The Chinese University of Hong Kong, Shenzhen

**STA4020 Statistical Modelling in Financial Market**

---

**Project Report**

# The BTRM: Improvement of Markowitz Model Based on Adapted Risk Measure, Bayes Estimation, and Momentum

---


| | | |
|---|---|---|
| Advisor: | Prof. Sang Hu | |
| Students: | Huihan Yang | 120090438 |
| | Jianyu Zhu | 119010489 |
| | Junle Li | 118010142 |
| | Yangyifan Hui | 120090106 |
| | Yaoyu Yang | 119010385 |


SHENZHEN, JANUARY 2023

# Contents

**1 Abstract** **3**

**2 Introduction** **3**

**3 Data** **4**

**4 Methods** **4**

4.1 Overview . . . . . 4

4.1.1 Model Flow . . . . . 4

4.1.2 Time-Variant Data Selection . . . . . 4

4.2 Bayesian Percentile Estimation . . . . . 5

4.2.1 Static risk-preference estimation . . . . . 6

4.2.2 Dynamic risk-preference estimation . . . . . 6

4.3 Momentum Filter . . . . . 6

4.3.1 Momentum filter with smoothing kernel . . . . . 6

4.3.2 Momentum filter with Bayesian kernel . . . . . 7

4.4 Risk-Budgeting Model . . . . . 7

**5 Result & Analysis** **8**

5.1 Result . . . . . 8

5.1.1 Measurement . . . . . 8

5.1.2 Static risk-preference model . . . . . 9

5.1.3 Dynamic risk-preference model . . . . . 9

5.1.4 Momentum filter with smoothing kernel . . . . . 10

5.1.5 Momentum filter with Bayes kernel . . . . . 11

5.2 Advantage compared with traditional Markowitz model . . . . . 12

5.3 Future development . . . . . 12

**6 Conclusion** **12**

**7 Appendix** **14**

# 1 Abstract

In the project, we establish the Bayes-Based Time-Variant Risk-Preference Model(BTRM) to build up the investment strategy. The BTRM consists of three parts, the Bayesian Percentile Estimation(BPE), which estimate the excess return consistent to the risk-preference of investors, the Momentum Filter(MF), which applies momentum to filter the valuable stock pool, and the Risk Budgeting Model(RBM), which gives us the monthly portfolio by minimizing the total Risk Contribution(RC).

We validate the effectiveness of BTRM with the stocks pool from the constituents of CSI 300 Index from December, 2009 to November, 2022. We analyze the performance of our portfolio through several common indicators among which include the accumulated return, Sharp Ratio, maximum draw down and Calmar Ratio, compared with the bench mark model. We find that at most of the time, BTRM chooses the relatively risk-averse appetite. Additionally, the Dynamic BPE and the Baysian MF show a prudent investment behaviour, especially when the stock market slumps.

**Keywords: BTRM, Bayes, Momentum, RBM, Risk-preference**

# 2 Introduction

The Markowitz model, also known as the mean-variance model, is a mathematical model for constructing a portfolio of assets that provides the highest expected return for a given level of risk. It was developed by economist Harry Markowitz in the 1950s and is a cornerstone of modern portfolio theory [4]. The Markowitz model has been widely influential in finance and has been applied to a variety of investment decisions, including the selection of individual stocks, bonds, and other securities for inclusion in a portfolio. However, it also have some few problems. Firstly, the Markowitz model focuses on selecting individual securities based on their expected return and risk, which may not align with the investor's overall risk tolerance and investment objectives. Also, the model assumes that the statistical measures of risk and return are accurate and reliable, are based on historical data and may not accurately reflect future risk and return. Moreover, The model assumes that investors can easily buy and sell any asset in the portfolio at any time. However, this may not always be possible in practice due to market liquidity constraints.

Based on the aforementioned analysis, we set out to modify the classical Markowitz model and proposed our model named as Bayes-Based Time-Variant Risk-Preference Model(BTRM).

In the model, we improve the model in the prospects of Bayes estimation, momentum selection and risk-budgeting model. Firstly, instead of directly using historical data in the optimization in Markowitz model, we use Bayesian methods to predict the excess return and use it to do optimization allocation or selection. In that case, we have an innovative explanation on the percentiles of the posterior distribution, that we relate the excess return lying on the lower tail to the Risk-Averse Estimation(RAE), the excess return lying on the middle part to the Risk-Neutral Estimation(RNE), and the excess return lying on the upper tail to the Risk-Seeking Estimation(RSE). Secondly, instead of directly allocate all the stocks through optimization as Markowitz does, we add one more momentum filter to give higher condition restrictions on the stock before doing optimization allocation. Thirdly, the classical Markowitz model constructs the optimal portfolio by minimizing the risk, which is measured by the portfolio variance of excess return. However, recent studies have conducted the modification of Markowitz model that, based on the same idea, they tried other risk indexes, such as the Value at Risk(VaR), the Conditional Value at Risk(CVaR), and the Risk Contribution(RC). In the project, we choose to use the RC as the risk indicator since the RC-based portfolio outperforms the other indicators in measuring the percentage risk allocation in multi-asset portfolios [1]. Additionally, we add necessary constraints accordant with the Chinese Stock Market to make the investment strategy practical.

# 3 Data

Our project is based on the Chinese stock market. For typicality, we choose the CSI 300 Index (000300.SS) as the benchmark, which is a good gauge of Chinese stock market generally.

The CSI 300 Index is a capitalization-weighted stock market index designed to replicate the performance of the top 300 A-share stocks traded on the Shanghai Stock Exchange and the Shenzhen Stock Exchange. It can be seen as a barometer of the overall performance of Chinese stock market. Its constituent stocks are mostly representative with high liquidity and large size, covering numerous industries. Based on the principle of stability and dynamic tracking, its constituents shall be reviewed every 6 months.

We select stocks which were constituents of CSI 300 Index from December, 2009 to November, 2022. After eliminating Growth Enterprise Market stocks, STAR Market stocks and stocks which were ever delisted or under special treatment, we get a population of 430 stocks.

We retrieve and download the following data from China Stock Market & Accounting Research (CSMAR) Database:

- monthly returns of CSI 300 Index in the period 2009/12 - 2022/11;
- monthly returns of 430 selected stocks in the period 2009/12 - 2022/11;
- monthly risk-free rates in the Chinese bond market in the period 2009/12 - 2022/11, which can be directly found in CSMAR and herein used as monthly risk-free rates in the model;
- historical weights of CSI 300 Index constituents from 2009-12-01 to 2022-12-08.

By subtractions, we can get monthly excess returns.

Our data preparation pipeline rearranges and pre-processes the data for convenient use.

# 4 Methods

## 4.1 Overview

The target of the project is to build up the monthly dynamic investment portfolio with time-variant risk-preference using the Bayes-Based Time-Variant Risk-Preference Model(BTRM). Specifically, we regard the change of risk-preference as being positively consistent with the change of index, which is CSI 300 Index as chosen[6].

### 4.1.1 Model Flow

The realization of the BTRM is based on an iterative algorithm in which each iteration consists of three steps. Firstly, we use the Bayesian Percentile Estimation(BPE) to estimate posterior distribution of excess return in the next month, and choose the accordant percentile based on the risk preference. Secondly, we throw the estimated excess return into a momentum filter and get the adjusted investment pool. Lastly, we use the modified Markowitz Model, called the Risk-Budgeting Model(RBM), to generate the portfolio.

### 4.1.2 Time-Variant Data Selection

Denote the monthly excess return for stock i in month t as $R_{t,i}$. In the iteration for month T, denote information of historical excess returns as $\mathbf{R}_T = \{R_{t,i},\ t = 1, 2, \cdots, T,\ i \in S_T\}$, where $t = 1$ represents month 2010/01, $t = 2$ represents month 2010/02, $\cdots$, the rest $t$ are denoted in a similar fashion; $S_T$ denotes the set of stocks in this iteration.

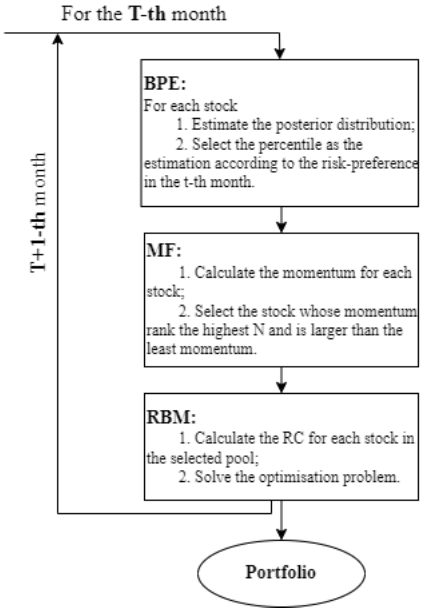



Figure 1: The basic structure of BTRM

In each iteration of month T, we decide $S_T$ from the 430 selected stocks by including those were constituents of CSI 300 Index for previous 2 years before T, i.e., from month T-24 to month T.

