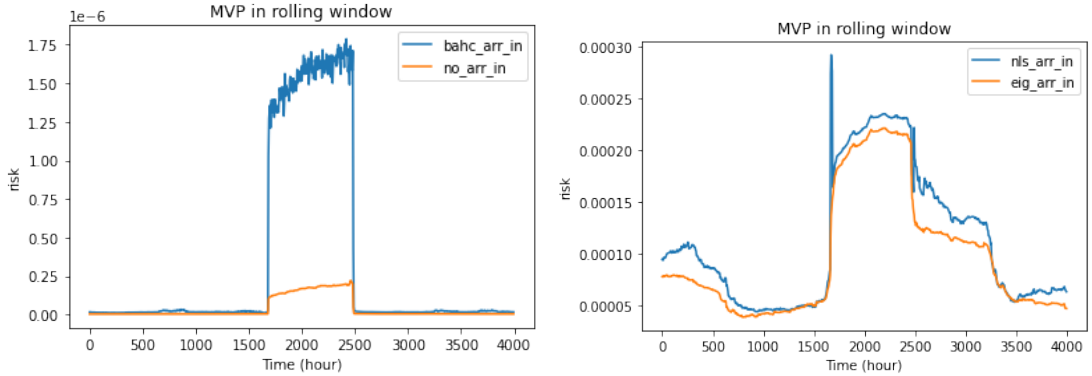

FIGURE 4.4
Realized risks of four methods across rolling windows with window size 800 (hours), x-axis values represent the starting time point of the window

FIGURE 4.5
Total Cryptocurrency Market Capitalization: the capitalization of all crypto assets, including stablecoins and tokens

In summary, our results show that eigenvalue clipping and linear shrinkage can lead to the loss of important information from the covariance matrix, which negatively influences the portfolio (i.e. lead to a larger risk compared to no clipping). The result of these two methods may be too sensitive to the threshold value in eigenvalue clipping and the target matrix in linear shrinkage together with the constant multiple. BAHC yields the best performance by grouping similar assets together, however, it is computationally expensive as it creates a lot of bootstrapped samples.

CHAPTER 5

DISCUSSION

5.1 CHALLENGE

5.1.1 DATA DOWNLOAD

The page where Binance displays the data URLs is dynamically fetched from the Binance database, which means every time we download one file, the Binance website engine will request a query to the database. In order to accelerate the downloading speed, we use *dask* library to download multiple files in parallel. We also achieve download recovery by checking which assets are not downloaded. It took around 12 hours to download all the raw data.

5.1.2 DATA MERGE

The complexity of naive merge is $\mathcal{O}(T_0 \times N_0)$, where T_0 represents the time and N_0 represents the number of assets. This incurs heavy computation overhead. In order to improve the merge efficiency, we divide the files into small chunks. We merge the bottom-level chunks first and merge the result chunks in a recursion way. Theoretically, this merge algorithm achieves $\mathcal{O}(log(T_0 \times N_0))$ complexity.

5.1.3 RUNNING

In the experiments, we did around 800 trials with a sliding window for each strategy to estimate the risk. To accelerate the process, we also use *dask* to run several trials in parallel. We use a laptop of Intel(R) Core (TM) i5-1035G1 CPU @ 1.00GHz to run all the tasks, which takes around 1 day.

5.2 IMPROVEMENTS

As MVP is built on top of historical data, it may not represent future market behaviors. Besides, it is not always true that the log return of price follows a normal distribution. The risk is even magnified in cryptocurrency where there are more *Black Swan* events in the crypto world that drastically change the market overnight, such as the crash of LUNA [6].

Different from the mature stock market, cryptocurrency is still at an early age where there is a lot of uncertainty regarding security, utility, law, and regulations. Some crypto assets may be highly weighted in a portfolio by mathematical analysis, while huge risks may be hidden behind the asset protocol itself. A good MVP in cryptocurrency should also consider the intrinsic risk of assets through their white papers, develop teams, and backers.

