Received: 31 March 2020 | Revised: 12 February 2021 | Accepted: 8 May 2021

DOI: 10.1002/isaf.1498

RESEARCH ARTICLE

WILEY

# Particle swarm optimization approach to portfolio construction

**Ren-Raw Chen**[1] | **Wiliam Kaihua Huang**[2] | **Shih-Kuo Yeh**[3]

[1]Graduate School of Business Administration, Fordham University, New York, NY, USA

[2]Union Bank of Switzerland, Shanghai, China

[3]Department of Finance, National Chung Hsing University, Taichung, Taiwan

**Correspondence**
Shih-Kuo Yeh, Department of Finance, National Chung Hsing University, 250 Kuo-Kuang Road, Taichung 402, Taiwan.
Email: seiko@nchu.edu.tw

**Summary**

Particle swarm optimization (PSO) is an artificial intelligence technique that can be used to find approximate solutions to extremely difficult or impossible numeric optimization problems. Recently, PSO algorithms have been widely used in solving complex financial optimization problems. This paper presents a PSO approach to solve a portfolio construction problem, since this methodology is a population-based heuristic algorithm that is suitable for solving high-dimensional constrained optimization problems. The computational results show that PSO algorithms have advantages in optimizing the Sortino ratio, especially in speed, when the size of the portfolio is large.

**KEYWORDS**

particle swarm optimization, portfolio construction, Sortino ratio

## 1 | INTRODUCTION

Portfolio optimization (or portfolio construction) has had a long history, and yet it remains as one of the most popular research topics in finance. Investors, especially institutional investors, pay attention to portfolio construction in a high-dimensional space, namely using fewer observations than the number of assets to construct an estimated diversified portfolio. The high-dimensional portfolio construction problem is popular and realistic, as there are, indeed, many assets in the market. Yet, until recently, high-dimensional portfolio construction problems were computationally impermissible. Luckily, with the help of fast-growing artificial intelligence (AI) techniques, this high-dimensional optimization problem can now possibly be solved by AI techniques in a faster and better way.

Particle swarm optimization (PSO) is an AI technique that can be used to find approximate solutions to extremely difficult or impossible numeric optimization problems. As a result, it is a natural choice to use to solve the portfolio optimization problem. In particular, we demonstrate in constrained cases[1] where PSO has the most advantage. This is because constraints in constructing a portfolio lead to non-differentiability of portfolio value in portfolio weights and, as expected, the usual numerical algorithms that require first- and second-order derivatives should fail.

We thank the two referees for their valuable comments. All errors are our own.

Though the use of PSO in finance applications has been voluminous (see Section 2), there have only been a limited number of papers using PSO on portfolios. Unfortunately, either (1) they employ very few stocks (Kendall & Su, 2005; Zhu et al., 2011), (2) experiments are performed on simulated data (Corazza et al., 2013), or (3) they explore limitations of PSO in complex constrained problems (Cura, 2009). What we demonstrate in this paper is the power of PSO in a high-dimensional search and how it can be easily used for a large number of stocks such as the S&P 500 portfolio.[2] In particular, we compare the performance of PSO with SciPy, a popular Python optimization algorithm, and examine how constraints imposed on the portfolio optimization problem (see Section 2 for details) render AI methods such as PSO particularly useful.

To demonstrate the importance of constraints for PSO, we employ two very popular portfolio performance indices used by the financial industry: the Sharpe ratio index, which represents a nonconstraint portfolio optimization problem, and the Sortino ratio index, which represents a constraint problem. Not only are these the two indices popular in practice, they are also similar to each other (calculation-wise) and yet different enough to cause drastic differences in PSO versus SciPy outcomes.

In addition to the methodological advantage by comparing the two indices, the introduction of PSO in solving the Sortino ratio index (first time in the literature) is useful in practice. The general portfolio theory (which the Sharpe ratio index is based upon) has been widely

criticized to inappropriately measure risk using standard deviation (or variance) of stock returns. The reason is that standard deviation is "symmetrical," in that it measures upside volatility (when the market rises) equally as downside volatility (when the market plummets). Yet, investors view upward volatility favorably and downward volatility only as risk. The Sortino ratio index efficiently corrects this bias. The Sortino ratio will only include downward returns in the standard deviation calculation and, hence, only capture downward returns as risk. This straightforward correction in computing the standard deviation unfortunately causes the standard optimization algorithms that rely on differentiability to fail. As we demonstrate, PSO can outperform SciPy in such a situation.

Lastly, we compare a few varieties of PSO for the portfolio optimization problem and find no improvements. The very basic PSO is already efficient for the portfolio optimization problem.

The remainder of this paper is organized as follows. Section 2 presents a literature review. PSO algorithms and the portfolio optimization problem are explained in Section 3. Section 4 presents the computational experimental results. Finally, the conclusions are drawn in Section 5.

## 2 | LITERATURE REVIEW

PSO is a heuristic optimization method proposed by Kennedy and Eberhart (1995). In 2000, some popular heuristic optimization methods, including genetic algorithm, Tabu search, and simulated annealing, were used to find the cardinality-constrained efficient frontier for a portfolio (Chang et al., 2000). Kendall and Su (2005), for the first time, applied PSO to solve the portfolio optimization problem based on the Sharpe ratio subject to constraints of both restricted and unrestricted short selling. The PSO results outperformed Excel Solver on the portfolios with 5, 12, and 20 stocks. Zhu et al. (2011) demonstrated that PSO is better in optimizing the Sharpe ratio under restricted and unrestricted portfolio compared with a genetic algorithm and Excel Solver on portfolios with 8, 15, and 49 assets. However, when optimizing the efficient frontier, Zhu et al. (2011) found PSO is only generally better than other methods. Also, their experiment illustrates that even using models with linear decreasing parameters cannot significantly improve the results. Corazza et al. (2013) relaxed the assumption that returns are elliptically distributed and hence no easy solutions to the portfolio problem can be found. In studying a series of novel risk measures (which are non-differentiable) via a large number of simulations, they concluded that PSO provides satisfactory outcomes. Lastly, Cura (2009) employed PSO to find the cardinality-constrained (i.e., a 0/1 selection variable was included in choosing stocks) efficient frontier. He compared PSO with three other search algorithms (genetic algorithm, tabu search, and simulated annealing) on four portfolios: the Hang Seng in Hong Kong, the DAX 100 in Germany, the FTSE 100 in the UK, the S&P 100 in the USA, and the Nikkei in Japan. The experimental results showed that none of the four heuristics significantly outperformed the others, but PSO was better when dealing with portfolio construction under low risk.

Besides the original PSO, many modified PSO algorithms have been created for different portfolio optimization problems. Golmakani and Fazel (2011) come up with a CBIPSO algorithm which means combining Binary PSO proposed by Kennedy and Eberhart (1997) and improved PSO with mutation. The proposed combined binary and improved PSO performed better than a generic algorithm in optimizing the efficient frontier. Deng et al. (2012) invent an improved PSO combining mutation and linear decreasing coefficients to optimize the efficient frontier. The results confirm that the proposed PSO is more robust than other PSO-variants and outperforms other genetic algorithm, simulated annealing and Tabu search.

In addition to portfolio optimization, PSO has been also used in other financial applications, some in combination with other AI/ML methods. Cagcag Yolcu and Alpaslan (2018) used PSO in combination with a fuzzy time series to study the TAIEX stock index of Taiwan. Gao and Su (2020) studied bitcoin returns from June 2, 2016, to December 30, 2018, using PSO and other AI/machine-learning tools (support vector machine and NN). In a series of studies, Lahmiri (2016, 2017) investigated (1) six interest rate series and (2) electricity and crude oil prices using various varieties of neural networks in which PSO was used to determine the initial weights. In a subsequent article, Lahmiri (2018) tried to forecast intra-day stock prices with various AI/machine-learning models: support vector regression, singular spectrum analysis, and PSO. Pradeepkumar and Ravi (2017) forecast the volatilities of four exchange rates (JPY, GBP, EUR, and INR), two commodity indices (gold and crude oil), and two stock indices (S&P 500 and NSE India Stock Index) using a PSO-trained quantile regression neural network. Lu et al. (2013) used a two-level PSO to solve a credit portfolio management problem that sought to minimize the maximum expected loss due to defaults.[3] Marinakis et al. (2009) used PSO for feature selection and classification in credit risk and audit. Though neural networks are the common models for classifying credit quality, Marinakis et al. (2009) used PSO to select features that determine the credit quality of a company.