We set the range of T from month 2017/12 to 2022/10, accordingly we have T+1 from month 2018/01 to 2022/11. Now, in the iteration for month T, we use information $\mathbf{R}_T$ in BTRM to find estimate of the optimal portfolio and apply the resulting portfolio in the following month T+1 to calculate its realized monthly return.

After iterations, we attain the realized monthly returns of the estimated optimal portfolios from 2018/01 to 2022/11 and conduct out-of-sample test to compare the performance.

## 4.2 Bayesian Percentile Estimation

The Bayesian estimation has been introduced into portfolio analysis because the Bayesian framework neatly accounts for the practical problems which are parameter uncertainty and model uncertainty encountered by all investors, whereas standard statistical models often ignore them [2]. In this project, we choose the normal-normal conjugate family of distributions, which is a natural and common informative prior in portfolio analysis under the Bayesian framework. Therefore, we consider a normal distribution for the excess return $R_{t,i}$ for each stock i in every month t, and a normal prior for $\mu_{t,i}$, the expectation of $R_{t,i}$. The conjugate prior is given by

$$R_{t,i} \sim N(\mu_{t,i}, \sigma_i^2) \tag{1}$$

and

$$\mu_{t,i} \sim N(a_i, b_i) \tag{2}$$

where $a_i$ is the time-constant prior mean estimated by the sample mean of all historical excess return of stock i, and $b_i$ is the time-constant prior variance estimated by the adjusted sample variance of all historical excess return of stock i. In the month t, given the excess return the past $n = 10$ months, $R_{t-10,i}, \cdots, R_{t-1,i}$, the posterior distribution of $\mu$ given the past excess return is updated by

$$\mu | R_{t-10,i}, \cdots, R_{t-1,i} \sim N(\tilde{\mu}_{t,i}, \tilde{\sigma}_{t,i}^2) \tag{3}$$

where

$$\tilde{\mu}_{t,i} = \frac{\sigma_i^2 a_i}{\sigma_i^2 + nb_i^2} + \frac{nb_i^2 \bar{R}_i}{\sigma_i^2 + nb_i^2} \quad \tilde{\sigma}_{t,i}^2 = \frac{b_i^2 \sigma_i^2}{\sigma_i^2 + nb_i^2} \tag{4}$$

#### 4.2.1 Static risk-preference estimation

The individual estimation of excess return is highly correlated to the personal risk-preference. Empirically, the risk-inverse investor tends to underestimate the excess return, while the risk-seeking investor tends to overestimate the excess return. Therefore, we propose an innovative meaning of the percentile of posterior distribution that the lower tail refers to the underestimation of excess return and the upper tail refers to the overestimation of excess return. The percentile estimation $\mu_{i,t}^{\alpha}$ of stock $i$ in the month $t$ is given by

$$\alpha = P(\mu_{i,t} \leq \mu_{i,t}^{\alpha} | R_{t-10,i}, \cdots, R_{t-1,i}), \ 0 < \alpha < 1 \tag{5}$$

The greater the value of $\alpha$ is, the higher the level of risk-seeking of investors is. Specifically, we choose $\alpha = 0.0, 0.1, 0.2, 0.3, \cdots, 0.9$, ten levels of risk-preference, to construct ten static BPEs.

For code, please check github repository (links attached in the end): Bayesian_Portfolio.ipynb model A.

#### 4.2.2 Dynamic risk-preference estimation

Based on the Static risk-preference estimation, we could further create the dynamic BPE in which the risk-preference, or the choice of $\alpha$ is dependent on the time $t$. We select the CSI 300 index as the indicator of the investor's risk preference. Specifically, we define the Index Indicator at time t as followed. Given the indexes at the time window T, which are $I_{t-T}, \cdots, I_{t-1}$, we define the $II_t$ as the exponentially weighted sum of index differences:

$$\Delta_i = I_{t-1-i} - I_{t-2-i} \quad \forall i = 0, 1, \cdots, T-2 \tag{6}$$

the $II_t$ is

$$II_t = \frac{\Delta_0 + (1-\alpha)\Delta_1 + \cdots + (1-\alpha)^{T-2}\Delta_{T-2}}{1 + (1-\alpha) + \cdots + (1-\alpha)^{T-2}} \tag{7}$$

We choose $\alpha = \frac{1}{3}$. Based on the percentile of $II_t$ in all index differences within the whole past time window, we could select the corresponding $\alpha_t$ from $\{0.05, 0.10, 0.20, 0.50, 0.80, 0.90\}$ to realize the time-variant BPE. For example, if $II_t$ lies in the percentage interval $[0, 0.05)$, $\alpha = 0.05$.

For code, please check github repository (links attached in the end): Bayesian_Portfolio.ipynb model B.

### 4.3 Momentum Filter

There is substantial evidence that the performance of a stock in the past three- to twelve- month period is an important indicator to reveal the current performance of the stock. Until recently, trading strategies that exploit this phenomenon were consistently profitable in the United States and in most developed markets [3]. Therefore, the past performance, which is defined as momentum, is a practical filter that we apply to choose the stocks whose estimated excess return is more valuable than others'.

In the project, we apply the MF with two kinds of kernel. The first one is the smoothing linear kernel with adaptable number of momentum months; the second is the Bayesian kernel founded on the posterior distribution acquired in the BPE.

#### 4.3.1 Momentum filter with smoothing kernel

The MF with smoothing kernel apply a linear kernel constructed by the weighted sum of past excess return. Here, we use the equal weights on through the a period of T months in the past. The momentum of each stock

**Algorithm** Momentum Filter in the iteration of month T

**Input:** information of historical excess returns $\mathbf{R}_T = \{R_{t,i},\ t = 1, 2, \cdots, T,\ i \in S_T\}$, rank threshold $N$, minimum threshold $Mom_{min}$

**Output:** set of selected stocks after filter

1: Calculate the momentum for each stock i $Mom_{T,i}$ // two different kernels
2: Sort the stocks in terms of $Mom_{T,i}$ and assign $rank_i$
3: For $i \in S_T$:
4: If $(rank_i \leq N)$ & $(Mom_{T,i} \geq Mom_{min})$:
5: Select i
6: Return: set of selected i

i is given by

$$Mom_{T,i} = \frac{\sum_{k=1}^{T} R_{T-k,i}}{T} \tag{8}$$

After acquiring the $Mom_{T,i}$ for each stock i, we filter screen out the valuable stocks in two steps:

- Setting the rank N, we select the stocks with the highest N momentum;
- We check the momentum of the sorted stocks and abandon the stock that does not hit the minimum requirement.

For code, please check github repository (links attached in the end): basic_risk_budgeting.ipynb

#### 4.3.2 Momentum filter with Bayesian kernel

The MF with Bayesian kernel inherit the estimating result of static BPE as the momentum. Therefore, the momentum of each stock i is given by

$$Mom_{T,i} = \mu_{i,T}^{\alpha} \tag{9}$$

With the momentum calculated, we apply the same steps to filter the stock pool as steps in MF with smoothing kernel.

For code, please check github repository (links attached in the end): Bayesian_Portfolio.ipynb model D.