BIBLIOGRAPHY

[1] Satoshi Nakamoto. 'Bitcoin whitepaper'. In: *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* (2008).

[2] Joël Bun, Jean-Philippe Bouchaud and Marc Potters. 'Cleaning large correlation matrices: tools from random matrix theory'. In: *Physics Reports* 666 (2017), pp. 1–109.

[3] Michele Tumminello, Fabrizio Lillo and Rosario N Mantegna. 'Hierarchically nested factor model from multivariate data'. In: *EPL (Europhysics Letters)* 78.3 (2007), p. 30006.

[4] Christian Bongiorno, Salvatore Miccichè and Rosario N Mantegna. 'Nested partitions from hierarchical clustering statistical validation'. In: *arXiv preprint arXiv:1906.06908* (2019).

[5] Christian Bongiorno and Damien Challet. 'Covariance matrix filtering with bootstrapped hierarchies'. In: *PloS one* 16.1 (2021), e0245092.

[6] Ekin Genç Krisztian Sandor. 'The Fall of Terra: A Timeline of the Meteoric Rise and Crash of UST and LUNA'. In: (2022).

APPENDIX

```
import pandas as pd
import pyarrow.parquet as pq
import numpy as np
import matplotlib.pyplot as plt
import wget
import dask
import os
from tqdm import tqdm
import glob
import os

from numpy import linalg as LA
import math
import sklearn.preprocessing
import bahc
import pyRMT

%matplotlib inline
```

Download k-line spot data by every minute from Binance

```
url_template = r"https://data.binance.vision/data/spot/daily/klines/{0}/1h/{0}-1
dir_template = r"data/spot/daily/klines/{0}/1h/{0}-1h-{1}.zip"


# @dask.delayed
def download_one_date(url, path):
    try:
        tmp = wget.download(url, out=path)
        return True
    except:
        # print("{} download failed".format(url))
        return False

def download_all_dates(token_pair: str, dates, path):
    first_meet = False
    for each in dates:
        res = download_one_date(url_template.format(token_pair, str(each.date())
        if first_meet == False and res == True:
```

```
            first_meet = True
        if first_meet == True and res == False:
            break
    # promises = [download_one_date(url_template.format(token_pair, str(each.dat
    # alldata=dask.compute(promises)

def get_asset_pairs(x):
    with open("asset_pairs.txt", "r") as f:
        names = f.read()
    names = names.replace("\t", "")
    names = names.replace("\n", "")
    names = names.split("/")
    x_names = list(filter(lambda each: each.endswith(x), names))
    print("x: {} results length: {}".format(x, len(x_names)))
    return x_names

@dask.delayed
def main_download(pair, dates):
    path = "data/spot/daily/klines/{0}/1h".format(pair)
    if not os.path.exists(path):
        os.makedirs(path)
    download_all_dates(pair, dates, path)

USDT_pairs = get_asset_pairs("USDT")
dates = pd.date_range(start="2021-03-01",end="2023-01-15")
print(len(list(dates)))

promises = [main_download(each, dates) for each in USDT_pairs]
dask.compute(promises)
```