A thorough review of the literature revealed that most comparisons are between heuristics (genetic algorithm, simulated annealing, and Tabu search) and Visual Basic for Applications. None of the papers compare the results with the popular and powerful Python optimization package SciPy-SLSQP (sequential least-squares programming). The experimental results of this paper will be compared with results optimized by SciPy in Section 4.

As the PSO-related literature is surveyed, we find that a speed comparison and the relationship between algorithm speed and parameters setting seem rarely examined. Since speed is also a very crucial factor in practical portfolio construction, its relevant performance comparison will also be incorporated in Section 4.

## 3 | ALGORITHMS ABOUT PSO

PSO is a kind of optimization algorithm in computer science. It was invented by Kennedy and Eberhart (1995) by simulating the social behavior of flocking birds. Subsequently, a lot of research has been

done about PSO, including parameter sensitivity analysis and modified PSO algorithms that improve the performance compared with the original PSO.

Jamous et al. (2015) found that a new particle swarm with center of mass can perform better in convergence rate, complexity and scalability by introducing popular existing modified PSO algorithms – Center Particle Swarm Optimization and Linear Decreasing Weight Particle Swarm Optimization. The following will demonstrate more details about the original PSO and two modified PSO algorithms mentioned above.

### 3.1 | Basic PSO

PSO is a social population-based heuristic search algorithm for optimization. A PSO swarm represents a population containing a lot of particles. Each particle represents a possible solution. For portfolio optimization, each particle is a possible weight assignment for the portfolio. Denote the $k$th particle (where $k = 1,...,K$ particles) at the $l$th iteration (where $l = 1,...,L$ iterations) as $X_k^l = \langle x_1, x_2, ..., x_n \rangle_k^l$ is a vector with $n$ elements.[4]

Initially, the particles are randomly dropped in a feasible area. Then, at each iteration, the particle $X_k^l$ will be updated according to velocity $V_k^l$, which is also a vector with $n$ dimensions. The update formula is as follows:

$$X_k^{l+1} = X_k^l + V_k^l \tag{1}$$

where the velocity is decided by the following rule, which is the essential part of the PSO algorithm:

$$V_k^{l+1} = wV_k^l + c_1 r_1 \left(p_k^l - X_k^l\right) + c_2 r_2 \left(g_l - X_k^l\right) \tag{2}$$

where $r_1$ and $r_2$ are uniform random numbers and $c_1$ and $c_2$ are positive acceleration coefficients. Each particle, which is a vector of values $X_k^l = \langle x_1, x_2, ..., x_n \rangle_k^l$, is valued by target function $F\left(X_k^l\right)$. In this paper, the target function is the Sharpe ratio (Equation 13) or the Sortino ratio (Equation 16). If PSO is minimizing the target function, then the less the target function value is the better the performance. In addition, $p_k^l$ is the personal best position that the particle $X_k^l = \langle x_1, x_2, ..., x_n \rangle_k^l$ has experienced since the first iteration. Moreover, $g_l$ is the global best position (at iteration $l$) that the whole swarm has experienced since the first iteration. Finally, $w$ is the decay factor for previous velocity. Intuitively, the velocity is comprised of three velocity components: the previous velocity, a velocity toward the personal best position, and a velocity toward the global best position.

PSO is controlled by three parameters: $w$, $c_1$, and $c_2$. The constant $w$ measures the weight for previous velocity, if $w \le 1$ the velocity will decay overtime until it becomes zero, if $w > 1$, the velocity will keep accelerating and cannot move back. The first scale constant $c_1$ scales the contribution of personal cognitive experience. A larger $c_1$ relative to $c_2$ means the particle is confident about itself and will focus on a local search. Similarly, $c_2$ scales the contribution of swarm social experience. A larger $c_2$ means the particle is confident about the swarm and pays more attention to a global search. So, there is a trade-off between a global and local search when balancing $c_1$ and $c_2$. In our experiments, the parameters are set as $c_1 = 1.49618$, $c_2 = 1.49618$, $w = 0.7298$, as suggested by van den Bergh and Engelbrecht (2006).

### 3.2 | Particle swarm with center of mass optimization

Particle swarm with center of mass optimization (PSOCM) adds in a new virtual particle to the basic PSO by considering the swarm as a system of point masses. The center of mass can be expressed as

$$X_{\text{cm}} = \sum_{k=1}^{n_S} F(X_k) \sum_{k=1}^{n_S} \frac{1}{F(X_k)} \otimes X_k \tag{3}$$

where $X_k$ and $X_{\text{cm}}$ are both vectors, and $n_S$ is the number of particles in the swarm. The $\otimes$ symbol in the equation represents element-wise multiplication; for example, the value of $1/F(X_k)$ will multiply each element in vector $X_k$.

Then, the new velocity update rule can be formulated as follows:

$$V_k^{l+1} = wV_k^l + c_1 r_1 \left(\frac{p_k^l + X_{\text{cm}}}{2} - X_k^l\right) + c_2 r_2 \left(\frac{g_l + X_{\text{cm}}}{2} - X_k^l\right) \tag{4}$$

According to the study by Jamous et al. (2015), the term $\left[\left(p_k^l + X_{\text{cm}}\right)/2\right] - X_k^l$ is responsible for the attraction of a particle's current position towards its own best position and the center of mass of the swarm, which helps the cognitive behavior component to avoid the local optima. The term $\left[\left(g_l + X_{\text{cm}}\right)/2\right] - X_k^l$ helps improve the diversity of the swarm during the global search. The parameters are set to be $c_1 = 0.5$, $c_2 = 2.3$, $w = 0.5$ for the experiment part of this paper.

### 3.3 | Linear decreasing coefficients technique

As mentioned, coefficients $c_1$, $c_2$, and $w$ control the trade-off between a global search and local search. Since the particles are randomly dropped in the feasible space at the initial stage, then intuitively the particles should focus on a global search in the beginning; then, when converging at the final stage, the particles should pay more attention to a local search. To achieve this goal, the coefficients can decrease or increase linearly as iteration increases. This technique can be formulated as follows:

$$\begin{aligned} w &= (w_{\min} - w_{\max}) \frac{L-k}{L} + w_{\max} \\ c_1 &= (c_{1,\min} - c_{1,\max}) \frac{k}{L} + c_{1,\max} \\ c_2 &= (c_{2,\max} - c_{2,\min}) \frac{k}{L} + c_{1,\min} \end{aligned} \tag{5}$$

where $L$ is the maximum iteration the PSO algorithms can run, which is set to be 100 in the experiment part, and $k$ represents the current iteration. PSO and PSOCM with this technique are denoted as PSOLD and PSOCMLD, respectively.

### 3.4 | Constraint satisfaction

For the portfolio optimization problem, there is a necessary constraint, Equation 3, that is the sum of weights equals to one and an optional short-selling constraint, Equation 5. Since the particle moves randomly, in most cases, the particle cannot provide a feasible solution after position update. This paper employs the following algorithm proposed by Cura (2009) to make sure the particle is feasible.

