Diversity-weighted Portfolio in Stochastic Portfolio Optimization: A closer look at its application and efficacy

Chenghao Zhu, Haoyang Shang, Yuhan Wang, Yuxuan Zhou

Boston University, Questrom School of Business

December 11, 2023

Abstract

In this paper, we provided a comprehensive analysis of diversity-weighted portfolios within Stochastic Portfolio Theory (SPT). We examined the application and effectiveness of Bayesian non-parametric methods, particularly in addressing challenges inherent in SPT, like market imperfections and model limitations. The study compared Frequentist and Bayesian parametric inference methods in portfolio optimization and explored the impact of negative parameter values on portfolio returns. Empirical analysis included the evaluation of portfolio performance using data from the S&P 500 index, assessing the merits of diversity-weighted portfolios with both positive and negative parameters.

1 Introduction

Stochastic Portfolio Theory, an alternative approach to portfolio selection developed by Robert Fernholz in 2002, aims to outperform the market index with probability one. Fernholz uses a 'master equation', a pathwise decomposition of relative performance, to avoid challenges encountered in classical portfolio optimization methods, such as explicit model postulation and calibration, as well as the (normative) no-arbitrage assumption.

However, Stochastics Portfolio Theory remains imperfect for wider adoption due to several remaining problems and limitations. Three of the most notable issues are: the difficulty of finding relative arbitrages under reasonable assumptions (inverse problem), ignorance of several market imperfections such as the possibility of default and transaction cost, and lack of adoption by factors other than market capitalization when exploiting market inefficiency.

The study "Stochastic Portfolio Theory: A Machine Learning Perspective"

by Yves-Laurent Kom Samo and Alexander Vervuurt, published in 2016, introduces a Bayesian non-parametric method to overcome challenges in Stochastic Portfolio Theory. The researchers explore a variety of investment strategies influenced by a function acting on a diverse set of trading characteristics, like market capitalization. This function is governed by a Gaussian process (GP) prior. They evaluate the likelihood of a strategy being 'exceptional' based on a performance metric chosen by the user (such as excess return compared to the market index or Sharpe ratio), and benchmarks that are considered 'exceptional'. The authors utilize Monte Carlo Markov Chain (MCMC) techniques to generate samples from the posterior of the GP that defines the 'exceptional' strategy.

In our project, we explored how authors used Bayesian Inference in Diversity-weighted Portfolios, to tackle the challenges of Stochastic Portfolio Theory. Moreover, we compared the performance of Frequentist parametric inference with that of the Bayesian parametric inference method when sampling posterior distribution. In the last part of the project, we also explored how a negative parameter p in a Diversity-weighted portfolio could drastically increase its return compared to a positive parameter p.

2 Diversity-weighted Portfolio

We provide a brief overview of SPT, outlining the broad category of market models that its findings apply to, the nature of the portfolio selection criterion, and the methods used to develop strategies that meet this requirement.

2.1 The Model of Stock Market

In SPT, the stock capitalizations are modelled as Itô processes. Namely, the dynamics of the n positive stock capitalization processes $X_i(\cdot)$, $i = 1, \ldots, n$

are described by the following system of SDEs

$$dX_i(t) = X_i(t)\left(b_i(t)dt + \sum_{\nu=1}^{d}\sigma_{i\nu}(t)dW_\nu(t)\right), \tag{1}$$

for $t \geq 0$ and $i = 1, \ldots, n$. Here, $W_1(\cdot), \ldots, W_d(\cdot)$ are independent standard Brownian motions with $d \geq n$, and $X_i(0) > 0$, $i = 1, \ldots, n$ are the initial capitalizations.

We assume all processes to be defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, and adapted to a filtration $\mathcal{F} = \{\mathcal{F}(t)\}_{0\leq t<\infty}$.

The rates of return $b_i(\cdot)$, $i = 1, \ldots, n$ and volatilities $\sigma(\cdot) = (\sigma_{i\nu}(\cdot))_{1\leq i\leq n, 1\leq\nu\leq d}$, are some unspecified $\mathcal{F}$ progressively measurable processes and are assumed to satisfy the integrability condition

$$\sum_{i=1}^{n}\int_0^T\left(|b_i(t)| + \sum_{\nu=1}^{d}(\sigma_{i\nu}(t))^2\right)dt < \infty \tag{2}$$

for all $T \in (0, \infty)$, and the non-degeneracy condition

$$\exists\epsilon > 0 \text{ s.t. } \xi^\top\sigma(t)\sigma^\top(t)\xi \geq \epsilon\|\xi\|^2, \forall\xi \in \mathbb{R}^n, t \geq 0 \tag{3}$$

2.2 Relative Arbitrage

It is now necessary for us to understand how to use portfolios to build and assess a portfolio's performance within this model framework. These are $\mathbb{R}^n$-valued and $\mathcal{F}$-progressively measurable processes $\pi(\cdot) = (\pi_1(\cdot), \ldots, \pi_n(\cdot))^T$, where $\pi_i(t)$ stands for the proportion of wealth invested in stock i at time t.

Long-only portfolios is what we used and there is no money market. These invest solely in the stocks; they take values in the closure $\overline{\Delta_+^n}$ of the set

$$\Delta_n^+ = \{x \in \mathbb{R}^n : x_1 + \ldots + x_n = 1, 0 < x_i < 1, i = 1, \ldots, n\} \quad (4)$$

Without losing generality, let us assume that each firm has one outstanding share. The related wealth process $V^\pi(\cdot)$ of an investor executing $\pi(\cdot)$ can be observed to evolve in the following way.

$$\frac{dV_\pi(t)}{V_\pi(t)} = \sum_{i=1}^{n} \pi_i(t) \frac{dX_i(t)}{X_i(t)}, \quad V^\pi(0) = 1 \quad (5)$$

Performance is primarily measured in SPT in relation to the market index. This is the wealth process $V^\mu(\cdot)$ that results from a buy-and-hold portfolio, given by the vector process $\mu(\cdot) = (\mu_1(\cdot), \ldots, \mu_n(\cdot))^T$

$$\mu_i(t) \frac{X_i(t)}{\sum_{j=1}^{n} X_j(t)} \quad (6)$$

Let $T > 0$. A strong relative arbitrage with respect to the market over the time-horizon $[0, T]$ is a portfolio $\pi(\cdot)$ such that

$$P(V^\pi(T) > V^\mu(T)) = 1 \quad (7)$$

Any portfolio that beats the market in the context of (7) is considered a relative arbitrage in SPT, and the degree of the beat is theoretically meaningless.

2.3 Functionally-generated Portfolios

Now we will introduce a particular type of portfolios named functionally-generated portfolios studied by Fernholz [1999].