### 4.4 Risk-Budgeting Model

The RBM is a modified version of the Markowitz model, in which we use the Risk Contribution(RC) as the measure of risk, instead of the sample variance of excess return. After years of discussion, we are assured that the Risk Contribution represents the expected contribution to potential losses of a portfolio, and the additivity with adaptive weights of Risk Contribution makes it more practical than the classical Markowitz model[5]. The Risk Contribution is defined as followed:

Given the portfolio consisting N assets, the weight vector is

$$\mathbf{w} = (w_1, w_2, \cdots, w_N)$$

and the portfolio risk, which is the sample variance in the project

$$R(w) = \mathbf{w}^\top \mathbf{\Sigma} \mathbf{w} \tag{10}$$

, where $\mathbf{\Sigma}$ is the covariance matrix of N assets. The Risk Contribution for the stock i is

$$RC_i(\mathbf{w}) = w_i \frac{\partial R(\mathbf{w})}{\partial w_i} = \frac{w_i (\mathbf{\Sigma} \mathbf{w})_i}{(\mathbf{w}^\top \mathbf{\Sigma} \mathbf{w})^{\frac{1}{2}}} \tag{11}$$

. Then, the objective function is the weighted sum of $RC_i(w),\ i = 1, 2, \cdots, N$. In the project, we use the equally weighted sum, so the objective function is

$$\min_{\mathbf{w}} \sum_{i=1}^{N} RC_i(\mathbf{w}) \tag{12}$$

In order to make the optimization problem practical and suitable for the Chinese market, we set the constraints as followed:

1. We need to limit the upper bound of the portfolio volatility, measured by the standard deviation.

$$(\mathbf{w}^\top \mathbf{\Sigma} \mathbf{w})^{\frac{1}{2}} \leq STD_{upper\ bound} \tag{13}$$

2. We need to set the least acceptable portfolio return.

$$\mathbf{w}^\top \mathbf{R} \geq R_{lower\ bound} \tag{14}$$

3. We need to qualify the Risk Contribution and the Proportion of Risk Contribution of each stock in an acceptable range.

$$RC_{lower\ bound,i} \leq \frac{w_i(\mathbf{\Sigma w})_i}{(\mathbf{w}^\top \mathbf{\Sigma} \mathbf{w})^{\frac{1}{2}}} \leq RC_{upper\ bound,i} \tag{15}$$

$$PRC_{lower\ bound,i} \leq \frac{w_i(\mathbf{\Sigma w})_i}{\mathbf{w}^\top \mathbf{\Sigma} \mathbf{w}} \leq PRC_{upper\ bound,i} \tag{16}$$

4. We regulate the summation of all weights to 1, and all weights must be non-negative to avoid taking the leverage.

$$\sum_{i=1}^{N} w_i = 1 \tag{17}$$

$$0 \leq w_i \leq 1 \quad \forall i = 1, 2, \cdots, N \tag{18}$$

For code, please check github repository (links attached in the end): ./utils/mean_variance.py

# 5 Result & Analysis

## 5.1 Result

### 5.1.1 Measurement

We use annual return, annual volatility, accumulated return, Sharp Ratio, maximum draw down and Calmar Ratio to analyze the performance of our portfolio.

Sharp Ratio

$$SR = \frac{R - r_f}{\sigma} \tag{19}$$

where $R$ refers to the annual return, $r_f$ refers to the annual risk free rate and $\sigma$ refers to the annual volatility.

Calmar Ratio

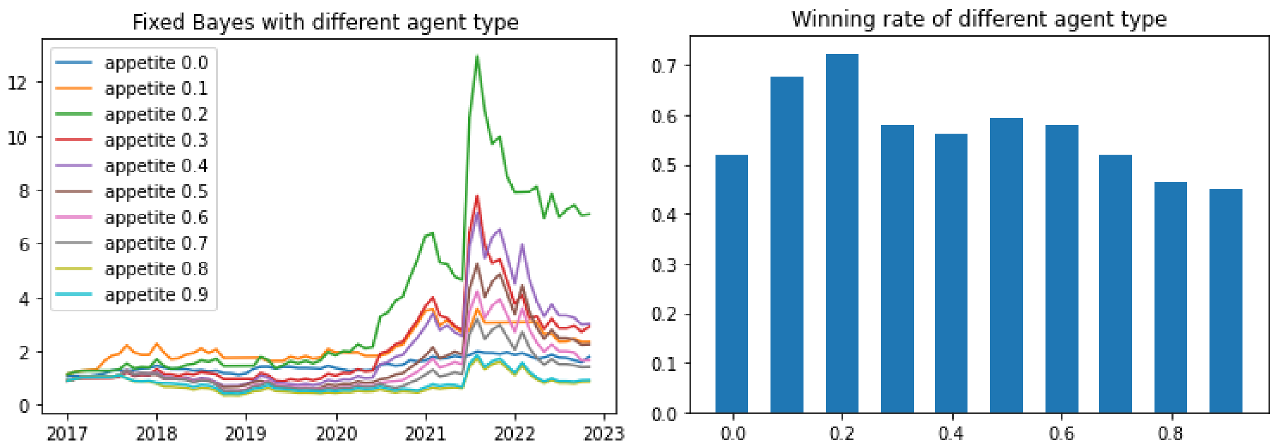
$$CR = \frac{R}{mdd} \tag{20}$$

where $R$ refers to the annual return and $mdd$ refers to the maximum draw down.

In the later comparison, the benchmark model is chosen as the equal weighted portfolio among the same stock pool.

### 5.1.2 Static risk-preference model

As described in the method, we fixed the investor's risk-averse type in the static risk preference. We tried several risk-averse type and below are the results. The smaller agent type refers to more risk averse type and higher agent type refers to more risk seeking type.




(a) Accumulated return

(b) Winning rate

Figure 2

In Figure2, the left plot is the accumulated return and the right plot is the winning rate plot. In general, we could conclude that all of them exhibits similar behavior over time. They excellently grab one chances during the year 2021 and rapidly increase. After that, they all suffer from a large scale of draw down. Comparatively, we can see that investors with more risk averse tend to behave better than the investors who are more risk seeking. Also the winning rate, which measures the times of right prediction, tend to be relatively high among the risk-averse investors. More specifically, the risk appetite 0.2 outperform other appetites with annual return 39.25%. During the growth year 2020 and 2021, it exhibits significant profit-making ability. During the recession year, it exhibits more robustness facing the drop of whole stocks market.

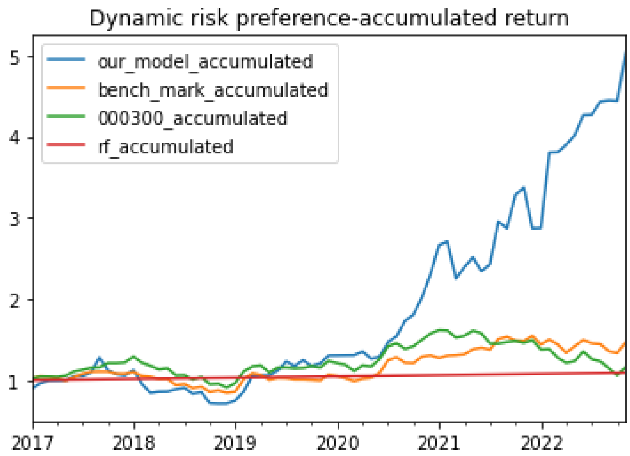
However, despite the robustness showed by fixed Bayes with appetite 0.2, it still have a large draw down (46.47%) and therefore a large volatility(10.69) due to the high influences from politics and macroeconomics of Chinese stock market.

### 5.1.3 Dynamic risk-preference model

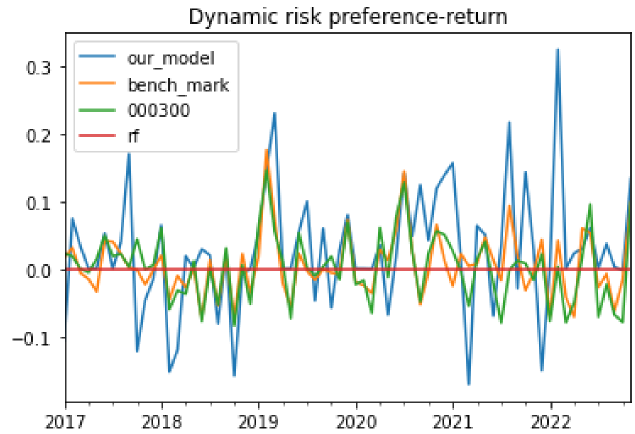
As described in the method, we make the investor's risk-averse type in the dynamic risk preference. That is, the investor's risk-averse type becomes adaptive to the market situation, which further stimulates the real investment environment.

In general(Figure 3), this time-variance risk-preference model tend to perform increasingly better. Surprisingly, this model may fix the disadvantage from the fixed preference model. Unlike the unsatisfying performance of fixed Bayesian model in 2022, this model also has good performance with annual return 31.4% in the recession year. Thus, small maximum draw down with 19%. Also, it has the Sharp Ratio of 0.15 and Calmar ratio of 15.6874.