For portfolio optimization, vector $X_k^l = \langle x_1, x_2, \ldots, x_n \rangle_k^l$ contains $n$ asset weights. $x_i$ represents the $i$th element in $X_k^l$ or, say, the weight for $i$th asset. (The subscript $k$ in $X_k^l$ means the $k$th particle in the swarm, which is different from the subscript $i$ in $x_i$). In order to let each particle be feasible and satisfy Equation 5, four different measures ($\delta^*$, $\varepsilon^*$, $\eta$, and $\phi$) about the sum of difference between original weight $x_i$ and bounds $u_i$ or $\ell_i$ are defined as follows:

$$\begin{aligned} \delta^* &= \sum_{i=1}^{n} \max\{0, u_i - x_i\} \\ \varepsilon^* &= \sum_{i=1}^{n} \max\{0, x_i - \ell_i\} \\ \eta &= \sum_{i=1}^{n} \max\{0, x_i - u_i\} \\ \phi &= \sum_{i=1}^{n} \max\{0, \ell_i - x_i\} \end{aligned} \tag{6}$$

If a particle exceeds the upper bound or goes below the lower bound on any dimension, then it will be arranged as follows:

$$x_i = \begin{cases} \ell_i \\ u_i \\ x_i + \dfrac{u_i - x_i}{\delta^*}\eta \\ x_i + \dfrac{x_i - \ell_i}{\varepsilon^*}\phi \end{cases} \tag{7}$$

### 3.5 | Portfolio selection problem

Now we turn to the portfolio problem. The variables to be solved for $\langle x_1, x_2, \ldots, x_n \rangle$ are portfolio weights. Each particle $i$ at iteration $j$ contains $n$ such weights $\langle x_1, x_2, \ldots, x_n \rangle_k^l$. As we loop through each iteration, for each particle, its best history will be recorded (as personal best $p_k^l$). The best of the personal bests will be the global best $g_l$. At each iteration, the global best is updated, and convergence can be reached after a large enough number of iterations.

According to the Markowitz (1952) mean and variance model, risk is represented by the variance of the portfolio returns and the performance by the expected return. Investors wish to maximize their investment performance and minimize their risk. The problem can be formulated as follows:

$$\begin{aligned} \max \mu_P &= \sum_{i=1}^{n} x_i \mu_i \\ \min \sigma_P^2 &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \sigma_{ij} \end{aligned} \tag{8}$$

subject to

$$\sum_{i=1}^{n} x_i = 1 \tag{9}$$

where $n$ is the total number of assets, $x_i$ is the weight for the $i$th asset in the portfolio, $\mu_i = E[R_i]$ is the expected return of asset $i$ (and $R_i$ is the random return), $\sigma_{ij} = \mathrm{cov}[R_i, R_j]$ is the covariance between returns $i$ and $j$, $\sigma_P^2$ (measure for risk) is the variance of the portfolio return, and $\mu_P = E[R_P]$ is the expected return for the portfolio.

With data, individual means and variances/covariances are computed as follows:

$$\begin{aligned} \mu_i &= E[R_i] \\ &= \frac{1}{T} \sum_{t=1}^{T} R_{i,t} \end{aligned} \tag{10}$$

and

$$\begin{aligned} \sigma_{ij} &= \mathrm{cov}\left[R_i, R_j\right] \\ &= \frac{1}{T-1} \sum_{t=1}^{T} \left(R_{i,t} - \mu_i\right)\left(R_{j,t} - \mu_j\right) \end{aligned} \tag{11}$$

where $T$ is the total number of observations in the data.

An alternative (and equivalent) calculation of mean and variance of the portfolio can be computed more conveniently as follows:

$$\begin{aligned} \mu_P &= \frac{1}{T} \sum_{t=1}^{T} R_{P,t} \\ \sigma_P^2 &= \frac{1}{T-1} \sum_{t=1}^{T} \left(R_{P,t} - \mu_P\right)^2 \end{aligned} \tag{12}$$

where

$$R_{P,t} = \sum_{i=1}^{n} w_i R_{i,t}$$

The mean–variance problem is a multiobjective optimization problem (specifically, two objectives). This is a nontrivial optimization problem. No single solution exists that can simultaneously satisfy two objectives optimally. As a result, there are several optimal solutions, which are called Pareto optimal solutions. Among Pareto optimal solutions, none of the objective functions can be improved while not degrading other object functions. Then the efficient frontier is interpolated by all the Pareto optimal solutions.

Sharpe (1966) introduced a measure of risk-adjusted portfolio performance and proposed a reward-to-variability ratio (called the Sharpe ratio or Sharpe index), which is defined as follows:

$$I_{\text{Sharpe}} = \frac{\mu_P - R_f}{\sigma_P} \tag{13}$$

where $R_f$ is the return from the risk-free asset and

$$\sigma_P = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(R_{P,t} - \mu_P)^2} \tag{14}$$

The Sharpe ratio represents the extra return per unit risk. Using Equation 13 as the objective function, and subject to Equation 9, is a single-objective portfolio optimization problem. Since the portfolio weights can be negative and positive, this is an unrestricted portfolio optimization. When returns are normally distributed, a closed-form solution exists for the portfolio weights that can be the benchmark for this problem.

Restricted portfolio optimization places a constraint on the weights of assets. It can be formulated as follows:

$$\ell_i < x_i < u_i \tag{15}$$

where $\ell_i$ is the lower bound for the weight of the $i$th asset and $u_i$ is its upper bound.

If portfolio construction is restricted from short-selling, which means $\ell_i$ is always nonnegative, then portfolio optimization is defined as to maximize Equation 13 subject to Equations 9 and 15 and there exists no closed-form solution.

Traditionally, risk measure is represented as standard deviation of the portfolio return. It considers both upside and downside moves, which is unreasonable. Sortino (1994) introduced a new risk-adjusted measure of portfolio performance that only takes the downside move as the risk measure. It can be formulated as follows:

$$I_{\text{Sortino}} = \frac{\mu_P^* - R_f}{\sigma_P^*} \tag{16}$$

where

$$\sigma_P^* = \sqrt{\sum_{t=1}^{T}\left(R_{P,t}^* - \mu_P^*\right)^2} \tag{17}$$

and $R_P^*$ is the portfolio returns below the risk-free rate. Hence, $\mu_P^*$ and $\sigma_P^*$ can be viewed as the mean and standard deviation of those returns that fall below a certain target return. In other words, only those returns, but not all returns, are used in the mean and variance calculation.

These optimization problems are all nonconvex problems, which means any problem where the objective or any of the constraints are nonconvex. Such problems may have multiple feasible regions and multiple locally optimal points within each region. Consequently, as the number of assets increases, they become complex high-dimensional problems. In the rest of this paper, the performance and speed of SciPy-SLSQP, standard PSO and PSOCM will be tested on these optimization problems. SciPy is a popular, free and open-source Python library used for scientific computing and technical computing. In addition, SLSQP optimizer is a sequential least-squares programming algorithm that uses the traditional quasi-Newton method to obtain optimal solutions. Since it is a key algorithm included in SciPy packages, we choose SciPy-SLSQP as the benchmark for comparison testing.

## 4 | EXPERIMENTAL RESULTS

This part will test the performance and speed of SciPy-SLSQP, PSO, and PSOCM on five portfolios with different number of assets for RSharpe, URSharpe and RSortino portfolio optimization problems. Also, the effect of the number of particles on performance and speed will be tested. The experiment was done on a PC with Intel Core i7–7,700 3.6GHz and the algorithms are coded using Python 3.7.

We note that SciPy is an "off-the-shelf" library popular in the Python community. PSO and its variants are coded from scratch. The parameters of the PSO are set as $c_1 = 0.25$, $c_2 = 0.25$, $w = 0.9$. The two random numbers $r_1$ and $r_2$ in Equation 2 are uniform random numbers. Initial positions $p_k^0$ and velocities $V_k^0$ (for all particles $k = 1,\ldots,K$) are also uniform random numbers.

### 4.1 | Data

The data are daily stock price from S& 500 index components from October 2017 to October 2018. The S&P 500 contains 500 stocks, and the data are resampled into eight portfolios with different numbers of stocks. By the way, their components are overlapped and selected at random.

### 4.2 | Unrestricted Sharpe ratio

This section starts from the simplest case, the unrestricted Sharpe ratio index. Since there is a closed-form solution that can be used as a benchmark, we can examine the accuracies of the two numerical methods (SciPy and PSO) and, furthermore, the time required for the computation to achieve such an accuracy (note that the closed-form solution can be computed infinitely fast).