Consider a function $G \in C^2(U, \mathbb{R}_+)$, where U is an open neighborhood of Δ_n^+ and such that $x \mapsto x_i D_i \log G(x)$ is bounded on Δ_n^+ for $i = 1, \ldots, n$. Then G is said to be the generating function of the functionally-generated portfolio $\pi(\cdot)$, given, for $i = 1, \ldots, n$, by

$$\frac{\pi_i(t)}{\mu_i(t)} = \frac{D_i G(\mu(t))}{G(\mu(t))} + 1 - \sum_{j=1}^{n} \mu_j(t) \frac{D_j G(\mu(t))}{G(\mu(t))} \quad (7)$$

In this case, the partial derivative with regard to the variable i is written as D_i, and the second partial derivative with respect to the variables i and j is written as D_{ij}^2.

Theorem 3.1 of Fernholz [1999] asserts that the performance of the wealth process corresponding to $\pi(\cdot)$, when measured relative to the market, satisfies the $\mathbb{P}$-almost sure decomposition

$$\log\left(\frac{V_\pi(T)}{V_\mu(T)}\right) = \log\left(\frac{G(\mu(T))}{G(\mu(0))}\right) + \int_0^T g(t)\, dt \quad (8)$$

where the quantity

$$g(t) := -\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{D_{ij}^2 G(\mu(t))}{2G(\mu(t))} \mu_i(t) \mu_j(t) \tau_{ij}^{\mu}(t) \quad (9)$$

is called the drift process of the portfolio $\pi(\cdot)$. Here, we have written $\tau_{ij}^{\mu}(\cdot)$ for the relative covariances; denoting by e_i the i-th unit vector in $\mathbb{R}^n$, these are defined for $1 \leq i, j \leq n$ as

$$\tau_{\mu_{ij}}(t) := (\mu(t) - e_i)^T \sigma(t) \sigma(t)^T (\mu(t) - e_j) \quad (10)$$

If $T > 0$, the left side of master equation (8) may be constrained away from zero under appropriate conditions on the market model (1), demonstrating that $\pi(\cdot)$ represents an arbitrage with respect to the market over $[0, T]$.

2.4 Diversity-weighted Portfolio

One of the most-studied FGPs is the diversity-weighted portfolio (DWP) with parameter $p \in \mathbb{R}$ (defined in (4.4) of Fernholz et al. [2005]).

$$\pi_i^{(p)}(t)\frac{(\mu_i(t))^p}{\sum_{j=1}^n(\mu_j(t))^p}, \quad i = 1, \ldots, n., \quad (11)$$

This portfolio is a relative arbitrage with respect to $\mu(\cdot)$ over $[0, T]$ for any p and $T > \frac{2\log n}{\varepsilon\delta p}$, under the condition (ND^{ε}), and that of diversity (D^{δ}), introduced below.

$$\exists\delta \in (0,1)\colon P\left(\max_{\{1\le i\le n, t\in[0,T]\}} \mu_i(t) < 1-\delta\right) = 1 \quad (D^{\delta})$$

3 Positive and Negative Parameter P

In Vervuurt and Karatzas [2015], authors found that Diversity-weighted Portfolios with negative parameter p was shown to outperform the market over sufficiently long time horizons and under suitable market assumptions.

First, we will illustrate that under $\mathrm{ND}^{\varepsilon}$ and no-failure (NF) condition, the diversity-weighted portfolio $\pi_i^{(p)}(\cdot)$ with parameter

$$p \in \left(\frac{\log n}{\log(n\varphi)}, 0\right)$$

is a strong arbitrage relative to the market $\mu(\cdot)$ for $[0, T]$, for any number

$$T > \frac{-2n\log(n\varphi)}{\varepsilon(1-p)(n-(n\varphi)^p)}$$

$$\exists\varphi \in (0, \frac{1}{n}) \text{ s.t. } P(\mu_{(n)}(t) > \varphi, \forall t \in [0,T]) = 1 \quad (NF)$$

To prove it, given the fact that the portfolio (11) is generated by the function

$$G_p : x \mapsto \left(\sum_{i=1}^{n} x_i^p \right)^{1/p}, \quad (12)$$

Under the condition that $\sum_{i=1}^{n} x_i$, we apply Lagrange multipliers method to maximize G_p for $p < 0$, which comes up with $x_i = \frac{1}{n}$, we can then derive the upper and lower bounds for G_p,

$$n^{(1-p)} = \sum_{i=1}^{n} \left(\frac{1}{n} \right)^p \leq \sum_{i=1}^{n} (\mu_i(t))^p = (G_p(\mu(t)))^p < \sum_{i=1}^{n} \varphi^p = n\varphi^p, \quad (13)$$

Therefore, with negative p, we can say that

$$\log \left(\frac{G_p(\mu(T))}{G_p(\mu(0))} \right) \geq \log(n\varphi), \quad (14)$$

And for $\pi_{(1)}^{(p)}(t) = \max(\pi^{(p)}(t))$ and $\mu_{(n)}(t) = \min(\mu(t))$,

$$\pi_{(1)}^{(p)}(t) = \frac{(\mu_{(n)}(t))^p}{\sum_{i=1}^{n} (\mu_i(t))^p} < \frac{\varphi^p}{n^{(1-p)}} = \frac{(n\varphi)^p}{p} < 1, \quad (15)$$

Then for the excess growth rate $\gamma_\pi^*(t) = \frac{1}{2} \left(\sum_{i=1}^{n} \pi_i(t) a_{ii}(t) - \sum_{i,j=1}^{n} \pi_i(t) a_{ij}(t) \pi_j(t) \right)$, where $a(\cdot) = \sigma(\cdot)\sigma^T(\cdot)$, combined with ND^ε, we can get that for long-only portfolios,

$$\gamma_\pi^*(t) \geq \frac{\varepsilon}{2} (1 - \pi_{(1)}(t)), \forall t \geq 0, \quad (16)$$

In conjunction with (15), equation (16) can be turned into

$$\int_0^T \gamma_{\pi^{(p)}}^*(t)\, dt \geq \frac{\varepsilon}{2} \int_0^T (1 - \pi_{(1)}^{(p)}(t))\, dt \geq \frac{\varepsilon}{2} T \left(1 - \frac{(n\varphi)^p}{n} \right), \quad (17)$$

Finally, combining all these together, we will obtain the relative performance as

$$\begin{aligned} &\log \left(\frac{V^{\pi^{(p)}}(T)}{V^{\mu}(T)} \right) = \log \left(\frac{G_p(\mu(T))}{G_p(\mu(0))} \right) \\ &+ (1-p) \int_0^T \gamma_{\pi^{(p)}}^*(t)\, dt > \log(n\varphi) + (1-p) \frac{\varepsilon}{2} T \left(1 - \frac{(n\varphi)^p}{n} \right) > 0, \quad (18) \end{aligned}$$

(4)