We believe that the improvement of the maximum draw down is the effect of our dynamic risk preference method since this is the only factor that changed. More specifically, we further look into the distribution of appetite selection. We find that, for the most of the cases, our model still chooses the risk-averse appetite and only in a few cases, it will choose the risk-seeking appetite. This behaviour coincides with the phenomena in stock market that the stocks which turns out to have the best performance over the year has only a few significant increase during the year. More ideally, we believe that this improvement may shed light on the idea




(a) Accumulated return

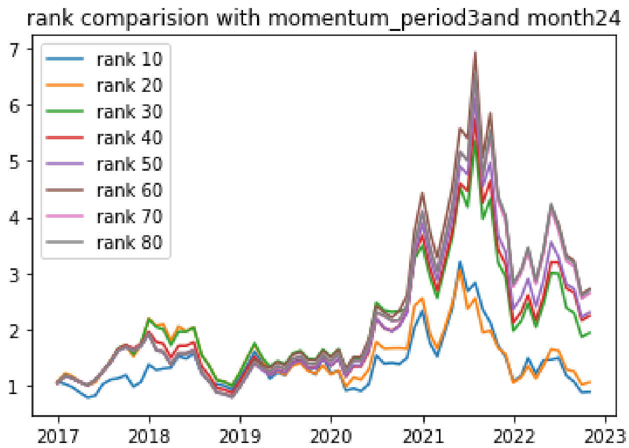



(b) Return

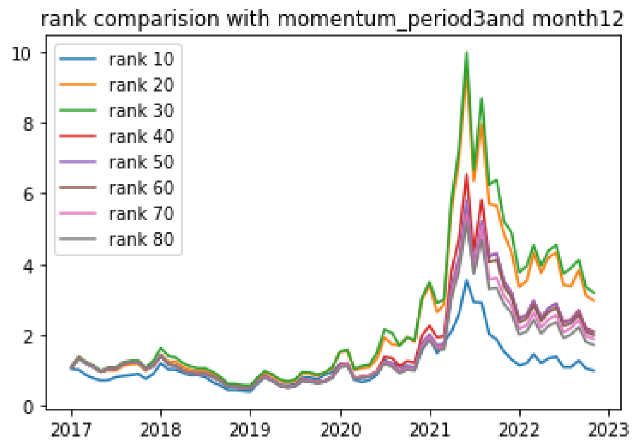
Figure 3

for investors that considering adjust risk appetite to generate profit during the recession and this model could be further improved if considering the macroeconomic factors.

#### 5.1.4 Momentum filter with smoothing kernel




(a) Accumulated return



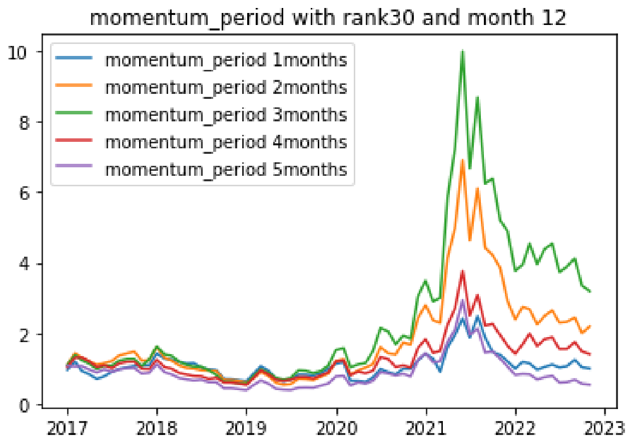

(b) Accumulated return

Figure 4

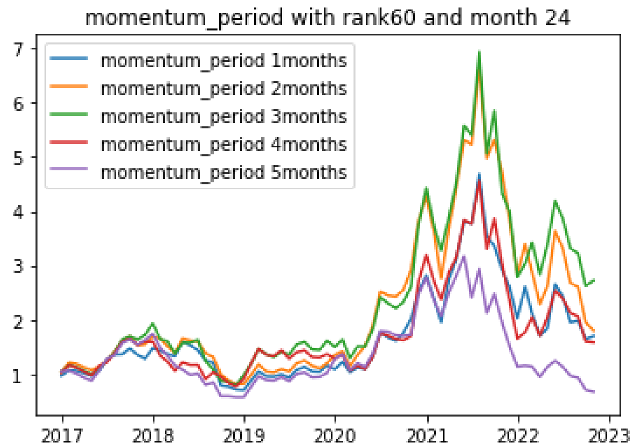
As described in the method, this filter has three parameters, namely time range of data, momentum period and the rank in momentum.

Most plots of rank behaves like the plot in the left, that the rank around 60 (i.e. using momentum to select the top 60 stocks) tend to performs the best. However, things are different when the time range of data change to one year. If momentum selection is based on one year time range, the rank 20(i.e. using momentum to select the top 20 stocks) tend to outperform other choices of rank. Surprisingly, the rank between 20 to 30 of one year time range data with momentum period3 outperforms other choices of parameters in this model largely.

This mainly could be explained by the short and long term momentum. When choosing a short time range(i.e. one year), the short term momentum tend to have strong effects and better prediction ability. Moreover, because of the short time, stock performance may fluctuate in a large scale therefore only the top stock's strong momentum effect lasts. However, in the large time range, the stocks are selected based on their long time performance, which means the result will be more robust. Therefore, large number of rank better improve the performance.




(a) Accumulated return



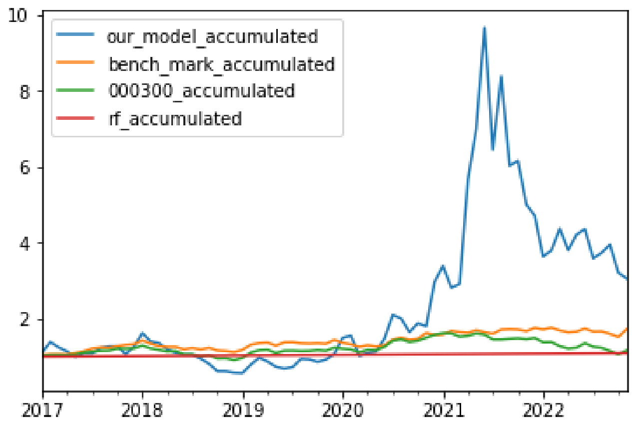

(b) Accumulated return

Figure 5

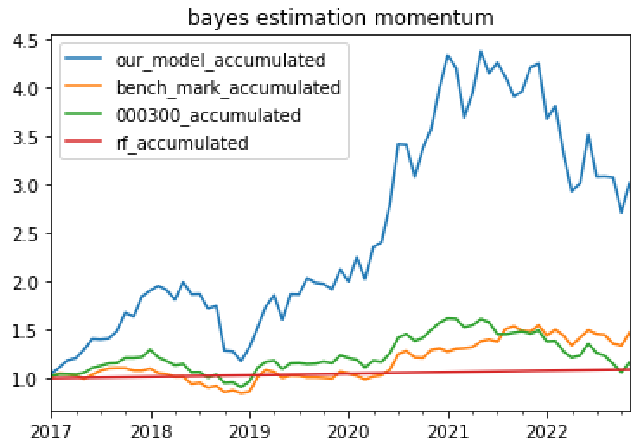
The performance behaviors of different momentum periods are similar and the parameter momentum period 3 months outperforms others in all the test.(Figure 5)

Among all the parameter testing plot, they have similar tendencies. They experience the waiting time from 2017-2019, the boosting time 2021 and draw down later. Here we choose the momentum filter smoothing kernel model with one year data range, three month momentum period and rank 30. This model results in the annual return of 20.78% but a large volatility of 7.43 and a small Sharp Ratio. (Figure 6 (a))

#### 5.1.5 Momentum filter with Bayes kernel




(a) momentum filter with smoothing kernel




(b) momentum filter with Bayes kernel

Figure 6

As described in the method, instead of directly using past historical data, we use the estimated monthly data to do the momentum selection. As can be seen in the plot(Figure 6 (b), the momentum filter with Bayes kernel tend to be more robust and performs better during the recession time. This model results in the annual return of 24.6% and a relatively small volatility of 3.35 comparing with the smoothing kernel model. It also has a 0.17 Sharp Ratio and 7.67 Calmar Ratio.

This is mainly because Bayes kernel, which uses the current month's price estimation to do the momentum selection, puts more weights on the current month compared with the smoothing kernel, which uses previous months' historical price to do the momentum selection. Therefore, it could react to the drop more quickly and wisely.

Comparing with the fixed preference Bayesian model, it is also interesting to notice that the Bayes kernel performances during 2017-19 and 2021-2022 are better than we directly use the estimated prices into the optimization problem. This result indicates that the Bayes estimation are correct in the general sense but may have some error if consider precisely and quantitatively. Furthermore, this result also shows that momentum selection with Bayes kernel could be implemented during the recession, which may shed light on the further research direction.