The experiment used portfolios with 10, 25, 50, 100, and 200 stocks.[5] Figure 1 shows that the SciPy algorithm performs really well on this optimization problem. The Sharpe ratios obtained from SciPy are same as the values of the closed-form solutions, which are 3.390, 5.404, 8.971, 14.296, and 30.471 respectively for portfolios with 10, 25, 50, 100, and 200 stocks. On the other hand, PSO and

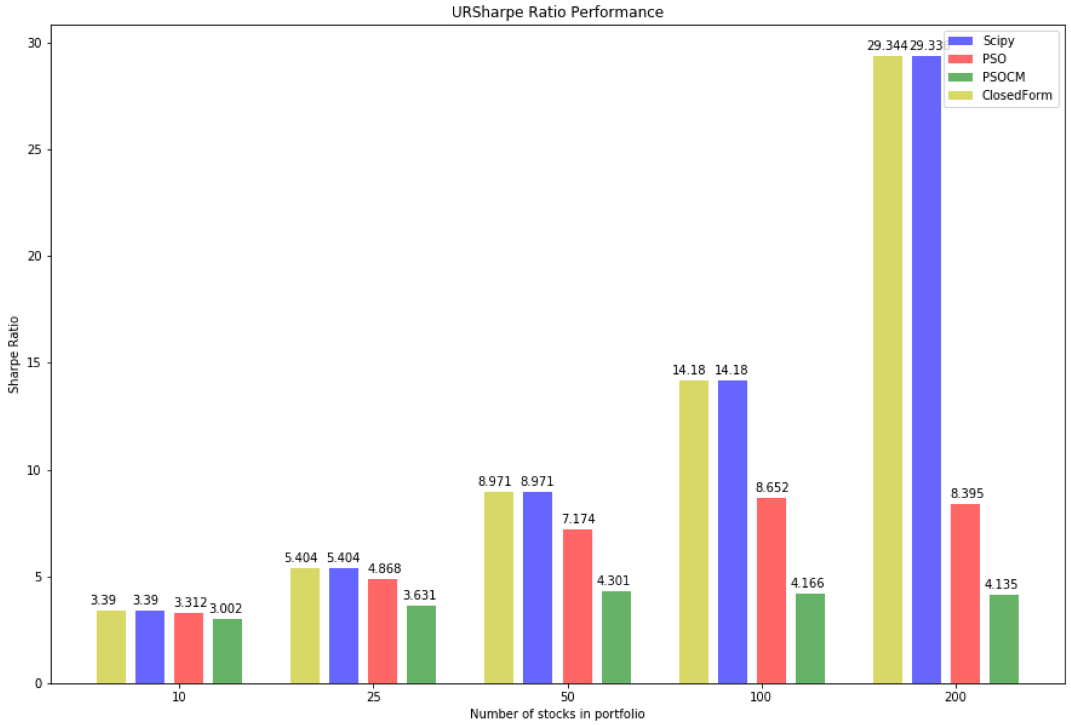
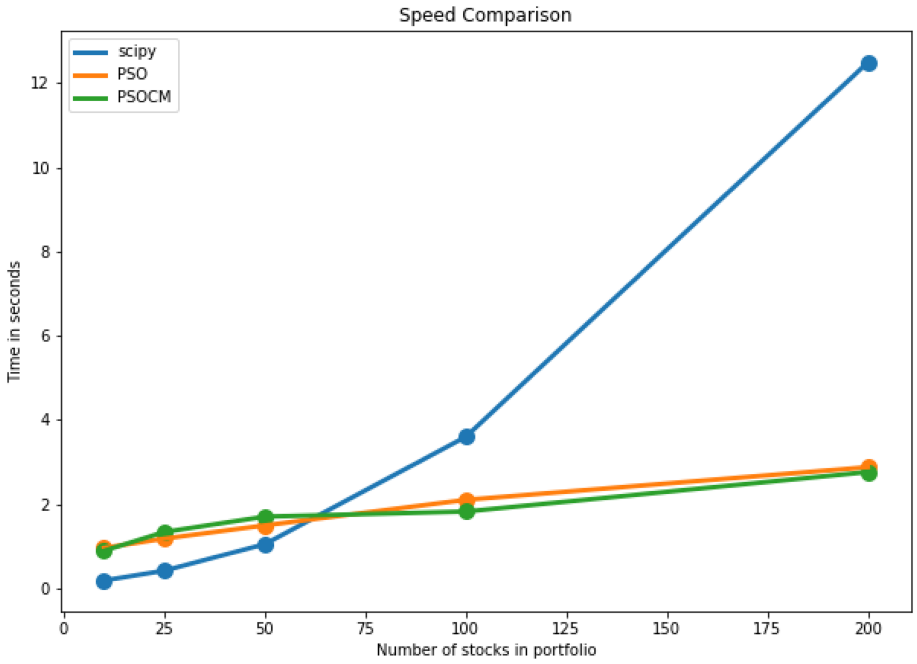



**FIGURE 1** Performance comparison of unrestricted Sharpe ratio

**FIGURE 2** Speed comparison of unrestricted Sharpe ratio




PSOCM perform poorly, especially when the number of stocks is large. For example, for 10 and 25 stocks, PSO produces Sharpe ratios of 3.312 and 4.877, respectively, both of which are slightly lower than the correction solutions. However, in the case of 200 stocks, PSO produces a Sharpe ratio of 10.612, which is substantially lower than the correct solution. PSOCM is even worse than PSO. This implies that PSO (or PSOCM) is not necessarily superior to a popular optimization tool such as SciPy. In fact, as we can see, it is significantly worse than SciPy in the given problem.

Certainly, it is clear that the reason SciPy can produce such an accurate approximation (no error in the third decimal digit) is because the Sharpe ratio index is differentiable in portfolio weights. As a result, it can quickly use first and second derivatives to reach a better result in the next iteration, and it can converge speedily. On the contrary, PSO relies on particles moving around in a heuristic manner to find 200 weights, which is unintelligent and inefficient. Hence, it is not surprising that the results are not as good.

Figure 2 presents the time used by the various methods. The time SciPy spends is exponentially increasing as the number of stocks increases. For example, SciPy spends less than 0.5 s in the cases of 10 and 25 stocks but more than 8 s in the case of 200 stocks. This is an increase in time required to arrive at an accurate solution of about 20 times. On the other hand, the time spent by PSO and PSOCM only increases linearly (from 1 s to 2 s) between 10 stocks and 200 stocks. PSO and PSOCM spend almost the same amount time.

From this simple case, we conclude that the PSO family algorithms perform badly on the unrestricted Sharpe ratio optimization problem. A gradient-based optimization method, like SciPy, performs quite well and its result is almost the same as the closed-from solution.