Then we move on to the comparison between negative and positive parameters. Specifically, the diversity-weighted portfolio with $\pi^{(p^-)}(\cdot)$ with negative parameter

$$p^- \in \left(\frac{\log n}{\log(n\varphi)}, 0\right)$$

is then a strong arbitrage relative to the diversity-weighted portfolio $\pi^{(p^+)}(\cdot)$ with positive parameter

$$p^+ \in \left(\max\left\{0, 1 - \frac{\varepsilon(n-(n\varphi)^{p^-})(1-p^-)}{4K(n-1)}\right\}, 1\right)$$

Over any horizon $[0, T]$ of length

$$T > \frac{-2\log(n\varphi)}{C}$$

Here the positive constant C is defined as

$$C := \frac{\varepsilon}{2}\left(1 - \frac{(n\varphi)^{p^-}}{n}\right)(1-p^-) - \frac{2K}{n(n-1)}(1-p^+)$$

The proof is as follows. We first introduce a similar condition as $\mathrm{ND}^{\varepsilon}$, which is called the bounded variance assumption:

$$\exists K > 0 \text{ s.t. } \xi^T\sigma(t)\sigma^T(t)\xi \leq K\|\xi\|^2, \forall \xi \in \mathbb{R}^n, t \geq 0, \quad (BV)$$

$$\gamma_{\pi^*}(t) \leq 2K(1-\pi_{(1)}(t)), \forall t \geq 0, \quad (19)$$

To make this easier to understand, we write $\pi^{\pm}(\cdot)$ and $G_{\pm}$ for $\pi^{(p^{\pm})}(\cdot)$ and $G_{p^{\pm}}$, respectively. Note that for $p^+ > 0$ there is

$$n\varphi^{p^+} < (G_+(\mu(t)))^{p^+} \leq n^{(1-p^+)} \quad (20)$$

Which gives the lower bound for $p = p^+$. Use observation $\pi^+_{(1)}(t) \geq \frac{1}{n}$, we get that

$$\int_0^T \gamma^*_{\pi^+}(t)\,dt \leq 2K\int_0^T (1-\pi^+_{(1)}(t))\,dt \leq 2KT\left(1-\frac{1}{n}\right) \quad (21)$$

Hence, we see that by the virtue of the two equations the upper bound is then

$$\log\left(\frac{V^{\pi^+}(T)}{V^{\mu}(T)}\right) < -\log(n\varphi) + 2(1-p^+)KT\left(1-\frac{1}{n}\right) \quad (22)$$

Combining this with equation (18) above, we find that

$$\log\left(\frac{V^{\pi^-}(T)}{V^{\pi^+}(T)}\right) = \log\left(\frac{V^{\pi^-}(T)}{V^{\mu}(T)}\right) - \log\left(\frac{V^{\pi^+}(T)}{V^{\mu}(T)}\right)$$

$$> 2\log(n\varphi) + CT > 0 \quad (23)$$

given that

$$CT > -2\log(n\varphi) > 0 \quad (24)$$

An easy calculation shows that $C > 0$, whereas the last inequality in (24) comes from $\varphi < \frac{1}{n}$. Therefore in this case $\pi^-(\cdot)$ outperforms $\pi^+(\cdot)$ strongly over the time-horizon $[0, T]$.

4 Model Specification

4.1 Definition

This part is focused on defining a model for long-only portfolios based on a set of trading characteristics $X \subset \mathbb{R}^d$ for some $d \geq 1$. The portfolio weights are determined by a continuous function $f : X \to \mathbb{R}^+$:

$$\pi_i^f(t) = \frac{f(x_i(t))}{\sum_{j=1}^n f(x_j(t))}, \quad i = 1, \ldots, n, \quad (25)$$

Numerous elements, including market-to-book value, credit ratings, industry momentum, firm size, and balance sheet variables, might influence the choice

of trading characteristics. Functionally-generated portfolios, such as equally-weighted, entropy-, and diversity-generated portfolios, are included in the model as special examples.

The key to this model is the investment map f, which determines how trading opportunities, indicated by evolving trading characteristics $x_i(t)$, are sized. Whence, learning an investment strategy in our framework is equivalent to learning an investment map f. To do so, we consider two families of functions:

Parametric Approach: Where f takes a simple parametric form:

$f : \mu \mapsto \mu^p, \quad$ for $p \in \mathbb{R}$ which corresponds to the diversity-weighted portfolio

Non-Parametric Approach: Where $\log f$ is a path of a mean-zero Gaussian process with a covariance function k:

$$\log f \sim \mathrm{GP}(0, k(\cdot, \cdot)).$$

We must provide an optimality criterion that encodes the user's investment purpose in order to learn "good" investment maps. To do so, we consider a performance functional P_D that maps the logarithm of a candidate investment map to the historical performance $P_D(\log f)$ of the portfolio $\pi^{f(\cdot)}$ as in Eq. (25) over some finite time horizon, given historical data D. An example of a performance functional is the excess return relative to a benchmark portfolio π^*.

$$ER(\pi^f|\pi^*) = \prod_{t=1}^{T}(1 + \sum_{i=1}^{n} r_i(t)\pi_i^f(t)) - \prod_{t=1}^{T}(1 + \sum_{i=1}^{n} r_i(t)\pi_i^*(t)), \quad (26)$$

Another example performance functional is the annualized Sharpe Ratio,

defined as

$$\text{"}SR\text{"}(\pi) = \left(\sqrt{B}\right)\left(\frac{\hat{E}(\{r(1),\ldots,r(T)\})}{\hat{S}(\{r(1),\ldots,r(T)\})}\right), \quad (27)$$

4.2 Inference

For the remainder of this article, the total excess return, adjusted for transaction costs, will be used as the performance functional in relation to the equally weighted portfolio (EWP), which has constant weights.

$$\pi_i(t) = \frac{1}{n}, \quad i = 1,\ldots,n, \quad \forall t \geq 0$$

over the whole training period

$$P_D(\log f) = \text{"ER"}(\pi_f \,|\, \text{"EWP"}), \quad (28)$$

When needed, we use as likelihood model

$$L(\text{"PD"}(\log f)) = \gamma(\text{"PD"}(\log f); a, b), \quad (29)$$

The probability density function of the Gamma distribution with mean a and standard deviation b is represented by the symbol $\gamma(\cdot; a, b)$. As previously discussed, a and b need not be learned as they reflect the investment manager's risk appetite. In the tests that follow, $a = 7.0$ and $b = 0.5$ are used. Stated differently, we conjecture that the ideal investment plan ought to be planned such that, given an initial unit of wealth, the terminal wealth at the conclusion of the training period will, on average, exceed by 7.0 units the terminal wealth that the equally weighted portfolio will attain at the end of the same trading tenure.

Frequentist parametric: Maximizing $P_D(\log f)$ for $p \in [-10, 10]$ yields the optimal parameter of the DWP, which is the first inference method we

explore. In contrast, the SPT literature often considers p to be in the range of $[-1, 1]$. On the uniform grid with mesh size 0.05^2, we use brute force maximizing to prevent any issues with local maxima.