### 5.2 Advantage compared with traditional Markowitz model

Firstly, instead of directly using historical data in the optimization in Markowitz model, we use Bayesian methods to predict the excess return and use it to do optimization allocation or selection. Furthermore, compared to the traditional Bayes, we develop a time-variant risk-preference model, which could adaptively adjust the risk appetite according to the market situation. This makes our model more practical and reasonable to the real world. Secondly, instead of directly allocate all the stocks through optimization as Markowitz does, we add one more momentum filter to give higher condition restrictions on the stock before doing optimization allocation. This behaviour considers the whole market situation thus makes it more robust. Thirdly, the classical Markowitz model constructs the optimal portfolio by minimizing the risk, which is measured by the portfolio variance of excess return, which makes it only focus on the particular asset's risk but ignore the important of the investor's overall risk tolerance and investment objectives. We use the RC as the risk indicator since the RC-based portfolio outperforms the other indicators in measuring the percentage risk allocation in multi-asset portfolios. Additionally, we add necessary constraints like limit to short selling and leverage accordant with the Chinese Stock Market to make the investment strategy practical.

Here we doesn't use the time-variant risk-preference Bayes estimation to do the momentum selection because we think this would involve more behavioral finance analysis and are beyond our ability.

### 5.3 Future development

- We may use weekly data instead of monthly data to be more precise.
- We may try some normality tests to test the normality assumption. If not satisfied, some other assumptions (perhaps multivariate, or more fit distributed) should be taken.
- We may try adjusting hyper-parameters more objectively in our model, perhaps utilizing some other mathematical optimization methods.
- We may take the traction costs into consideration in our model to be more practical.
- We may try to involve some macroeconomic factors into our model to make it more considerate.

## 6 Conclusion

Our group propose to improve the mean-variance model in the aspects of Bayes estimation, momentum selection and risk-budgeting model. Furthermore, in the Bayes estimation, we also propose the time-variant risk-preference to be more adaptive to the market. We separately analysis them and combine their advantages together. We find that the momentum filter could help to catch the opportunity and the Bayes estimator can improve the robustness. We are also surprised to find that time-variance risk-preference Bayes could behave excellently during the recession, which may inspire us on further discussion.

# References

[1] David Ardia, Kris Boudt, and Giang Nguyen. Beyond risk-based portfolios: balancing performance and risk contributions in asset allocation. *Quantitative Finance*, 18(8):1249–1259, 2018.

[2] Doron Avramov and Guofu Zhou. Bayesian portfolio analysis. *Annu. Rev. Financ. Econ.*, 2(1):25–47, 2010.

[3] Narasimhan Jegadeesh and Sheridan Titman. Momentum. *Annu. Rev. Financ. Econ.*, 3(1):493–509, 2011.

[4] Harry M Markowitz and G Peter Todd. *Mean-variance analysis in portfolio choice and capital markets*, volume 66. John Wiley & Sons, 2000.

[5] Edward E Qian. On the financial interpretation of risk contribution: Risk budgets do add up. *Available at SSRN 684221*, 2005.

[6] Joanna Sokolowska and Piotr Makowiec. Risk preferences of individual investors: The role of dispositional tendencies and market trends. *Journal of Behavioral and Experimental Economics*, 71:67–78, 2017.

# 7 Appendix

The code is attached below. You may also refer to the link https://github.com/Linnore/STA4020-Projecto.git to find the code.

- Data Preparation

```python
import numpy as np
import pandas as pd


def get_excess_return_rates(R_df: pd.DataFrame, rf=None):
    if rf is None:
        rf = pd.read_csv("../data./Monthly_rf_Rate.csv",
                         index_col=0, parse_dates=True)
    R_excess_df = R_df - rf.values
    return R_excess_df
```

- Data Preparation Pipeline

```python
# %%
import os
import pandas as pd
import numpy as np

# %%
crt_dir = os.path.abspath("")
data_dir = os.path.abspath("data")
idx000300_dir = os.path.join(data_dir, "000300Weight_of_Constituent_Stock")

# %% [markdown]
# # Data Preparation

# %% [markdown]
# todo: description of our index data. why 000300? what did we do?

# %% [markdown]
# The historical daily constituent data of index 000300 from 2009-12 to current time we downloaded from CSMAR are seperated into 3 files due to the CSMAR's 5-year data maximum query policy. The following cell combine all data files and read them into RAM:

# %%
def genIDX_all(dir, namelist, output_name=None, force=False):
    if not os.path.lexists(os.path.join(dir, output_name)) or force:
        dflist = []
        for name in namelist:
            file = os.path.join(dir, name)
            dflist.append(pd.read_csv(file, header=0, index_col=1,
            parse_dates=True))
        df = pd.concat(dflist)
        df.to_csv(os.path.join(dir, output_name))
```

```
        return df
    else:
        return pd.read_csv(os.path.join(dir, output_name), header=0, index_col=0, parse_dates=True)

# %%
idx_filelist = list(filter(lambda file: file.startswith("IDX_Smprat_"), os.listdir(idx000300_dir)))
idx_filelist

# %%
df = genIDX_all(idx000300_dir, idx_filelist, 'All_IDX_Smprat.csv')
df

# %% [markdown]
# **Enable the following cell if you need to regenerate the combined dataset:**

# %%
# df = genIDX_all(idx000300_dir, idx_filelist, 'All_IDX_Smprat.csv', force=True)

# %% [markdown]
# **Select the stocks that are in 000300 portfolio during 2019-12-01 to 2022-12-08**

# %%
numOfDays = df.index.unique().size
stock_mask = df.groupby("Stkcd")["Indexcd"].count() >0#==numOfDays
stock_list = stock_mask.index[stock_mask]
stock_list = stock_list[(680000>stock_list)&(stock_list>=600000)].values
print(stock_mask.index)

# %% [markdown]
# **Now we have obtained the list of stocks of interest. Next step is to obtain the monthly return rates of these stocks:**

# %%
TRD_df = pd.read_csv(os.path.join(data_dir, "TRD_Mnth.csv"),
                     header=0, index_col=1, parse_dates=True)
stock_dict = stock_mask.to_dict()
TRD_df = TRD_df[TRD_df['Stkcd'].apply(lambda x: stock_dict.get(x, False))]


# %% [markdown]
# **Store the monthly return rates of our selected stockes:**

# %%
```

```python
R_df_list = []
for stock in stock_list:
    stock_df = TRD_df[TRD_df['Stkcd'] == stock]
    tmp = pd.DataFrame(stock_df['Mretwd'], index=stock_df.index)
    tmp.columns = [stock]
    R_df_list.append(tmp)


# %%
R_df = pd.concat(R_df_list, axis=1)
#R_df = R_df.fillna(-999.0)
R_df.to_csv(os.path.join(data_dir, "Monthly_Return_Rates.csv"))
R_df

# %%
rf = pd.read_csv(os.path.join(data_dir, "Monthly_rf_Rates.csv"), index_col=0,
parse_dates=True)
R_excess_df = R_df - rf.values
R_excess_df.to_csv(os.path.join(data_dir, "Monthly_Excess_Return_Rates.csv"))
R_excess_df