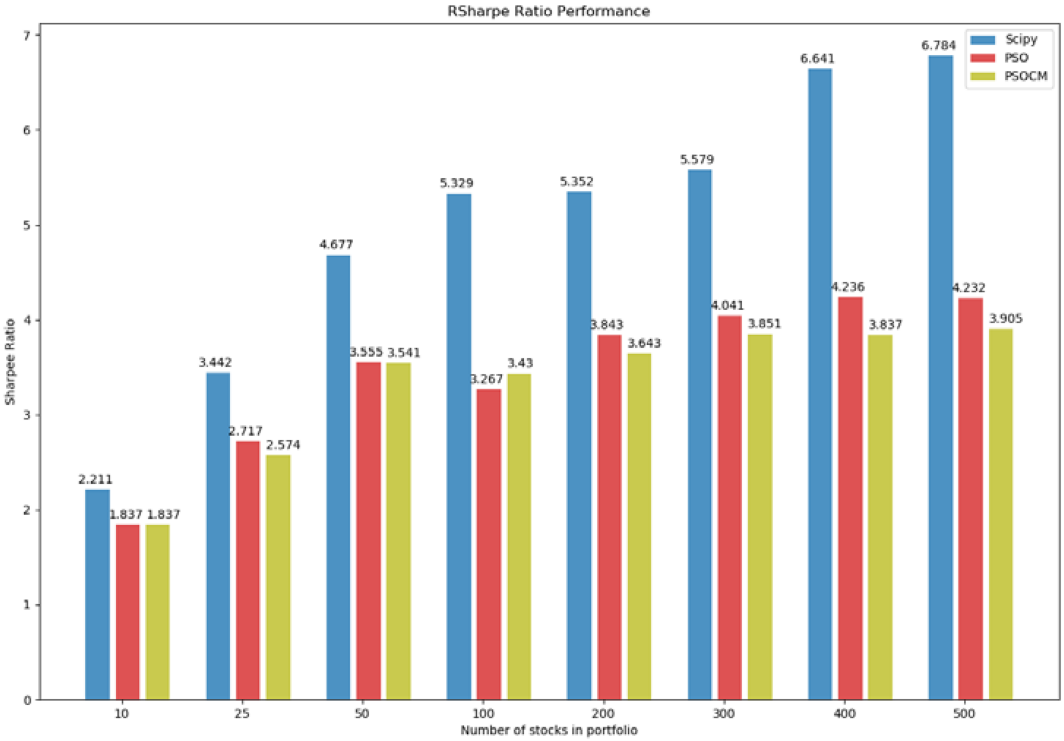



**FIGURE 3** Performance comparison of restricted Sharpe ratio

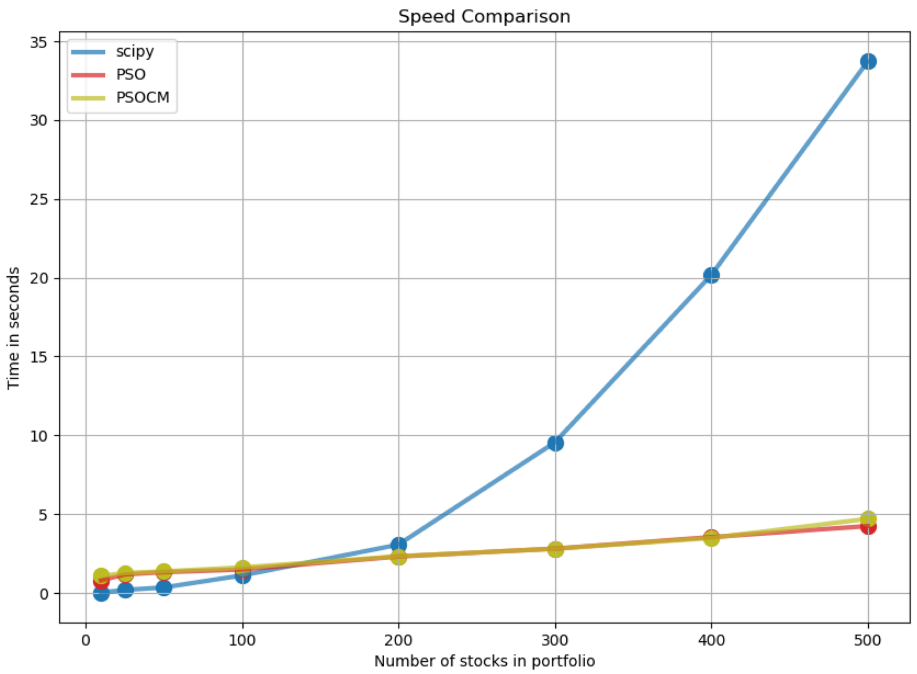



**FIGURE 4** Speed comparison of restricted Sharpe ratio

### 4.3 | Restricted Sharpe ratio

In the previous section, the weights could be negative in order to short certain stocks. In reality, however, many investors or institutions are restricted to construct a long-only portfolio. Hence, the long-only constraint is applied to the Sharpe ratio optimization. Moreover, there is no closed-form solution for this restricted optimization problem.

For this part, the data are resampled into eight portfolios with 10, 25, 50, 100, 200, 300, 400, and 500 stocks. All eight of the portfolios are used to calculate the results for comparison. As Figure 3

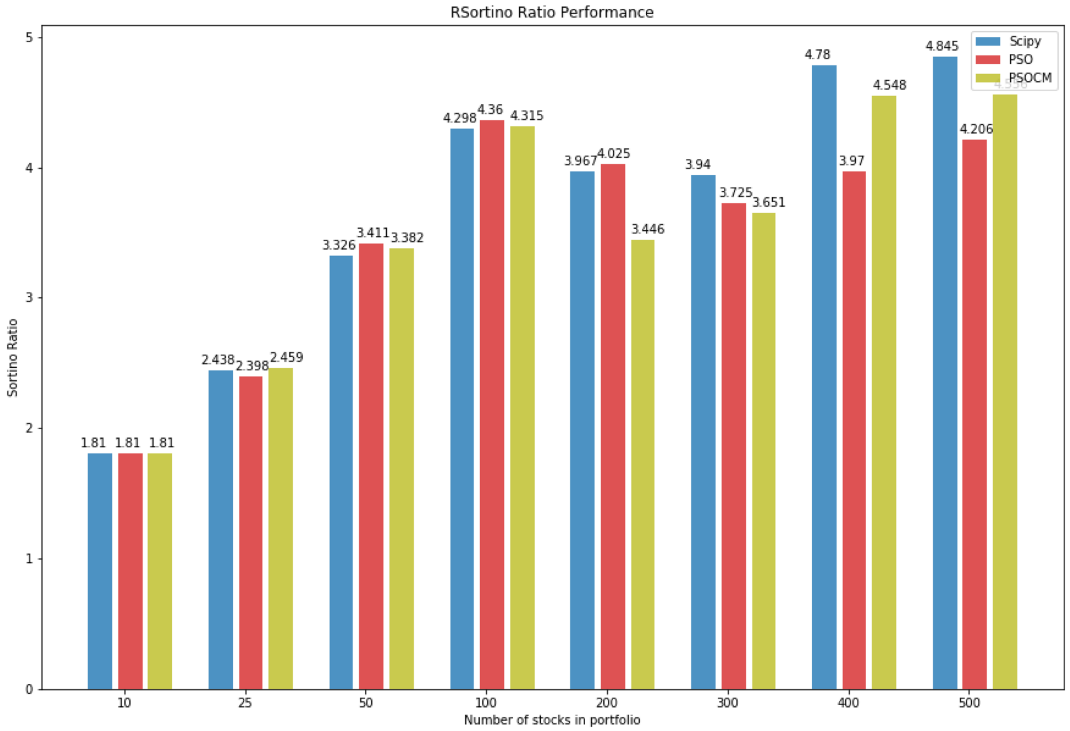



**FIGURE 5** Performance comparison of Sortino ratio

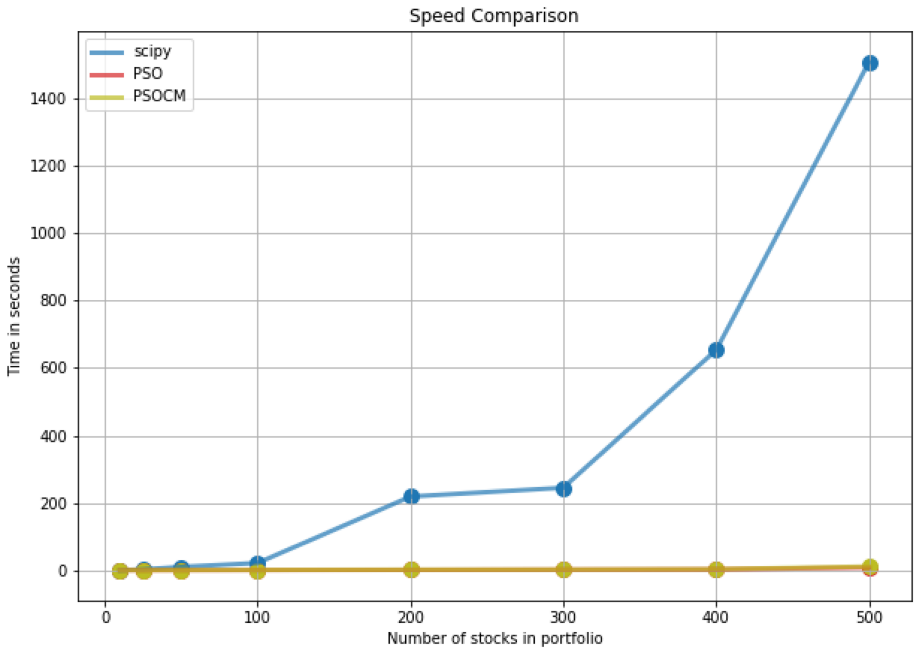



**FIGURE 6** Speed comparison of Sortino ratio

shows, SciPy still dominates the other methods across the test cases with different numbers of stocks. This can be interpreted that the Sharpe ratio is a continuous function, so a gradient-based optimization method like SciPy is very effective.