Bayesian parametric: The second inference strategy we examine involves sampling from the posterior distribution over the exponent p in the DWP scenario using the Metropolis-Hastings algorithm (Hastings [1970]).

$$p(p|D) \propto L(P_D(p)) \times \mathbf{1}_{\{p\in[-10,10]\}}, \quad (30)$$

where we have rewritten $L(P_D(\log f))$ as $L(P_D(p))$ to make the dependency on p explicit. We sample a proposal update p^* from a Gaussian centered at the current exponent p and with standard deviation 0.5. The acceptance probability is easily found to be

$$r = \min\left(1, \frac{L(P_D(p^*))}{L(P_D(p))} \times \mathbf{1}_{\{p^*\in[-10,10]\}}\right), \quad (31)$$

Specifically, we note that as long as p is started within $[-10, 10]$, the indicator function in Eq. (30) won't interfere with the Markov chain. Once 10,000 MH repeats have been completed, the first 5,000 are often eliminated as "burn-in." After the proper DWP, we use the posterior mean exponent that we learned from training data to swap in our testing horizon.

$$\hat{f}(\mu) = \mu^{\mathbb{E}(p|D)}, \quad (32)$$

5 Empirical Findings

The stocks we analyze in our experiments consist of the constituents of the S&P 500 index, accounting for changes in index components. We adjust our portfolios on a monthly basis. Our data is sourced from Yahoo Finance, covering the period from January 2008 to December 2022.

We explore optimal investment strategies, as outlined in the prior section, utilizing five consecutive years of data for training purposes and five subsequent years for testing. Our initial data set starts in January 2008 for

Portfolio	IS Return(%)	OOS Return(%)	IS SR	OOS SR
Market	13.1686	13.279	0.3169	0.3043
EWP	20.1405	16.9966	1.4615	1.236
Positive DWP	17.3811	14.8478	1.0947	0.8915
Negative DWP	53.7388	53.6636	1.1967	0.9483
Positive MCMC DWP	19.5541	15.567	1.3059	1.1809
NEgative MCMC DWP	46.7826	56.8979	1.2454	1.0223

Figure 1: Average performance for the portfolios

training, followed by a sequential shift of both training and testing datasets by one year. This process results in six pairs of training and testing subsets. We assess and compare the performance of these portfolios, including the equally-weighted portfolio, the market portfolio, and four diversity-weighted portfolios. In one diversity-weighted portfolio, the exponent p is determined by maximizing the evaluation functional, while in the other, p is learned using MCMC, with both positive and negative one.

Figure 1 summarizes the average in-sample and out-of-sample return and Sharpe ratio for our 6 portfolios during 6 different time periods. We can see that diversity-weighted portfolios with negative p indeed outperform those with positive one in annual return no matter whether MCMC is applied, while the DWP with positive p cannot even beat the equally weighted portfolio. However, when it comes to Sharpe ratio, the equally weighted portfolio surprisingly becomes the best one, which may be due to the fact that diversity-weighted portfolios are poor at controlling volatility. Figure 1 shows the exact wealth process for these 6 portfolios with an in-sample-period of 2009-2013 and out-of-sample-period of 2014-2018, which can help better illustrate our findings above.

Besides, based on our research, we can also see that the diversity-weighted-portfolio coming up by MCMC can slightly outperform that using simple maximization, with a better effort at risk management.

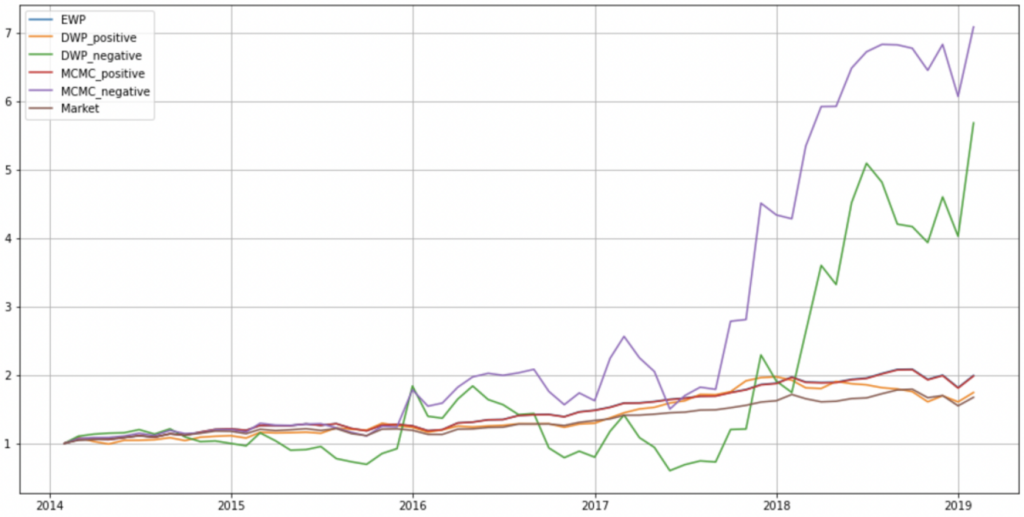


Figure 2: Exact wealth process for 2009-2018

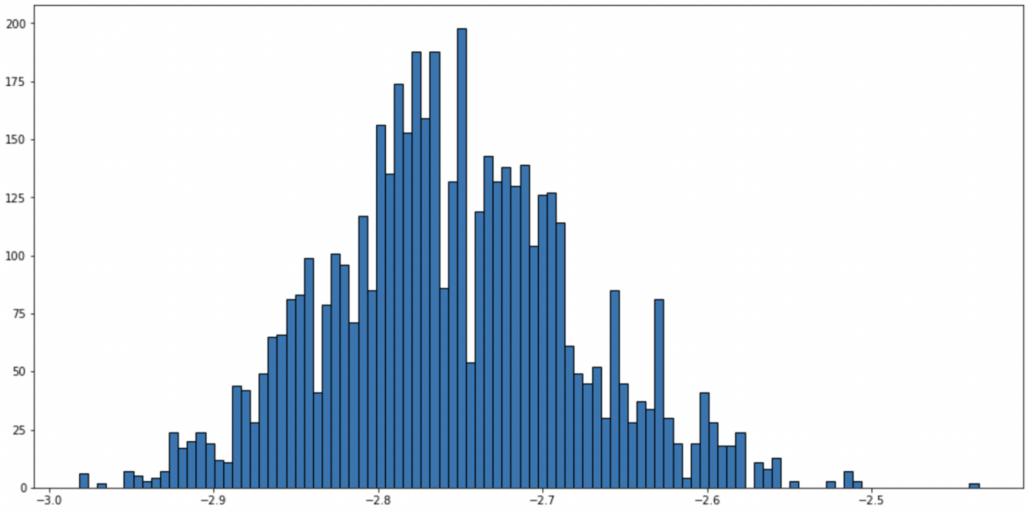


Figure 3: Posterior distribution over the exponent p

An example of the posterior distribution over the exponent p in the Bayesian parametric method is also illustrated in Figure 3, which suggests that p can be highly concentrated after 5000 tries.