```

- Bayes estimator

```python
from scipy.stats import norm
import pandas as pd
import talib as ta
#crt_dir = os.path.abspath("")
#data_dir = os.path.abspath("data")
agent_list = [.05, .10, .20, .50, .80, .90]
agent_list2 = [0.3,0.4,0.6,0.7]
#path = './data/Monthly_Excess_Return_Rates.csv'

def data_load(path):
    '''
    Input:
    path of the data (Monthly_Excess_Return_Rates.csv)
    Output: df
    '''
    all_data = pd.read_csv(path)
    return all_data

def bayesFormula(returnR, var, a, b):
    n = len(returnR)
    coeff = b**2/(var+n*b**2)
    sigma2 = coeff*var
```

```python
    miu = (1-n*coeff)*a+coeff*sum(returnR)
    return miu, sigma2

def bayesPredict(stocks, agent_type, batch_size=9, relaxCoeff=8,dynamic=False):
    '''
    This function is used to predict "next-month" return.
    The prior and likelihood are both NORMAL distribution model.
    Prior model has informative parameter settings (based on the stock history)
    its MEAN = Historical mean rate of return; its VARIANCE =  relaxCoeff * Sample VAR
    This is a meaningful setting, if we believe stock market follows a MEAN-REVERTING process.

    Input:
    stocks df: monthly excess return from 2010 to 2022
    agent_type: float in (0,1); near 0 -> more conservative for stock return; near 1 -> more aggressive
    batch_size: we use this number of months as Bayes observations
    relaxCoeff: int; bigger -> prediction emphasizes more on the recent batch; smaller -> more stable around historical mean
    Output: predicted df
    '''
    if dynamic == True:
        return bayesPredict_dynamic(stocks,agent_type = 0.3,batch_size=9, relaxCoeff=8,alpha = 1/3)
    else:
        # Suppose our rolling-horizon prediction starts at:
        starting_date = '2017-01-01'
        # We will use sample variance before this date as "b" in bayes formula.
        date_list = list(stocks['Trdmnt'])
        starting_idx = date_list.index(starting_date)
        stock_list = []

        for name in stocks:
            if name.isdigit():
                stock_list.append(name)
        predict_df = pd.DataFrame(stocks, columns=stock_list)
        timeLine = pd.to_datetime(stocks['Trdmnt'])
        predict_df.index = timeLine

        for name in stock_list:
            returnR = stocks[name]
            hist_R = returnR[:starting_idx]
            B = hist_R.var()
            A = hist_R.mean()
            # For every month after starting date, we use Bayes predict.
```

```
            # We only use "batch_size" monthly data for prediction this step.
            # print('History sample mean:', A)
            for seq_num in range(starting_idx, len(date_list)):
                ref_batch = returnR[seq_num-batch_size:seq_num]
                good = ref_batch[ref_batch==ref_batch]
                var = good.var()
                miu, sigma2 = bayesFormula(returnR=good, var=var, a=A, b=relaxCoeff*B)
                percentile_R = norm.ppf(q=agent_type, loc=miu, scale=(sigma2**0.5))
                if percentile_R < -1 or percentile_R > 1:
                    print('Strange Prediction!')
                predict_df.loc[timeLine[seq_num], name] = percentile_R

                #print(date_list[seq_num], 'Bayes predicted \mu', round(miu,5), \
                #    'Actual return is', round(returnR[seq_num],5), \
                #    'Mean return in last batch', round(good.mean(),5))
            # print(len(predict_dict[name]), (len(date_list)-starting_idx))

        #print(predict_df.tail())
        #print(stocks.tail())
        return predict_df


def agent_choosing(x):
    x = x[0]
    if x < percentile[0]:
        return agent_list[0]
    elif x < percentile[1]:
        return agent_list[1]
    elif x<percentile[2]:
        return agent_list[2]
    elif x<percentile[3]:
        return agent_list[3]
    #else: return agent_list[4]

def dynamic_agent(index_return,alpha):
    agent_list = [.10, .30, .50, .70, .90]
    index_return = index_return.ewm(alpha = 1/3).mean()
    global percentile
    percentile = []
    for i in agent_list2:
        tmp = index_return.quantile(i).values[0]
        percentile.append(tmp)
    agent = index_return.apply(lambda row: agent_choosing(row.values),axis=1)
    agent.to_csv('agent_distribution.csv')
    return agent
```

```python


def bayesPredict_dynamic(stocks,agent_type = 0.3,batch_size=9, relaxCoeff=8,alpha = 1/3):
    '''
    This function is used to predict "next-month" return.
    The prior and likelihood are both NORMAL distribution model.
    Prior model has informative parameter settings (based on the stock history)
    its MEAN = Historical mean rate of return; its VARIANCE =  relaxCoeff * Sample VAR
    This is a meaningful setting, if we believe stock market follows a MEAN-REVERTING process.
    Input:
    stocks df: monthly excess return from 2010 to 2022
    agent_type: float in (0,1); near 0 -> more conservative for stock return; near 1 -> more aggressive
    batch_size: we use this number of months as Bayes observations
    relaxCoeff: int; bigger -> prediction emphasizes more on the recent batch; smaller -> more stable around historical mean
    Output: predicted df
    '''
    index_return = pd.read_csv("data/000300.csv",index_col=1,parse_dates=True)
    index_return=index_return.drop(columns=['Indexcd'])
    index_return = index_return.diff(1).fillna(0)
    # Suppose our rolling-horizon prediction starts at:
    starting_date = '2017-01-01'
    # We will use sample variance before this date as "b" in bayes formula.
    date_list = list(stocks['Trdmnt'])
    starting_idx = date_list.index(starting_date)
    stock_list = []
    # agent_type_list = []



    for name in stocks:
        if name.isdigit():
            stock_list.append(name)
    predict_df = pd.DataFrame(stocks, columns=stock_list)
    timeLine = pd.to_datetime(stocks['Trdmnt'])
    predict_df.index = timeLine
    agent = dynamic_agent(index_return,alpha)
    for name in stock_list:
        returnR = stocks[name]
        hist_R = returnR[:starting_idx]
        B = hist_R.var()
        A = hist_R.mean()
```

```
        # For every month after starting date, we use Bayes predict.
        # We only use "batch_size" monthly data for prediction this step.
        # print('History sample mean:', A)
        for seq_num in range(starting_idx, len(date_list)):
            ref_batch = returnR[seq_num-batch_size:seq_num]
            good = ref_batch[ref_batch==ref_batch]
            var = good.var()
            miu, sigma2 = bayesFormula(returnR=good, var=var, a=A, b=relaxCoeff*B)
            percentile_R = norm.ppf(q=agent[seq_num], loc=miu, scale=(sigma2**0.5))
            if percentile_R < -1 or percentile_R > 1:
                print('Strange Prediction!')
            predict_df.loc[timeLine[seq_num], name] = percentile_R

            #print(date_list[seq_num], 'Bayes predicted \mu', round(miu,5), \
            #     'Actual return is', round(returnR[seq_num],5), \
            #     'Mean return in last batch', round(good.mean(),5))
        # print(len(predict_dict[name]), (len(date_list)-starting_idx))

    #print(predict_df.tail())
    #print(stocks.tail())
    return predict_df
```