However, Figure 4 demonstrates that the SciPy speed slows down dramatically as the number of stocks increases. In the case of a portfolio with 500 stocks, the time spent by SciPy is almost seven times as much as PSO or PSOCM.

## 4.4 | Sortino ratio

Since variance is not an appropriate approach to measure the risk, the semi-variance introduced by Sortino (1994) could be a better alternative. That is because this risk measure is a modification of the Sharpe ratio and penalizes only those returns falling below a user-specified target or required rate of return, whereas the Sharpe ratio penalizes both upside and downside volatility equally. The semi-deviation

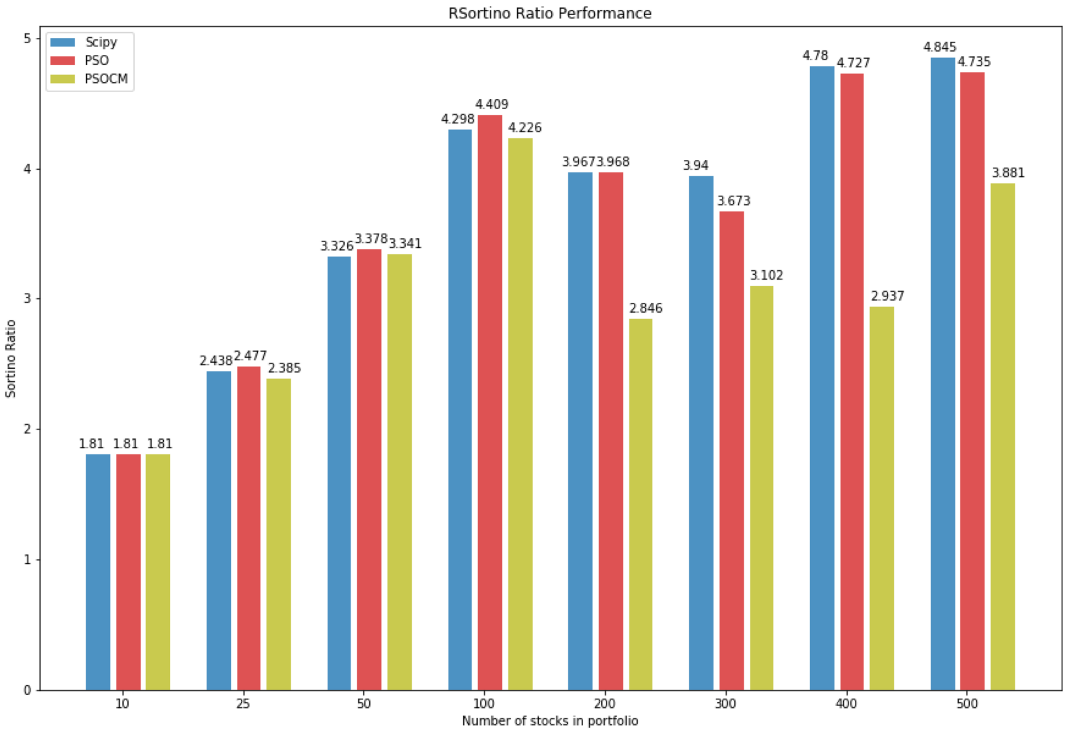



**FIGURE 7** Performance comparison of Sortino ratio (linear decreasing)

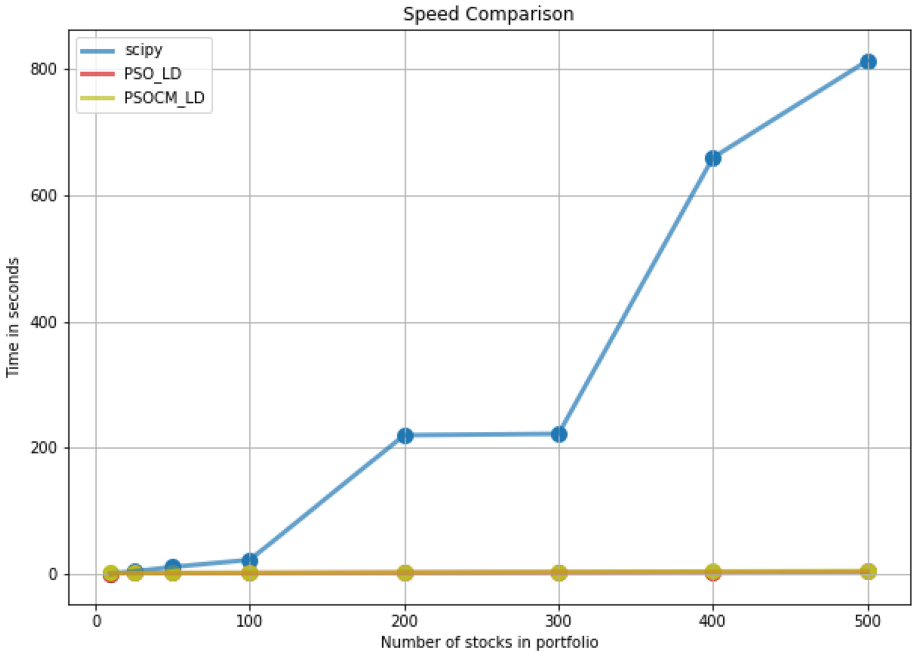



**FIGURE 8** Speed comparison of Sortino ratio (linear decreasing)

**TABLE 1** Restricted Sortino statistics for various asset-size portfolios after 10 trials

| | Mean | *SD* | Min | Max |
|---|---|---|---|---|
| **10-assets portfolio** | | | | |
| SciPy speed (s) | 0.8342 | 0.0166 | 0.8027 | 0.8957 |
| SciPy performance | 1.8102 | 0.0000 | 1.8102 | 1.8102 |
| PSOCM speed (s) | 1.3521 | 0.0271 | 1.2974 | 1.4225 |
| PSOCM performance | 1.8100 | 0.0002 | 1.8092 | 1.8102 |
| PSO speed (s) | 1.1844 | 0.0758 | 0.9662 | 1.3480 |
| PSO performance | 1.8102 | 0.0002 | 1.8095 | 1.8102 |
| **25-assets portfolio** | | | | |
| SciPy speed (s) | 3.1976 | 0.0662 | 3.1060 | 3.5267 |
| SciPy performance | 2.4380 | 0.0000 | 2.4380 | 2.4380 |
| PSOCM speed (s) | 1.3252 | 0.0285 | 1.2583 | 1.3911 |
| PSOCM performance | 2.4537 | 0.0241 | 2.2882 | 2.4682 |
| PSO speed (s) | 1.1546 | 0.0306 | 1.1045 | 1.2517 |
| PSO performance | 2.4324 | 0.0594 | 2.2831 | 2.4801 |
| **50-assets portfolio** | | | | |
| SciPy speed (s) | 10.97229 | 0.18991 | 10.71678 | 12.04511 |
| SciPy performance | 3.32609 | 0.00000 | 3.32609 | 3.32609 |
| PSOCM speed (s) | 1.53058 | 0.05003 | 1.42520 | 1.65748 |
| PSOCM performance | 3.37902 | 0.01451 | 3.34480 | 3.40615 |
| PSO speed (s) | 1.29709 | 0.03399 | 1.23508 | 1.36360 |
| PSO performance | 3.33191 | 0.14434 | 2.77057 | 3.42075 |
| **100-assets portfolio** | | | | |
| SciPy speed (s) | 22.36182 | 0.18363 | 21.98680 | 22.69867 |
| SciPy performance | 4.29830 | 0.00000 | 4.29830 | 4.29830 |
| PSOCM speed (s) | 1.90289 | 0.07461 | 1.72031 | 2.08328 |
| PSOCM performance | 4.34393 | 0.02481 | 4.25329 | 4.39807 |
| PSO speed (s) | 1.57922 | 0.05952 | 1.47330 | 1.77109 |
| PSO performance | 4.35923 | 0.03975 | 4.21532 | 4.40892 |
| **200-assets portfolio** | | | | |
| SciPy speed (s) | 220.0648 | 2.5146 | 218.0249 | 236.4216 |
| SciPy performance | 3.9673 | 0.0000 | 3.9673 | 3.9673 |
| PSOCM speed (s) | 2.6051 | 0.1139 | 2.2703 | 2.8725 |
| PSOCM performance | 3.6747 | 0.1668 | 3.1985 | 3.9235 |
| PSO speed (s) | 2.2295 | 0.0470 | 2.1086 | 2.3445 |
| PSO performance | 3.7347 | 0.2005 | 3.2400 | 4.0324 |
| **300-assets portfolio** | | | | |
| SciPy speed (s) | 222.0061 | 1.0849 | 220.4933 | 228.1118 |
| SciPy performance | 3.9404 | 0.0000 | 3.9404 | 3.9404 |
| PSOCM speed (s) | 3.2374 | 0.1471 | 2.9475 | 3.5476 |
| PSOCM performance | 3.6190 | 0.1916 | 3.1637 | 3.9109 |
| PSO speed (s) | 2.8569 | 0.0453 | 2.7036 | 2.9710 |
| PSO performance | 3.6560 | 0.2026 | 3.0186 | 3.9369 |
| **400-assets portfolio** | | | | |
| SciPy speed (s) | 659.5646 | 5.6992 | 651.1509 | 685.9767 |
| SciPy performance | 4.7805 | 0.0000 | 4.7805 | 4.7805 |
| PSOCM speed (s) | 3.8912 | 0.2059 | 3.4765 | 4.3502 |