6 Conclusion

The empirical study shows that, Diversity-weighted portfolios with negative p indeed outperforms those with positive parameters, since having superior returns. However, Diversity-weighted-portfolio may be poor at risk-management, as the extra returns did not result in a better Sharpe ratio, suggesting increased volatility. Overall, Diversity-weighted-portfolio coming up by MCMC can slightly outperform that using simple maximization, although by a small margin.

References

[1] Samo, Yves-Laurent Kom, and Alexander Vervuurt. "Stochastic Portfolio Theory: A Machine Learning Perspective." *arXiv.Org*, 9 May 2016, arxiv.org/abs/1605.02654.

[2] Vervuurt, Alexander, and Ioannis Karatzas. "Diversity-Weighted Portfolios with Negative Parameter." *arXiv.Org*, 1 July 2015, arxiv.org/abs/1504.01026.

In [1]:
```python
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import minimize_scalar
from functools import partial
```

In [2]:
```python
spy_index = pd.read_excel("spy500.xlsx")
```

In [3]:
```python
spy_list = [string.replace('.', '-') for string in list(spy_index['Symbol'])]
```

In [4]:
```python
stock_data = yf.download('SPY', start='2008-01-01', end='2023-11-30')
monthly_spy = stock_data.resample('M').last()
```

```
[*********************100%***********************]  1 of 1 completed
```

In [5]:
```python
m_price = pd.DataFrame(index = monthly_spy.index,columns = spy_list)
for stock_code in m_price.columns:
    stock_data = yf.download(stock_code, start='2008-01-01', end='2023-11-30')
    monthly_data = stock_data.resample('M').last()
    m_price[stock_code] = monthly_data['Adj Close']
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
```

In [6]:
```python
m_price = m_price.iloc[:-11]
```

In [36]:
```python
m_price.to_excel("All SP500.xlsx")
```

In [7]:
```python
monthly_spy = monthly_spy.iloc[:-11]
```

In [8]:
```
m_price
```

Out[8]:

Date	MMM	AOS	ABT	ABBV	ACN	ADM	ADBE	ADP	A
2008-01-31	48.898937	4.546392	18.715803	NaN	25.789371	29.657907	34.930000	23.896479	13.6233
2008-02-29	48.434818	4.733497	17.839647	NaN	26.258673	30.503391	33.650002	23.525488	12.8312
2008-03-31	48.898144	4.270931	18.372671	NaN	26.199074	27.838570	35.590000	25.144592	11.8963
2008-04-30	47.508129	4.043722	17.691273	NaN	27.972004	29.799994	37.290001	26.218239	12.3888
2008-05-31	48.226898	4.724647	18.898642	NaN	30.407913	26.931993	44.060001	25.536093	13.9017
...	...	...	...	...	...	...	...	...	
2022-08-31	115.903831	55.108898	100.164154	127.937202	282.691589	85.537041	373.440002	236.802841	24.4962
2022-09-30	102.994560	47.425869	94.416801	127.699333	252.154648	78.296211	275.200012	220.096909	21.7530
2022-10-31	117.246025	53.789490	96.999535	140.715378	279.460022	94.383690	318.500000	235.189087	25.3308
2022-11-30	118.788635	59.641720	105.470078	154.921448	296.223663	95.294067	344.929993	257.024628	28.0033
2022-12-31	113.083527	56.205013	107.636734	155.334778	262.666779	90.749268	336.529999	233.551819	27.8484

180 rows × 503 columns

In [9]:
```
def ER(p,start_index,end_index):
    port_return = 1
    ew_return = 1
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        miu_series = price_series/sum(price_series)
        miu_p_series = miu_series**p
        port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
        ew_w = np.full(port_size,1/port_size)   # weights for EW
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for p
        ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
        port_return *= (1 + port_return_now)
        ew_return *= (1 + ew_return_now)
    return (port_return - ew_return)
```

In [10]:
```
def DWP_cal(p,start_index,end_index):
    port_return = 1
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        miu_series = price_series/sum(price_series)
        miu_p_series = miu_series**p
        port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for p
        port_return *= (1 + port_return_now)
    return port_return
```

In [11]:
```
def EW_cal(start_index,end_index):
    ew_return = 1
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        ew_w = np.full(port_size,1/port_size)   # weights for EW
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
        ew_return *= (1 + ew_return_now)
    return ew_return
```

In [12]:
```
def DWP_sp(p,start_index,end_index):
    return_list = []
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        miu_series = price_series/sum(price_series)
        miu_p_series = miu_series**p
        port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for p
        return_list.append(port_return_now)
    return np.sqrt(12)*np.mean(return_list)/np.std(return_list)
```

In [13]:
```
def EW_sp(start_index,end_index):
    return_list = []
    if end_index == len(m_price):
        end_index -= 1
    for i in range(start_index,end_index):
        price_series = m_price.iloc[i].dropna()   # Price for existing companies
        port_size = len(price_series)
        ew_w = np.full(port_size,1/port_size)   # weights for EW
        now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].i
        ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
        return_list.append(ew_return_now)
    return np.sqrt(12)*np.mean(return_list)/np.std(return_list)
```

In [14]:
```
def target_pdf(x):
    return np.exp(-2 * (x-7)**2) / (np.sqrt(2 * np.pi)*0.5)
def positive_mh(start_index,end_index,times):
    samples = [0.1]
    total_num = 0
    while total_num < times:
        current_sample = samples[-1]
        proposal = np.random.normal(current_sample, 0.5)
        if 0 < proposal <= 10:
            now_er = ER(proposal,start_index,end_index)  # new ER
            past_er = ER(current_sample,start_index,end_index) # last ER
            acceptance_prob = min(1, target_pdf(now_er) / target_pdf(past_er))
        else:
            acceptance_prob = 0
        if np.random.rand() < acceptance_prob:
            samples.append(proposal)
        else:
            samples.append(current_sample)
        total_num += 1
    use = int(times/2)
    return sum(samples[use:])/len(samples[use:])
```

In [15]:
```
def negative_mh(start_index,end_index,times):
    samples = [-0.1]
    total_num = 0
    while total_num < times:
        current_sample = samples[-1]
        proposal = np.random.normal(current_sample, 0.5)
        if -10 <= proposal < 0:
            now_er = ER(proposal,start_index,end_index)  # new ER
            past_er = ER(current_sample,start_index,end_index) # last ER
            acceptance_prob = min(1, target_pdf(now_er) / target_pdf(past_er))
        else:
            acceptance_prob = 0
        if np.random.rand() < acceptance_prob:
            samples.append(proposal)
        else:
            samples.append(current_sample)
        total_num += 1
    use = int(times/2)
    return sum(samples[use:])/len(samples[use:])
```