Data loading pre-processing

```
@dask.delayed
def process_raw(pair, path):
    names = [
        "Open time",
        "Open",
        "High",
        "Low",
        "Close",
        "Volume",
        "Close time",
        "Quote asset volume",
        "Number of trades",
        "Taker buy base asset volume",
        "Taker buy quote asset volume",
        "Ignore",
    ]
    asset_data = pd.read_csv(path, names=names, header=None)
    # btcdata = pd.read_csv(dir_template.format(pair, date), names=names, header
```

```
    asset_data["time"] = pd.to_datetime(asset_data["Open time"], unit='ms')
    asset_data[pair] = asset_data["Close"]
    date_indexed = asset_data.set_index("time")
    date_indexed.drop([
        "Open time",
        "Open",
        "High",
        "Low",
        "Close",
        "Volume",
        "Close time",
        "Quote asset volume",
        "Number of trades",
        "Taker buy base asset volume",
        "Taker buy quote asset volume",
        "Ignore",
        ], axis=1, inplace=True)
    # date_indexed.drop('Close time', axis=1, inplace=True)

    # date_indexed.drop('time', axis=1, inplace=True)

    # date_indexed["s"] = (date_indexed["isBuyerMaker"].astype(int)-0.5)*(-2)
    # date_indexed["mid"] = date_indexed["price"]
    return date_indexed


def load_one_pair(pair):
    files = glob.glob("data/spot/daily/klines/{}/1h/*".format(pair))
    files = [each for each in files if "(" not in each]
    if len(files) == 0:
        print(f"{pair} is empty, no files found")
        return False, None
    tasks = [process_raw(pair, each) for each in files]
    p_data_arr = dask.compute(tasks)
    result = pd.concat(p_data_arr[0])
    return True, result

def merge_assets(pd_arr, col: str):
    assets_close_matrix = pd_arr[0]
    for each in pd_arr[1:]:
        assets_close_matrix = assets_close_matrix.merge(each, how="outer", on=co
        # assets_close_matrix = assets_close_matrix.join(each)
        # print(assets_close_matrix.shape)
        if assets_close_matrix.shape[0] > 16464:
            print(f"{each.columns} wrong rows: {assets_close_matrix.shape[0]}")
            assert True==False
    return assets_close_matrix

def main_load_and_merge_all_assets():
```

```
    existing_pairs = os.listdir("data/spot/daily/klines")
    print("Number of pairs: ", len(existing_pairs))
    chunk_sz = 10
    subsets = [existing_pairs[i:i + chunk_sz] for i in range(0, len(existing_pai
    # print(sum([len(each) for each in subsets]))
    for i in tqdm(range(len(subsets))):
        chunk = subsets[i]
        tmp_assets_arr = list()
        for each in chunk:
            success, tmp_asset = load_one_pair(each)
            if success:
                if tmp_asset.shape[0] > 16464:
                    print(f"{each} wrong rows: {tmp_asset.shape[0]}")
                tmp_assets_arr.append(tmp_asset)
        # tmp_assets_arr = [load_one_pair(each) for each in chunk]
        tmp_merge_result = merge_assets(tmp_assets_arr, "time")
        tmp_merge_result.to_pickle(f"data/clean/tmp_merge_{i}.pkl")


def inspect_assets_shape():
    existing_pairs = os.listdir("data/spot/daily/klines")
    print("Number of pairs: ", len(existing_pairs))
    # assets_pd_arr = [load_one_pair(each) for each in existing_pairs]
    for pair in existing_pairs:
        success, tmp_asset = load_one_pair(pair)
        assert tmp_asset.shape[0] <= 16464, f"{pair} wrong rows: {tmp_asset.shap

# inspect_assets_shape()
res = main_load_and_merge_all_assets()

def merge_all_chunks(arr_id):
    df_arr = [pd.read_pickle(f"data/clean/mk_{i}.pkl") for i in arr_id]
    # for each in df_arr:
    #     print(each.shape)
    res = merge_assets(df_arr, "time")
    return res

res = merge_all_chunks([i for i in range(40)])
res.to_pickle("data/clean/whole_usdt_merge.pkl")
print(res.shape)

all_data = pd.read_pickle("data/clean/whole_usdt_merge.pkl")
all_data = all_data.reset_index()
all_data = all_data.drop(["time"], axis=1)
t0 = 8000
t1 = 16000
X_raw = log_ret_all_data.iloc[t0:t1].dropna(axis=1)
all_data_length = X_raw.shape[0]
T = 500
```

```
T_test = 2500
N = 253

T_wsf_list = [300, 700, 900, 1100, 1300, 1500]
T_mk_list = [400, 600, 800, 1000, 1200, 1400]
T_list = T_wsf_list + T_mk_list

step_range = 4000
step_size = 5
```