- Momentum with smoothing

```
# %%
from scipy.optimize import minimize
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from utils import test_result
from utils.mean_variance import *
import os

%load_ext autoreload
%autoreload 2


# %% [markdown]
# # A test for the risk-budgeting model

# %% [markdown]
# ## Data/Stock choosing:
# 1. 2010-2022300-52
# 2. Monthly period from 2010 - 2022.
#

# %%
```

```python
crt_dir = os.path.abspath("")
data_dir = os.path.abspath("data")


# %% [markdown]
# Read in monthly return rates the 52 stocks and risk free rates from 2009-12 to 2022-12.

# %%
R_df = pd.read_csv(os.path.join(
    data_dir, "Monthly_Return_Rates.csv"), index_col=0, parse_dates=True)
R_df = R_df.fillna(0)
rf_df = pd.read_csv(os.path.join(
    data_dir, "Monthly_rf_Rates.csv"), index_col=0, parse_dates=True)
rf_df = rf_df.fillna(0)
rf_df = rf_df


# %% [markdown]
# Compute monthly **excess** return rates:

# %%
from utils import data_prep
R_excess_df = data_prep.get_excess_return_rates(R_df, rf_df)


# %% [markdown]
# CSI300 monthly return:

# %%
CSI300_df = pd.read_csv(os.path.join(
    data_dir, "000300.csv"), index_col=1, parse_dates=True)['Idxrtn']


# %% [markdown]
# # Construct Portfolio

# %% [markdown]
# * Consider the optimization problem:
#
# \begin{aligned}
# \min_{w} & \quad \sum_{i=1}^N RC_i(w)\\
# \textrm{s.t.} & \quad  \boldsymbol{1}^Tw \leq 1.5\\
#                     & \quad w  \geq \boldsymbol{0}\\
# \end{aligned}
#
```

```
#
# * Following the logic from sta4020 asg7. Starting from the end of 2015, at the end of every month, use the historical asset returns (from 2011/1 to the end of that month).

# %% [markdown]
# **Estimated Excess Return after 2017; And corresponding portfolio.**

# %% [markdown]
# **Please note that there are clear descriptions for the functions and constraint, please remember to check before implementing the models**

# %% [markdown]
# To add other constraints, please refer to `cons_*()` in `mean_variance.py`. For example, we implemented the box constraints for $RC_i(w)$: $$\boldsymbol{L} \leq RC(w) \leq \boldsymbol{U}.$$
#
# You can add the box constraints by adding `cons_sum_weight_lower_bound(L)` and `cons_sum_weight_upper_bound(U)` to the `constraint_list`. Note that L and U are N-dimension vector, given N assets.

# %%
R_excess_df_i = R_excess_df.iloc[-24:,:]
numOfDays = R_excess_df_i.index.unique().size
R_excess_df_i
numOfDays

# %%
sum_weight = 1.5
constraint_list = [
    cons_non_negative_weight(), cons_sum_weight_upper_bound(sum_weight)]
# mp = 30, rank = 30,month = 12
R_excess_hat, w_hat = portfolio_construction(
    momentum_period=3, rank=30, R_excess_df=R_excess_df,rf_df=rf_df,
    momentum_atLeast=0.05, num_atLeast=1,month = 12, objective=obj_Exp_minus_RC,
    constraints=constraint_list)


# %% [markdown]
# **Compute net excess return; and recover net return by adding risk-free return.**

# %%
R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
R_net_hat=R_net_hat-(sum_weight-1)*rf_df[rf_df.index >=
pd.Timestamp("2017")].values.reshape(-1)
```

```python
R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                            'return'], index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index)
test_result.calculate_result(
    R_net_hat_df, rf_df[rf_df.index >= pd.Timestamp("2017")])


# %% [markdown]
# #### Todo:

# %%

accumalte_list = []
def rank_function(momentum,month):
    for i in range(10,90,10):#i is the 52-i stock we decide to keep according to momentum
        R_excess_hat, w_hat = portfolio_construction(momentum_period=momentum, rank=i, R_excess_df=R_excess_df, rf_df=rf_df,momentum_atLeast=0.05, num_atLeast=1,month = month, objective=obj_Exp_minus_RC, constraints=constraint_list)
        R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
        R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                            'return'], index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index)

        result_df = test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >= pd.Timestamp("2017")])
        accumalte_list.append(result_df['accu_return'].values[0])
        p_cum = (R_net_hat_df['return'] + 1).cumprod()
        plt.plot(p_cum,label = 'rank '+str(i))
    plt.legend()
    plt.title('rank comparision with momentum_period '+str(momentum )+' and month '+str(month))
    plt.show()


# %%
rank_function(4,48)
rank_function(3,24)
rank_function(3,12)
rank_function(4,60)


# %%
accumalte_list = []
```

```python
def momentum_function(rank,month):
    for j in range(1,6):#i is the 52-i stock we decide to keep according to
    momentum
            R_excess_hat, w_hat = portfolio_construction(momentum_period=j,
            rank=rank, R_excess_df=R_excess_df, rf_df=rf_df,momentum_atLeast=0.05,
            num_atLeast=1, month=month, objective=obj_Exp_minus_RC,
            constraints=constraint_list)
            R_net_hat = np.sum(R_excess_hat, axis=1) + \
        rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
            R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                                'return'], index=CSI300_df[CSI300_df.index >=
                                pd.Timestamp("2017")].index)

            result_df = test_result.calculate_result(R_net_hat_df,
            rf_df[rf_df.index >= pd.Timestamp("2017")])
            accumalte_list.append(result_df['accu_return'].values[0])
            p_cum = (R_net_hat_df['return'] + 1).cumprod()
            plt.plot(p_cum,label = 'momentum_period '+str(j)+'months')
    plt.legend()
    plt.title('momentum_period comparision')
    plt.show()

# %%
momentum_function(30,12)
momentum_function(60,24)
momentum_function(40,48)
momentum_function(60,48)

# %% [markdown]
# **Construct equal weighted portfolio**

# %%
ew_rets = pd.DataFrame(np.sum(
    1.0*R_df[R_df.index >= pd.Timestamp("2017")]/R_df.shape[1], axis=1),
    columns=['return'])
rf_rets = pd.DataFrame(
    rf_df[rf_df.index.year>=2017].values, columns=['return'],
    index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index
)

# %% [markdown]
# **Plot the portfolio cumulative returns**

# %%
p_cum = (R_net_hat_df['return'] + 1).cumprod()
ew_cum = (ew_rets['return'] + 1).cumprod()
```

```
CSI300_cumrets = (CSI300_df[CSI300_df.index.year >= 2017] + 1).cumprod()
rf_cumrets = (rf_rets+1).cumprod()
pd.concat([p_cum, ew_cum, CSI300_cumrets, rf_cumrets], axis=1).plot()
plt.legend(['our_model_accumulated',
           'bench_mark_accumulated', '000300_accumulated', 'rf_accumulated'])
plt.show()


# %% [markdown]
# * maximize the sharp ratio and the formula derivation: this repository may helps
#
# https://github.com/PaiViji/PythonFinance--RiskBudgeted-Portfolio-Construction.git
#
# * we can also use this repository to check the quality of our work:
#
# https://github.com/jcrichard/pyrb.git

# %%
tmp_df = pd.read_csv('./data/Monthly_Excess_Return_Rates.csv',index_col=0,parse_dates=True)
df_2021 = tmp_df[tmp_df.index.year == 2020]
df_2022 = tmp_df[tmp_df.index.year == 2022]

# %%
f = df_2021.describe()
f.loc['mean'].mean()

# %%
e = df_2022.describe()
e.loc['mean'].mean()

```

- Momentum with bayes, fixed preference and dynamic preference

```
# %%
from utils.bayes import bayesPredict,bayesPredict_dynamic
from scipy.optimize import minimize
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from utils import test_result
from utils.mean_variance import *
import os
%load_ext autoreload
%autoreload 2

```

```python
crt_dir = os.path.abspath("")
data_dir = os.path.abspath("data")
file1 = os.path.join(data_dir, "000300.csv")
file2 = os.path.join(data_dir, "Monthly_Excess_Return_Rates_bayes.csv")
CSI300_df = pd.read_csv(file1, index_col=1, parse_dates=True)['Idxrtn']
excessR_df = pd.read_csv(file2)
bayesR_df = bayesPredict(excessR_df, agent_type=0.3,dynamic=True)


# %% [markdown]
# # A fixed Bayes estimator

# %%
#excessR_df.index = pd.to_datetime(excessR_df['Trdmnt'])
result_list = []

for i in range(10):
    excessR_df = pd.read_csv(file2)
    bayesR_df = bayesPredict(excessR_df, agent_type=i/10)
    excessR_df = pd.read_csv(file2, index_col=0, parse_dates=True)
    excessR_df = excessR_df.fillna(0)
    rf_df = pd.read_csv(os.path.join(data_dir, "Monthly_rf_Rates.csv"), index_col=0, parse_dates=True)
    rf_df = rf_df.fillna(0)
    R_df = pd.read_csv(os.path.join(data_dir, "Monthly_Return_Rates.csv"), index_col=0, parse_dates=True)
    R_df = R_df.fillna(0)
    constraint_list = [cons_non_negative_weight(), cons_sum_weight_upper_bound(1)]
    R_excess_hat, w_hat = portfolio2(Bayes_df=bayesR_df, R_excess_df=excessR_df, momentum_period=2, rank=20, momentum_atLeast=0.05, num_atLeast=2, objective=obj_Exp, constraints=constraint_list)
    R_net_hat = np.sum(R_excess_hat, axis=1) + \
        rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
    R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                                'return'], index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index)

    result_list.append(test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >= pd.Timestamp("2017")]))
    p_cum = (R_net_hat_df['return'] + 1).cumprod()
    plt.plot(p_cum,label = 'appetite '+str(i/10))
plt.legend()
plt.title('Fixed Bayes with different agent type ')

plt.show()

```

```python
# %%
winning_rate_list = []
for i in range(len(result_list)):
    winning_rate_list.append(result_list[i]['winning_rate'].values[0])

winning_rate_list[2] = 0.7233
winning_rate_list[1] = 0.6766

# %%
fig,ax = plt.subplots()
x = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
ax.bar(x,winning_rate_list,width=0.06)
ax.set_title('Winning rate of different agent type')
plt.show()

# %%
excessR_df = pd.read_csv(file2)
bayesR_df = bayesPredict(excessR_df, agent_type=0.2)
excessR_df = pd.read_csv(file2, index_col=0, parse_dates=True)
excessR_df = excessR_df.fillna(0)
rf_df = pd.read_csv(os.path.join(data_dir, "Monthly_rf_Rates.csv"), index_col=0,
parse_dates=True)
rf_df = rf_df.fillna(0)
R_df = pd.read_csv(os.path.join(data_dir, "Monthly_Return_Rates.csv"), index_col=0,
parse_dates=True)
R_df = R_df.fillna(0)
constraint_list = [cons_non_negative_weight(), cons_sum_weight_upper_bound(1)]
R_excess_hat, w_hat = portfolio2(Bayes_df=bayesR_df, R_excess_df=excessR_df,
momentum_period=2, rank=20, momentum_atLeast=0.05, num_atLeast=2,
objective=obj_Exp, constraints=constraint_list)
R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                            'return'], index=CSI300_df[CSI300_df.index >=
                            pd.Timestamp("2017")].index)
test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >=
pd.Timestamp("2017")])

# %%
ew_rets = pd.DataFrame(np.sum(
    1.0*R_df[R_df.index >= pd.Timestamp("2017")]/R_df.shape[1], axis=1),
    columns=['return'])
rf_rets = pd.DataFrame(
    rf_df[rf_df.index.year>=2017].values, columns=['return'],
    index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index
)
```

```
p_cum = (R_net_hat_df['return'] + 1).cumprod()
ew_cum = (ew_rets['return'] + 1).cumprod()
CSI300_cumrets = (CSI300_df[CSI300_df.index.year >= 2017] + 1).cumprod()
rf_cumrets = (rf_rets+1).cumprod()
pd.concat([p_cum, ew_cum, CSI300_cumrets, rf_cumrets], axis=1).plot()
plt.legend(['our_model_accumulated',
           'bench_mark_accumulated', '000300_accumulated', 'rf_accumulated'])
plt.title('Bayesian estimate with fixed appetite 0.2')
plt.show()

# %% [markdown]
# # B Dynamic Bayesian estimator

# %%
#excessR_df.index = pd.to_datetime(excessR_df['Trdmnt'])
excessR_df = pd.read_csv(file2)
bayesR_df = bayesPredict(excessR_df, agent_type=0.3,dynamic=True)
excessR_df = pd.read_csv(file2, index_col=0, parse_dates=True)
excessR_df = excessR_df.fillna(0)
rf_df = pd.read_csv(os.path.join(data_dir, "Monthly_rf_Rates.csv"), index_col=0,
parse_dates=True)
rf_df = rf_df.fillna(0)
R_df = pd.read_csv(os.path.join(data_dir, "Monthly_Return_Rates.csv"), index_col=0,
parse_dates=True)
R_df = R_df.fillna(0)
constraint_list = [cons_non_negative_weight(), cons_sum_weight_upper_bound(1)]
R_excess_hat, w_hat = portfolio2(Bayes_df=bayesR_df, R_excess_df=excessR_df,
momentum_period=2, rank=20, momentum_atLeast=0.05, num_atLeast=2,
objective=obj_Exp, constraints=constraint_list)
R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                            'return'], index=CSI300_df[CSI300_df.index >=
                            pd.Timestamp("2017")].index)

f=test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >=
pd.Timestamp("2017")])

# %%
ew_rets = pd.DataFrame(np.sum(
    1.0*R_df[R_df.index >= pd.Timestamp("2017")]/R_df.shape[1], axis=1),
    columns=['return'])
rf_rets = pd.DataFrame(
    rf_df[rf_df.index.year>=2017].values, columns=['return'],
    index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index
)
```

```python
p_cum = (R_net_hat_df['return'] + 1).cumprod()
ew_cum = (ew_rets['return'] + 1).cumprod()
CSI300_cumrets = (CSI300_df[CSI300_df.index.year >= 2017] + 1).cumprod()
rf_cumrets = (rf_rets+1).cumprod()

pd.concat([p_cum, ew_cum, CSI300_cumrets, rf_cumrets], axis=1).plot()
plt.legend(['our_model_accumulated',
            'bench_mark_accumulated', '000300_accumulated', 'rf_accumulated'])
plt.title('Dynamic risk preference-accumulated return')
plt.figure(figsize=(16,8))
plt.show()

# %%
pd.concat([R_net_hat_df['return'], ew_rets['return'], CSI300_df[CSI300_df.index.year >= 2017], rf_rets], axis=1).plot()
plt.legend(['our_model',
            'bench_mark', '000300', 'rf'])
plt.title('Dynamic risk preference-return')
plt.figure(figsize=(16,8))
plt.show()

# %% [markdown]
# # D Momentum with bayes kernel

# %%
excessR_df = pd.read_csv(file2)
bayesR_df = bayesPredict(excessR_df, agent_type=0.6)
excessR_df = pd.read_csv(file2, index_col=0, parse_dates=True)
excessR_df = excessR_df.fillna(0)
rf_df = pd.read_csv(os.path.join(data_dir, "Monthly_rf_Rates.csv"), index_col=0, parse_dates=True)
rf_df = rf_df.fillna(0)
R_df = pd.read_csv(os.path.join(data_dir, "Monthly_Return_Rates.csv"), index_col=0, parse_dates=True)
R_df = R_df.fillna(0)
constraint_list = [cons_non_negative_weight(), cons_sum_weight_upper_bound(1.5)]

R_excess_hat, w_hat=portfolio3(bayesR_df,21,excessR_df,0.003,1,pd.Timestamp('2017'),obj_Exp_minus_RC,constraint
R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                            'return'], index=CSI300_df[CSI300_df.index >=
                            pd.Timestamp("2017")].index)

```

```python
test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >= pd.Timestamp("2017")])

# %%
p_cum = (R_net_hat_df['return'] + 1).cumprod()
ew_cum = (ew_rets['return'] + 1).cumprod()
CSI300_cumrets = (CSI300_df[CSI300_df.index.year >= 2017] + 1).cumprod()
rf_cumrets = (rf_rets+1).cumprod()
pd.concat([p_cum, ew_cum, CSI300_cumrets, rf_cumrets], axis=1).plot()
plt.legend(['our_model_accumulated',
           'bench_mark_accumulated', '000300_accumulated', 'rf_accumulated'])
plt.title('bayes estimation momentum')
plt.show()



# %%
accumalte_list = []
def rank_function(least_momentum):
    for j in range(10,50,10):
        R_excess_hat, w_hat=portfolio3(bayesR_df,j,excessR_df,least_momentum,1,pd.Timestamp('2017'),obj_Exp_mi
        R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
        R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                            'return'], index=CSI300_df[CSI300_df.index >= pd.Timestamp("2017")].index)

        result_df = test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >= pd.Timestamp("2017")])
        accumalte_list.append(result_df['accu_return'].values[0])
        p_cum = (R_net_hat_df['return'] + 1).cumprod()
        plt.plot(p_cum,label = 'rank '+str(j))
    plt.legend()
    plt.title('rank comparison with least momentum'+str(least_momentum ))
    plt.show()

# %%
rank_function(0.003)
rank_function(0.005)

# %%
accumalte_list = []
def least_momentum_function(rank):
    for j in range(1,8):
```

```
        R_excess_hat,
        w_hat=portfolio3(bayesR_df,rank,excessR_df,j/1000,1,pd.Timestamp('2017'),obj_Exp_minus_R
        R_net_hat = np.sum(R_excess_hat, axis=1) + \
    rf_df[rf_df.index >= pd.Timestamp("2017")].values.reshape(-1,)
        R_net_hat_df = pd.DataFrame(R_net_hat, columns=[
                              'return'], index=CSI300_df[CSI300_df.index >=
                              pd.Timestamp("2017")].index)

        result_df = test_result.calculate_result(R_net_hat_df, rf_df[rf_df.index >=
        pd.Timestamp("2017")])
        accumalte_list.append(result_df['accu_return'].values[0])
        p_cum = (R_net_hat_df['return'] + 1).cumprod()
        plt.plot(p_cum,label = 'least momentum '+str(j/1000))
    plt.legend()
    plt.title('least momentum comparistion with rank'+str(rank ))
    plt.show()

# %%
least_momentum_function(20)


```

- test

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def calculate_result(R_excess_df,rf_df = 0):
    #
    R_excess_df['net_worth'] = 1*(1+R_excess_df['return']).cumprod(axis = 0)
    #
    accu_return = R_excess_df['net_worth'].iloc[-1]/1-1
    # 365
    annual_return = (1+accu_return)**(12/len(R_excess_df))-1
    # 365
    annual_vol = R_excess_df['net_worth'].std(ddof = 1)*np.sqrt(12)
    #rf = 0.01
    rf_df = 0.01
    sharpe_ratio = (annual_return-rf_df)/(annual_vol)
    #
    max_dd =
    ((R_excess_df['net_worth']-R_excess_df['net_worth'].cummax())/R_excess_df['net_worth'].cummax
    #
    wining_count = len(R_excess_df[R_excess_df['return'] > 0])/len(R_excess_df)
    #karmar ratio
    karmar = abs(accu_return/max_dd)
```

```
    result_df = pd.DataFrame()
    result_df['accu_return'] = [accu_return]
    result_df['annual_return'] = [annual_return]
    result_df['annual_vol'] = [annual_vol]
    result_df['sharpe_ratio'] = [sharpe_ratio]
    result_df['max_dd'] =[max_dd]
    result_df['winning_rate'] = [wining_count]
    result_df['karmar'] = [karmar]
    return result_df
```