In [165]:
```
ewp_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015-2019','2
                          columns = ['In sample return','Out of sample return','In sample
```

In [172]:
```
for i in range(6):
    ewp_result['In sample return'].iloc[i] = (EW_cal(i*12,i*12+60)**(1/5))-1
    ewp_result['Out of sample return'].iloc[i] = (EW_cal(i*12+60,i*12+120)**(1/5))-1
    ewp_result['In sample Sharpe ratio'].iloc[i] = EW_sp(i*12,i*12+60)
    ewp_result['Out of sample Sharpe ratio'].iloc[i] = EW_sp(i*12+60,i*12+120)
```

In [173]:
```
ewp_result
```

Out[173]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio
2008-2013-2017	0.128880	0.206449	0.668387	1.984674
2009-2014-2018	0.294831	0.147824	1.585950	1.236313
2010-2015-2019	0.226901	0.154104	1.560587	1.240552
2011-2016-2020	0.155977	0.197047	1.202287	1.158742
2022-2017-2021	0.195389	0.192977	1.767384	1.112454
2013-2018-2022	0.206449	0.121392	1.984674	0.683283

In [175]:
```
dwp_positive_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015
                          columns = ['In sample return','Out of sample return','In sample
```

In [180]:
```
for i in range(6):
    f_fixed = partial(ER, start_index = i*12, end_index = i*12+60)
    result = minimize_scalar(lambda x: -f_fixed(x), bounds=(0,10), method='bounded')
    p_op = result.x  # The optimal p calculated
    dwp_positive_result['p'].iloc[i] = p_op
    dwp_positive_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**(1/5)
    dwp_positive_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*12+120
    dwp_positive_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+60)
    dwp_positive_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60,i*12
```

In [181]:
```
dwp_positive_result
```

Out[181]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.034941	0.199054	0.263008	1.094159	9.999993
2009-2014-2018	0.245718	0.117571	1.287796	0.875638	2.059512
2010-2015-2019	0.226901	0.154104	1.560586	1.240551	0.000004
2011-2016-2020	0.144734	0.172419	1.0207	0.929848	2.167683
2022-2017-2021	0.178457	0.185201	1.141053	0.834859	3.1102
2013-2018-2022	0.212113	0.062519	1.295071	0.374085	3.40947

In [182]:
```
dwp_negative_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015
                          columns = ['In sample return','Out of sample return','In sample
```

In [183]:
```
for i in range(6):
    f_fixed = partial(ER, start_index = i*12, end_index = i*12+60)
    result = minimize_scalar(lambda x: -f_fixed(x), bounds=(-10,0), method='bounded')
    p_op = result.x  # The optimal p calculated
    dwp_negative_result['p'].iloc[i] = p_op
    dwp_negative_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**(1/5)
    dwp_negative_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*12+120
    dwp_negative_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+60)
    dwp_negative_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60,i*12
```

In [184]:
```
dwp_negative_result
```

Out[184]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.709799	0.324198	1.260885	0.676629	-3.994956
2009-2014-2018	0.853127	0.415565	1.777244	0.753048	-5.790833
2010-2015-2019	0.425495	0.565043	1.288168	0.880516	-4.726369
2011-2016-2020	0.412751	0.882758	0.809628	1.093726	-9.999995
2022-2017-2021	0.358023	0.637785	1.044382	1.156099	-2.32685
2013-2018-2022	0.465132	0.394466	1.000173	1.129521	-2.169052

In [185]:
```
dwp_mcmc_positive_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010
                          columns = ['In sample return','Out of sample return','In sample
```

In [186]:
```
for i in range(6):
    p_op = positive_mh(i*12,i*12+60,10000)  # The optimal p calculated
    dwp_mcmc_positive_result['p'].iloc[i] = p_op
    dwp_mcmc_positive_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**
    dwp_mcmc_positive_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*1
    dwp_mcmc_positive_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+6
    dwp_mcmc_positive_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60
```

In [190]:
```
dwp_mcmc_positive_result
```

Out[190]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.124145	0.204595	0.651217	1.97552	0.066688
2009-2014-2018	0.292322	0.146929	1.580352	1.231204	0.036686
2010-2015-2019	0.223552	0.15213	1.548176	1.233339	0.112518
2011-2016-2020	0.145077	0.185554	1.157055	1.170171	0.680614
2022-2017-2021	0.177104	0.180825	1.65503	1.098746	0.719661
2013-2018-2022	0.211047	0.063985	1.243715	0.376436	4.116748

In [188]:
```
dwp_mcmc_negative_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010
                           columns = ['In sample return','Out of sample return','In sample
```

In [191]:
```
for i in range(6):
    p_op = negative_mh(i*12,i*12+60,10000)  # The optimal p calculated
    dwp_mcmc_negative_result['p'].iloc[i] = p_op
    dwp_mcmc_negative_result['In sample return'].iloc[i] = (DWP_cal(p_op,i*12,i*12+60)**
    dwp_mcmc_negative_result['Out of sample return'].iloc[i] = (DWP_cal(p_op,i*12+60,i*1
    dwp_mcmc_negative_result['In sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12,i*12+6
    dwp_mcmc_negative_result['Out of sample Sharpe ratio'].iloc[i] = DWP_sp(p_op,i*12+60
```

In [192]:
```
dwp_mcmc_negative_result
```

Out[192]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio	p
2008-2013-2017	0.545999	0.451453	1.378603	0.888869	-2.47227
2009-2014-2018	0.603569	0.479669	2.001137	0.989865	-2.201698
2010-2015-2019	0.425108	0.567071	1.250468	0.879548	-5.043002
2011-2016-2020	0.409159	0.877568	0.80913	1.089986	-9.133334
2022-2017-2021	0.357996	0.642508	1.035215	1.154934	-2.345971
2013-2018-2022	0.465128	0.395604	0.99814	1.130416	-2.173547

In [193]:
```
market_result = pd.DataFrame(index = ['2008-2013-2017','2009-2014-2018','2010-2015-2019'
                           columns = ['In sample return','Out of sample return','In sample
```

In [196]:
```
for i in range(6):
    in_return = []
    out_return = []
    for j in range(i*12,i*12+60):
        in_return.append(monthly_spy['Adj Close'].iloc[j+1]/monthly_spy['Adj Close'].ilo
    if i!=5:
        market_result['Out of sample return'].iloc[i] = ((monthly_spy['Adj Close'].iloc[
        for k in range(i*12+60,i*12+120):
            out_return.append(monthly_spy['Adj Close'].iloc[k+1]/monthly_spy['Adj Close'
    else:
        market_result['Out of sample return'].iloc[i] = ((monthly_spy['Adj Close'].iloc[
        for k in range(i*12+60,i*12+119):
            out_return.append(monthly_spy['Adj Close'].iloc[k+1]/monthly_spy['Adj Close'
    market_result['In sample return'].iloc[i] = ((monthly_spy['Adj Close'].iloc[i*12+60]
    market_result['In sample Sharpe ratio'].iloc[i] = np.mean(in_return)/np.std(in_retur
    market_result['Out of sample Sharpe ratio'].iloc[i] = np.mean(out_return)/np.std(out
```