MVP functions

```
def eigenvalue_clipping(lambdas,v,lambda_plus):
    N=len(lambdas)

    # _s stands for _structure below
    sum_lambdas_gt_lambda_plus=np.sum(lambdas[lambdas>lambda_plus])

    sel_bulk=lambdas<=lambda_plus                          # these eigenvalues come f
    N_bulk=np.sum(sel_bulk)
    sum_lambda_bulk=np.sum(lambdas[sel_bulk])
    delta=sum_lambda_bulk/N_bulk                           # delta is their average,

    lambdas_clean=lambdas
    lambdas_clean[lambdas_clean<=lambda_plus]=delta


    C_clean=np.zeros((N, N))
    v_m=np.matrix(v)

    for i in range(N-1):
        C_clean=C_clean+lambdas_clean[i] * np.dot(v_m[i,].T,v_m[i,])

    np.fill_diagonal(C_clean,1)

    return C_clean


def solution_eig(C_asset):
    # C_corr = C_asset.corr()
    C_cov = C_asset.cov()
    l_e, V_e = LA.eig(C_cov)
    T, N = C_asset.shape
    q = N/T
    lambda_plus = (1+np.sqrt(q))**2

    C_clipped=eigenvalue_clipping(l_e,V_e,lambda_plus)
    return C_clipped

def weights_GVM(Sigma):
```

```
    Sigma_inv=LA.inv(Sigma)
    w_GVM=Sigma_inv.sum(axis=1)/Sigma_inv.sum()
    return w_GVM

@dask.delayed
def run_one_iteration(i, window, method="no"):
    # |train|    test     |
    # |--T--|------T------|
    C_asset_in = X_raw.iloc[i:i+window]
    C_asset_out = X_raw.iloc[i+window:i+window+T_test]

    # covariance matrix
    C_in_cov = C_asset_in.cov().values
    C_out_cov = C_asset_out.cov().values

    if method=="no":
        # 1. no correlation cleaning
        w = weights_GVM(C_in_cov)

    elif method=="eig":
        # 2. eigenvalue clipping
        C_clipped_in = solution_eig(C_asset_in)
        w = weights_GVM(C_clipped_in)

    elif method=="bahc":
        # 3. cleaning with bahc
        X_centered = sklearn.preprocessing.StandardScaler(with_mean=True,
                                with_std=False).fit_transform(C_asset_in.values)
        Sigma_BAHC=bahc.filterCovariance(X_centered.T)
        w=weights_GVM(Sigma_BAHC)

    elif method=="nls":
        # 4. cleaning with nls
        X_centered = sklearn.preprocessing.StandardScaler(with_mean=True,
                                with_std=False).fit_transform(C_asset_in.values)
        Sigma_NLS=pyRMT.optimalShrinkage(X_centered, return_covariance=True)
        w=weights_GVM(Sigma_NLS)

    # in sample risk
    sigma_in = w.T@(C_in_cov@w)

    # out sample risk
    sigma_out = w.T@(C_out_cov@w)

    return sigma_in, sigma_out

def run_one_T(t=300, method="no"):
    arr = {"in": [], "out": []}
```

```
    promises = [run_one_iteration(i, t, method) for i in range(0, step_range, st
    res = dask.compute(promises)

    for each in res[0]:
        sigma_in, sigma_out = each
        arr["in"].append(sigma_in)
        arr["out"].append(sigma_out)

    res_df = pd.DataFrame()
    res_df[f"{method}_arr_in"] = arr["in"]

    res_df[f"{method}_arr_out"] = arr["out"]

    res_df.to_pickle(f"data/clean/in_out_risk_{method}_{t}.pkl")

    print(res_df.mean())

for each in ["no", "eig", "nls", "bahc"]:
    for i in tqdm(range(len(T_in_list))):
        run_one_T(T_in_list[i], each)
```