(Continues)

**TABLE 1** (Continued)

| | Mean | *SD* | Min | Max |
|---|---|---|---|---|
| PSOCM performance | 4.4181 | 0.3049 | 3.5762 | 4.7707 |
| PSO speed (s) | 3.4603 | 0.1306 | 3.2948 | 4.2764 |
| PSO performance | 4.4836 | 0.3020 | 3.5912 | 4.9523 |
| **500-assets portfolio** | | | | |
| SciPy speed (s) | 816.7181 | 4.5883 | 810.3069 | 831.2686 |
| SciPy performance | 4.8445 | 0.0000 | 4.8445 | 4.8445 |
| PSOCM speed (s) | 4.5562 | 0.2333 | 4.0419 | 5.0660 |
| PSOCM performance | 4.4615 | 0.2268 | 3.5452 | 4.7422 |
| PSO speed (s) | 4.0683 | 0.0865 | 3.9083 | 4.3518 |
| PSO performance | 4.4258 | 0.3437 | 3.4324 | 4.9032 |

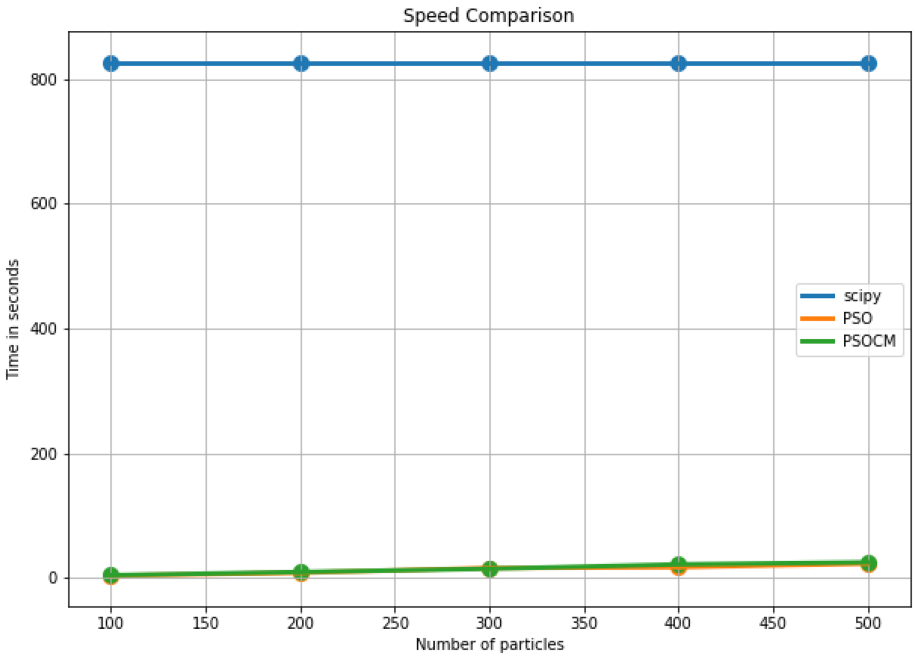



**FIGURE 9** Restricted Sortino speed comparison with increasing swarm size on 500-asset portfolio

calculation can be described as follows: first, using probability weights and asset returns to calculate portfolio return series; second, filtering the portfolio returns under target return; finally, calculating the square root of the probability-weighted squared below-target returns. Since when the weight changes a litter bit, a portfolio return may be filtered out or included in the filtered portfolio return series. Then, the semi-deviation would be discrete instead of continuous, which makes the Sortino ratio much harder to optimize.

According to Figure 5, PSO and PSOCM perform as well as SciPy does, and sometimes they even outperform SciPy. Taking speed into consideration, as Figure 6 shows, PSO and PSOCM run faster than SciPy. In some way, SciPy is a well-established and highly optimized package, and most of its internal algorithms are coded in C++, which is a high-level programming language. If PSO and PSOCM can be coded in C++, they could even be superior in speed.

The PSO and PSOCM with linear decreasing coefficients technique are also implemented for comparison. They are denoted as PSOLD and PSOCMLD, respectively. As Figure 7 shows, the linear decreasing technique does not improve the performance. It even deteriorates the performance. With regard to the speed aspect, Figure 8 demonstrates that this technique almost cannot slow down the speed.

## 4.5 | PSO experiments

Because there is random selection in the entire PSO calculation process, the optimization results would be different at each trial. In order to enhance the reliability of the empirical results, 10 trials of PSO calculation are implemented on 10-asset, 25-asset, 50-asset, 100-asset,

**TABLE 2** Restricted Sortino statistics on 500-asset portfolio with 500 or 1,000 particles under 10 trials

| | Mean | *SD* | Min | Max |
|---|---|---|---|---|
| **500 particles** | | | | |
| PSO speed (s) | 15.6964 | 0.6840 | 15.1196 | 17.5725 |
| PSO performance | 4.5148 | 0.2713 | 4.1702 | 4.8177 |
| PSOCM speed (s) | 18.3315 | 0.6169 | 17.3582 | 19.3303 |
| PSOCM performance | 4.5260 | 0.1823 | 4.0863 | 4.6697 |
| **1,000 particles** | | | | |
| PSO speed (s) | 30.3911 | 0.9536 | 29.3132 | 31.8532 |
| PSO performance | 4.6702 | 0.1931 | 4.3178 | 4.8956 |
| PSOCM speed (s) | 35.3760 | 1.2956 | 33.6119 | 37.0420 |
| PSOCM performance | 4.5248 | 0.2737 | 3.8721 | 4.7756 |

200-asset, 300-asset, 400-asset, and 500-asset randomly selected portfolios. The results are summarized from Table 1. In portfolio asset size, both performance (restricted Sortino ratio, unshaded rows) and speed (in seconds, shaded rows) are reported.

From Table 1, we can find that SciPy requires (exponentially) more time to implement as the size of the portfolio grows. It needs only a fraction of a second (mean speed 0.8342 s) for the 10-asset portfolio but over 13 min (mean speed 816.7181 s) for the 500-asset portfolio. On the extreme contrary, the times required for PSO methods do not increase much (in some cases they are even faster). In general, SciPy has an advantage in performance (marginally), but it pays a high price in speed (substantially). As we mentioned earlier, this is due to the fact that nondifferential functions (e.g., restricted Sortino ratio) contribute to AI-type methods.

Regarding the two PSO methods, PSOCM does not perform on average better than standard PSO. PSOCM performs better than PSO when the portfolio is smaller. Yet, in such cases, SciPy may have a better advantage in speed.

Next, we investigate the performance of the size of swarm. We study the Sortino ratio of a portfolio with 500 assets. The relationship between number of particles, performance, and speed are examined. Figure 9 illustrates the relationship between speed and the number of particles. The computation time increases linearly as the number of particles increases.

The performance is reported in Table 2. It can be shown that the performance of PSO improves as the number of particles increases. From these results, we can conclude that PSO is significantly affected by the number of particles used. When the number of particles reaches 1,000, some trials even outperform the SciPy results. However, the improvement of PSOCM is not that significant compared with PSO.

## 5 | CONCLUSION

We compared two AI-based optimization algorithms, PSO and PSOCM, with a popular numerical algorithm available in Python, SciPy, on portfolio construction subject to various constraints. When optimizing this problem based on the Sharpe ratio, SciPy clearly outperforms PSO and PSOCM. When the problem does not take short sale restriction into account, SciPy can achieve the same performance as a closed-form solution. This is because the Sharpe ratio is a continuous function, and hence a gradient-based method like SciPy is very suitable.

However, the Sharpe ratio is not a good measure when dealing with portfolio construction. That is because the Sharpe ratio equally penalizes both downside and upside volatility, and yet ignoring the fact that upside volatility is good for investors. Therefore, we consider the Sortino ratio, which is a better alternative as it only considers downside volatility as risk. However, the Sortino ratio function is discontinuous, which makes the optimization much harder.

Our experimental results confirm that PSOCM and PSO are faster than SciPy in a high-dimensional space and can achieve better results. PSO is good at searching optimal solutions, and its performance could be better than SciPy.

Between the two PSO methods, the results generated by PSO are more volatile than the results generated by PSOCM while we conduct a10-trial examination on portfolios with 500 stocks. Also, the effect of the number of particles is also considered in the experiment. The results illustrate that computation time increases linearly and the performance is improved for both PSO and PSOCM as the number of particles increases. However, PSO improves more significantly than PSOCM when the number of particles increases.

Future research might be conducted to solve the problem about portfolio construction subject to more complicated or practical constraints. For example, if the investment in each industry is restricted or the liquidity requirement is considered.

## ORCID

*Shih-Kuo Yeh* https://orcid.org/0000-0002-3741-4712

## ENDNOTES

[1] For example, see Zhu et al. (2011).

[2] Though not adopted in this paper, PSO can be "GPUized." In other words, the dimensionality is not a limitation to PSO as it can be dealt with by adding more particles that can be operated in parallel on multiple graphics processing units.

[3] This is similar to a minimax problem in a linear program. However, the problem here is nonlinear and PSO is used at two levels: one for minimum and one for maximum.

[4] As it will become clear, that element is a portfolio weight. Each particle $k$ determines what the optimal portfolio weights are. At each iteration $l$, every particle moves around to improve its objective function value.

[5] We cannot perform exercises on portfolios with more than 200 stocks because the closed-form solution cannot be computed (owing to the difficulty in inverting the variance–covariance matrix).

## REFERENCES

Cagcag Yolcu, O., & Alpaslan, F. (2018). Prediction of TAIEX based on hybrid fuzzy time series model with single optimization process. *Applied Soft Computing*, *66*, 18–33. https://doi.org/10.1016/j.asoc.2018.02.007

Chang, T.-J., Meade, N., Beasley, J. E., & Sharaiha, Y. M. (2000). Heuristics for cardinality constrained portfolio optimization. *Computers & Operations Research*, *27*, 1271–1302. https://doi.org/10.1016/S0305-0548(99)00074-X

Corazza, M., Fasano, G., & Gusso, R. (2013). Particle swarm optimization with non-smooth penalty reformulation, for a complex portfolio selection problem. *Applied Mathematics and Computation*, *224*, 611–624. https://doi.org/10.1016/j.amc.2013.07.091

Cura, T. (2009). Particle swarm optimization approach to portfolio optimization. *Nonlinear Analysis: Real World Applications*, *10*, 2396–2406. https://doi.org/10.1016/j.nonrwa.2008.04.023

Deng, G. F., Lin, W. T., & Lo, C. C. (2012). Markowitz-based portfolio selection with cardinality constraints using improved particle swam optimization. *Experts Systems with Application*, *39*, 4558–4566. https://doi.org/10.1016/j.eswa.2011.09.129

Gao, W., & Su, C. (2020). Analysis of earnings forecast of blockchain financial products based on particle swarm optimization. *Journal of Computational and Applied Mathematics*, *372*, 112724. https://doi.org/10.1016/j.cam.2020.112724

Golmakani, H. R., & Fazel, M. (2011). Constrained portfolio selection using particle swarm optimization. *Experts Systems with Application*, *38*, 8327–8335. https://doi.org/10.1016/j.eswa.2011.01.020

Jamous, R. A., Tharwat, A. A., Seidy, E. E., & Bayoumi, B. I. (2015). A new particle swarm with center of mass optimization. *International Journal of Engineering Research & Technology*, *4*(5), 312–317.

Kendall, G., & Su, Y. (2005). A practical swarm optimization approach in the construction of optimal risky portfolios. *Artificial Intelligence and Applications*, *453*, 140–145.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95—International conference on neural networks* (Vol. 4, pp. 1942–1948). Piscataway, NJ: IEEE Press.

Kennedy, J., & Eberhart, R. (1997). A discrete binary version of the particle swarm algorithm. In *Proceedings of 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* (Vol. 5, pp. 4104–4108).

Lahmiri, S. (2016). Interest rate next-day variation prediction based on hybrid feedforward neural network, particle swarm optimization, and multiresolution techniques. *Physica A: Statistical Mechanics and its Applications*, *444*, 388–396. https://doi.org/10.1016/j.physa.2015.09.061

Lahmiri, S. (2017). Comparing variational and empirical mode decomposition in forecasting day-ahead energy prices. *IEEE Systems Journal*, *11*(3), 1907–1910. https://doi.org/10.1109/JSYST.2015.2487339

Lahmiri, S. (2018). Minute-ahead stock price forecasting based on singular spectrum analysis and support vector regression. *Applied Mathematics and Computation*, *320*, 444–451. https://doi.org/10.1016/j.amc.2017.09.049

Lu, F.-Q., Huang, M., Ching, W.-K., & Siu, T. K. (2013). Credit portfolio management using two-level particle swarm optimization. *Information Sciences*, *237*, 162–175. https://doi.org/10.1016/j.ins.2013.03.005

Marinakis, Y., Marinaki, M., Doumpos, M., & Zopounidis, C. (2009). Ant colony and particle swarm optimization for financial classification problems. *Expert Systems with Applications*, *36*(7), 10604–10611. https://doi.org/10.1016/j.eswa.2009.02.055

Markowitz, H. (1952). Portfolio selection. *Journal of Finance*, *7*, 77–91.

Pradeepkumar, D., & Ravi, V. (2017). Forecasting financial time series volatility using particle swarm optimization trained quantile regression neural network. *Applied Soft Computing*, *58*, 35–52. https://doi.org/10.1016/j.asoc.2017.04.014

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of Business*, *39*, 119–138.

Sortino, F. A. (1994). Performance measurement in a downside risk framework. *Journal of Investing*, *3*, 59–64. https://doi.org/10.3905/joi.3.3.59

van den Bergh, F., & Engelbrecht, A. P. (2006). A study of particle swarm optimization particle trajectories. *Information Sciences*, *176*(8), 937–971.

Zhu, H., Wang, Y., Wang, K., & Chen, Y. (2011). Particle swarm optimization (PSO) for the constrained portfolio optimization problem. *Experts Systems with Application*, *38*, 10161–10169. https://doi.org/10.1016/j.eswa.2011.02.075

**How to cite this article:** Chen, R.-R., Huang, W. K., & Yeh, S.-K. (2021). Particle swarm optimization approach to portfolio construction. *Intelligent Systems in Accounting, Finance and Management*, *28*(3), 182–194. https://doi.org/10.1002/isaf.1498