In [197]:
```
market_result
```

Out[197]:

	In sample return	Out of sample return	In sample Sharpe ratio	Out of sample Sharpe ratio
2008-2013-2017	0.039721	0.157935	0.087764	0.462775
2009-2014-2018	0.190298	0.108372	0.354377	0.284419
2010-2015-2019	0.155017	0.122244	0.344500	0.302337
2011-2016-2020	0.107727	0.160408	0.265928	0.313492
2022-2017-2021	0.139419	0.166570	0.386198	0.311661
2013-2018-2022	0.157935	0.081210	0.462775	0.151122

In [65]:
```
wealth_process = pd.DataFrame(index = m_price.index[72:133],columns = ['EWP','DWP_positi
                                                                        'MCMC_positive','MC
```

In [66]:
```
wealth_process.iloc[0] = 1
```

In [67]:
```
start_index = 72
end_index = 132
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    ew_w = np.full(port_size,1/port_size)   # weights for EW
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    ew_return_now = np.dot(np.array(ew_w),np.array(now_return).T) # return for EW
    wealth_process['EWP'].iloc[i-71] = wealth_process['EWP'].iloc[i-72]*(1 + ew_return_n
```

In [68]:
```
start_index = 72
end_index = 132
p = 2.059512
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['DWP_positive'].iloc[i-71] = wealth_process['DWP_positive'].iloc[i-72
```

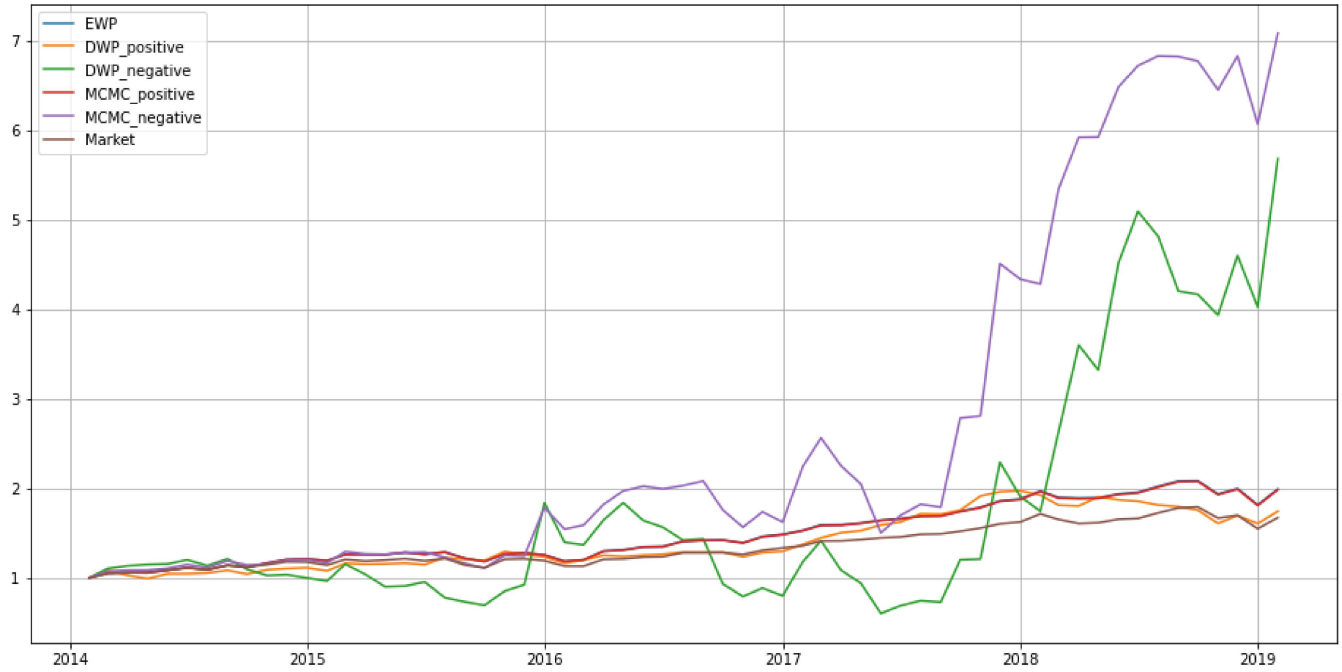
In [69]:
```
start_index = 72
end_index = 132
p = -5.790833
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['DWP_negative'].iloc[i-71] = wealth_process['DWP_negative'].iloc[i-72
```

In [70]:
```
start_index = 72
end_index = 132
p = 0.036686
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['MCMC_positive'].iloc[i-71] = wealth_process['MCMC_positive'].iloc[i-
```

In [71]:
```
start_index = 72
end_index = 132
p = -2.201698
for i in range(start_index,end_index):
    price_series = m_price.iloc[i].dropna()   # Price for existing companies
    port_size = len(price_series)
    miu_series = price_series/sum(price_series)
    miu_p_series = miu_series**p
    port_w = miu_p_series/sum(miu_p_series)     # weights for portfolio
    now_return = m_price[price_series.index].iloc[i+1]/m_price[price_series.index].iloc[
    port_return_now = np.dot(np.array(port_w),np.array(now_return).T) # return for portf
    wealth_process['MCMC_negative'].iloc[i-71] = wealth_process['MCMC_negative'].iloc[i-
```

In [74]:
```
for i in range(1,len(wealth_process)):
    wealth_process['Market'].iloc[i] = wealth_process['Market'].iloc[i-1]*monthly_spy['A
```

In [76]:
```
plt.figure(figsize = (16,8))
plt.plot(wealth_process,label = wealth_process.columns)
plt.legend()
plt.grid()
```



In [75]:
```
wealth_process
```

Out[75]:

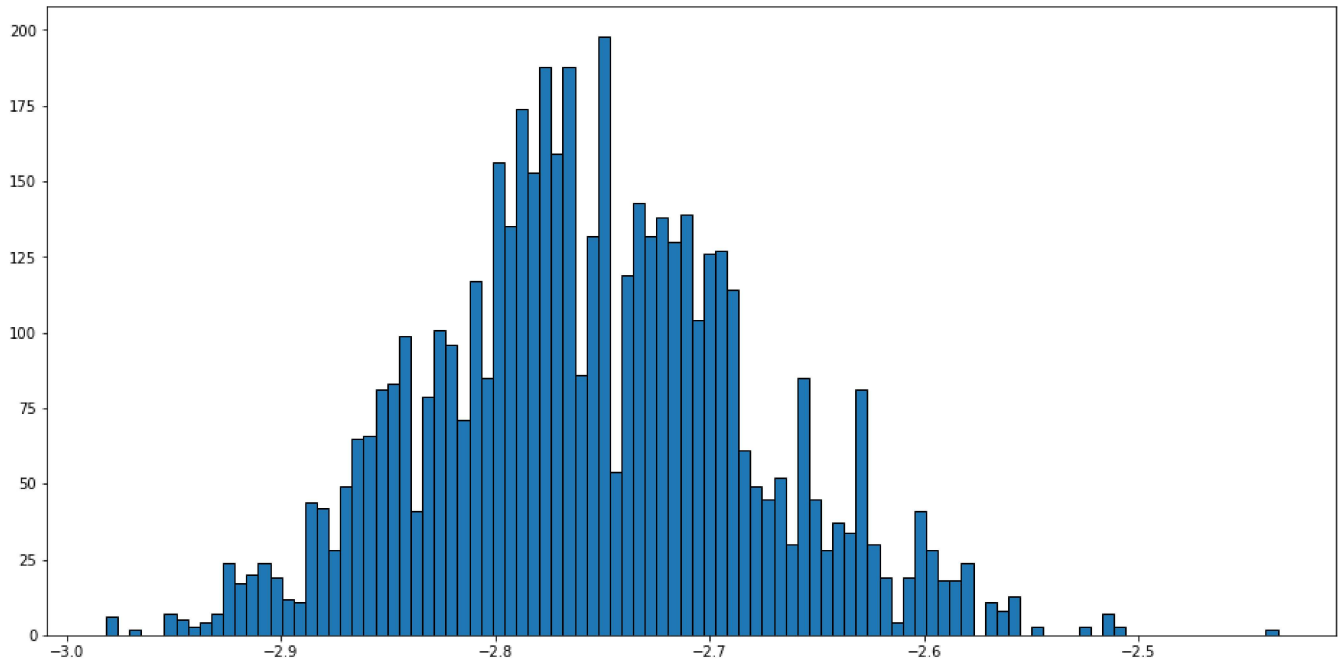
Date	EWP	DWP_positive	DWP_negative	MCMC_positive	MCMC_negative	Market
2014-01-31	1	1	1	1	1	1
2014-02-28	1.053516	1.083561	1.104321	1.053486	1.072681	1.045515
2014-03-31	1.057414	1.026214	1.134327	1.057174	1.083873	1.054188
2014-04-30	1.055021	0.989733	1.14848	1.054656	1.08527	1.061517
2014-05-31	1.083096	1.044341	1.154399	1.082829	1.10613	1.086151
...	...	...	...	...	...	...
2018-09-30	2.084444	1.756266	4.168382	2.077252	6.77684	1.791115
2018-10-31	1.936487	1.607426	3.935585	1.92886	6.454705	1.667342
2018-11-30	1.997767	1.695909	4.603026	1.990119	6.833905	1.69827
2018-12-31	1.814436	1.605865	4.027833	1.807924	6.071789	1.54874
2019-01-31	1.9924	1.743342	5.685332	1.984646	7.088731	1.672741

61 rows × 6 columns

In [44]:
```
start_index = 120
end_index = 180
times = 10000
samples = [-0.1]
total_num = 0
while total_num < times:
    current_sample = samples[-1]
    proposal = np.random.normal(current_sample, 0.5)
    if -10 <= proposal < 0:
        now_er = ER(proposal,start_index,end_index)  # new ER
        past_er = ER(current_sample,start_index,end_index) # last ER
        acceptance_prob = min(1, target_pdf(now_er) / target_pdf(past_er))
    else:
        acceptance_prob = 0
    if np.random.rand() < acceptance_prob:
        samples.append(proposal)
    else:
        samples.append(current_sample)
    total_num += 1
```

In [53]:
```
plt.figure(figsize = (16,8))
plt.hist(samples[5000:],bins = 100,edgecolor = 'black')
```

Out[53]:
```
(array([  6.,   0.,   2.,   0.,   0.,   7.,   5.,   3.,   4.,   7.,  24.,
         17.,  20.,  24.,  19.,  12.,  11.,  44.,  42.,  28.,  49.,  65.,
         66.,  81.,  83.,  99.,  41.,  79., 101.,  96.,  71., 117.,  85.,
        156., 135., 174., 153., 188., 159., 188.,  86., 132., 198.,  54.,
        119., 143., 132., 138., 130., 139., 104., 126., 127., 114.,  61.,
         49.,  45.,  52.,  30.,  85.,  45.,  28.,  37.,  34.,  81.,  30.,
         19.,   4.,  19.,  41.,  28.,  18.,  18.,  24.,   0.,  11.,   8.,
         13.,   0.,   3.,   0.,   0.,   0.,   3.,   0.,   7.,   3.,   0.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
          2.]),
 array([-2.9818129 , -2.97634696, -2.97088101, -2.96541507, -2.95994912,
        -2.95448317, -2.94901723, -2.94355128, -2.93808534, -2.93261939,
        -2.92715344, -2.9216875 , -2.91622155, -2.91075561, -2.90528966,
        -2.89982372, -2.89435777, -2.88889182, -2.88342588, -2.87795993,
        -2.87249399, -2.86702804, -2.86156209, -2.85609615, -2.8506302 ,
        -2.84516426, -2.83969831, -2.83423236, -2.82876642, -2.82330047,
        -2.81783453, -2.81236858, -2.80690263, -2.80143669, -2.79597074,
        -2.7905048 , -2.78503885, -2.77957291, -2.77410696, -2.76864101,
        -2.76317507, -2.75770912, -2.75224318, -2.74677723, -2.74131128,
        -2.73584534, -2.73037939, -2.72491345, -2.7194475 , -2.71398155,
        -2.70851561, -2.70304966, -2.69758372, -2.69211777, -2.68665182,
        -2.68118588, -2.67571993, -2.67025399, -2.66478804, -2.65932209,
        -2.65385615, -2.6483902 , -2.64292426, -2.63745831, -2.63199237,
        -2.62652642, -2.62106047, -2.61559453, -2.61012858, -2.60466264,
        -2.59919669, -2.59373074, -2.5882648 , -2.58279885, -2.57733291,
        -2.57186696, -2.56640101, -2.56093507, -2.55546912, -2.55000318,
        -2.54453723, -2.53907128, -2.53360534, -2.52813939, -2.52267345,
        -2.5172075 , -2.51174155, -2.50627561, -2.50080966, -2.49534372,
        -2.48987777, -2.48441183, -2.47894588, -2.47347993, -2.46801399,
        -2.46254804, -2.4570821 , -2.45161615, -2.4461502 , -2.44068426,
        -2.43521831]),
 <BarContainer object of 100 artists>)
```



In [ ]: