Stock Portfolio Optimization
An Algorithmic Improvement in Python's View

Shuting GU, Anshan HE, Meiyi HE, Jiayu QIU, Weiyang WEN

Center for Data Science, New York University
May 2019

Abstract

We present a systemic way to improve convex optimization performance in Python (`SciPy`) in the context of stock portfolio optimization. Interior Point Methods (IPM) are shown to have improved speed and accuracy. Additional optimization is achieved using CUDA (GPU) acceleration and reduced function calls.

1 Introduction and Motivation

Portfolio optimization is one of the central topics in the financial world. It is the process of selecting the best way to invest money into a given basket of stocks with certain requirements. Mathematically, it can be formulated into a constrained convex optimization problem and can be solved by standard tools like `SciPy` optimizer or the like. However, this kind of traditional optimization algorithms can be fairly slow especially when the number of stocks n becomes large.

From python's view, attempts to improve the speed of this process seems to be daunting. Firstly, core parts of `SciPy` are already implemented in C, C++ or Fortran, so wrapping the process using `Cython` would not result in much performance improvements. In addition, parallel processing cannot be easily applied due to the sequential nature of optimization.

Here we propose a way to improve the performance of convex optimization performance in Python (`SciPy`). The core idea is that convex problem can be reformulated into a linear one by the famous KKT condition, then it is solved by Newton's method. This is called the Interior Point Method (IPM). Another benefit for the IPM method is that CUDA (GPU) accelation, along with other improvements, are now possible.

This paper is organized as follows. Section 2 and 3 describe the data generation process and the formulation of the problem. Section 4 shows the standard way to solve it using Python's `SciPy` package. Section 5 and 6 describe the IPM method and its improvements. Section 7 and 8 show the results and conclusions.

2 Data Generation

We test our program using simulated returns. Returns are assumed following a multivariate normal distribution, where we incorporate eigenvalue structures of real world returns into the covariance matrix to make it more resemble the real world data. For each portfolio, a $2500 \times n$ matrix is generated and represents 10 years of trading data of n stocks. Then the sample means μ and the sample covariance matrix Q are calculated and saved into pickle files.

3 Problem Formulation

In general, portfolio optimization problem can be formulated mathematically in the following way:

$$\begin{aligned} \min_x \; & x^T Q x + c^T x \\ \text{s.t. } & Ax = b \\ & x \geq 0 \end{aligned}$$

where x is the weight vector we want to solve for. Multiple equality constraints can be in-

corporated by changing matrix A and b. Additional inequality constraints can be added as well. This is a standard constrained optimization problem.

4 Quadratic Optimization Using `SciPy`

Using the `SciPy` package is the most straightforward way to solve the optimization problem and it is easy and common to implement. We set the performance of `scipy.optimize` as our baseline.

We started with a simple case to test if `SciPy` optimizer can give a correct output. In this setting, we want to solve:

$$\min_x x^T Q x$$
$$\text{s.t. } \mu^T x = r$$

This is equivalent to finding the portfolio with the minimum risk given only a return constraint. This problem has a closed form for the analytical solution, $x^* = \frac{Q^{-1}\mu}{\mu^T Q^{-1}\mu} r$, which can be used as a benchmark of the performance. Compared with the true weights, the results given by `SciPy` have overall acceptable performance.

In the second test case, we added both equality and inequality constraints to make it more resemble the real-world requirements. We included some new requirements such as using up all the money and no short-selling. Mathematically, let

$$A = \begin{bmatrix} \mu \\ \mathbb{1} \end{bmatrix}, b = \begin{bmatrix} r \\ \mathbb{1} \end{bmatrix}, c = 0, x \geq 0$$

This one does not have a closed form for the analytical solution so we can only solve it by `SLSQP` solver.

But the shortcoming of using `SciPy` is clear. The processing time will grow quadratically as the number of stocks in the portfolio increases linearly. This makes it not appropriate for the real world stock portfolio optimization, which includes much more stocks.

5 Interior Point Method (IPM)

In the following sections, we firstly discuss how we transfer the quadratic problem into a linear problem by using primal-dual formulation and the Karush-Kuhn-Tucker (KKT) conditions. And then we apply the IPM to efficiently solve the linear programming problem.

5.1 The KKT conditions

As a standard way of solving a constrained optimization problem, we start with the Lagrangian function:

$\mathcal{L}(x, y, s) = x^T Q x + c^T x + y^T (Ax - b) - s^T x$

where y, s are Lagrange multipliers. The covariance matrix Q is positive semi-definite so the problem is convex. Therefore, any x satisfying the following conditions is the optimal solution to the optimization problem in section 3:

1. primal constraints: $Ax = b, x_i \geq 0, \forall i$;
2. dual constraints: $s \geq 0$;
3. complementary slackness: $s_i x_i = 0, \forall i$;
4. gradient of Lagrangian w.r.t. x vanishes: $2Qx + c - s + A^T y = 0$.

These are called the KKT conditions, and it can be written in a matrix form:

$$F(x^*, y^*, s^*) = \begin{bmatrix} 2Qx^* + c - s^* + A^T y^* \\ Ax^* - b \\ X^* S^* \mathbb{1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, s \geq 0, x \geq 0$$

where X^* and S^* are diagonal matrices with the entries of the x^* and s^* vectors, respectively, on the diagonal. Noted that this now becomes a linear programming problem. If we discard the nonnegativity constraints, we can apply Newton's method to solve the system of equations.

5.2 The Central Path

However, the existence of nonnegativity constraints makes the above problem difficult to be solved. So in this section, we introduce the central path method, which can help us to find the optimal solution.

We firstly define the feasible set,

$F := \{(x, y, s) : 2Qx + cs + Ay = 0, Ax = b, x \geq 0, s \geq 0\}$, and the relative interior of the feasible set, $F^0 := \{(x, y, s) : 2Qx + cs + Ay = 0, Ax = b, x > 0, s > 0\}$.
The IPM will firstly generate a sequence of points $(x^k, y^k, s^k) \in F^0$. The sequence will gradually approach to satisfy the third block in the system of equations, which is $x_i s_i = 0, \forall i$. The central path C is a trajectory in the relative interior of the feasible region F^0, which can be defined as $C = (x_\tau, y_\tau, s_\tau) : \tau > 0$. The trajectory is parameterized by a scalar $\tau > 0$, and the point $(x_\tau, y_\tau, s_\tau) \in C$ on the central path is the unique solution of the following system:

$$F(x_\tau, y_\tau, s_\tau) = \begin{bmatrix} 2Qx_\tau + c - s_\tau + A^T y_\tau \\ Ax_\tau - b \\ X_\tau S_\tau \mathbb{1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \tau \mathbb{1} \end{bmatrix}, s_\tau > 0, x_\tau > 0$$

As $\tau \to 0$, (x_τ, y_τ, s_τ) converges to an optimal solution.
However, when $\tau > 0$, finding the point (x_τ, y_τ, s_τ) is still difficult. Instead, we can solve it by finding an approximation to a central point (x_τ, y_τ, s_τ). This means we start from the point (x^k, y^k, s^k) which is close to the central point $(x_{\tau^k}, y_{\tau^k} s_{\tau^k})$ and then find another point $(x^{k+1}, y^{k+1}, s^{k+1})$, which is close to another central point $(x_{\tau^{k+1}}, y_{\tau^{k+1}} s_{\tau^{k+1}})$.
By always setting $\tau^{k+1} < \tau^k$, the iterations will approximate the optiomal solution when k increases. For the k-th point $(x^k, y^k, s^k) \in F^0$ in the iteration, we define $\tau^k = (x^k)^T (s^k)/n$. Choosing $\sigma^k \in (0, 1)$, we can further define $\tau^{k+1} = \sigma^k \tau^k$. Then, the k+1-th point $(x_{\tau^{k+1}}, y_{\tau^{k+1}} s_{\tau^{k+1}}) \in F^0$ can be sloved by approximately solving the central point corresponding to τ^{k+1}, which can be obtained by moving from (x^k, y^k, s^k) along the Newton's direction.
As one of the steepest descent methods that can improve convergence, Newton's method is applied here to solve the problem. Given the iteration formula $x_{k+1} = x_k - \eta_k \nabla F(x_k)^{-1} F(x_k)$, we plug in our equations, choose a step size η_k and let $\eta_k > 0$ to ensure that after updating x and s remain positive. In summary, the approximation of central points can be updated as follows:

$$\begin{bmatrix} x^{k+1} \\ y^{k+1} \\ s^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ y^k \\ s^k \end{bmatrix} - \eta_k \begin{bmatrix} 2Q & A^T & -I \\ A & 0 & 0 \\ S_k & 0 & X_k \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ XS\mathbb{1} - \sigma\tau\mathbb{1} \end{bmatrix}$$

5.3 Interior Point Generation

In order to apply IPM, we need to firstly find a starting point in the interior of the feasible set, i.e. $(x_0, y_0, s_0) \in F^0$. In this section, we discuss the two methods we have tried.
There is a general method to find a starting point. If we only consider the first equality constraint, we can use `SciPy` to find a satisfying x_0. Then using the conditions of F^0, we can arbitrarily find the corresponding y_0 and s_0, which satisfy all the conditions. Here, we use a higher tolerance for termination to make the x_0 can be found more quickly.
However, there is an easier and faster approach to find x_0. One of the equality constraints is the weighted average returns of all stocks should be equal to a pre-determined objective return. In order to achieve it, the value of the objective return should be no less than the minimum return of all stocks and no greater than the maximum one. This means that using these two stocks with the minimum return and the maximum return, we can construct a suitable pair of weights to achieve our goal. Since all weights for all stocks need to be positive, we divide 99% weight between these two selected stocks and assign 1% weight to the rest.

6 IPM with CUDA

After applied the IPM, the original problem becomes a linear programming problem thus we tried to use CUDA to see if there is any additional speed up.
CUDA is a parallel computing platform and application programming interface (API)

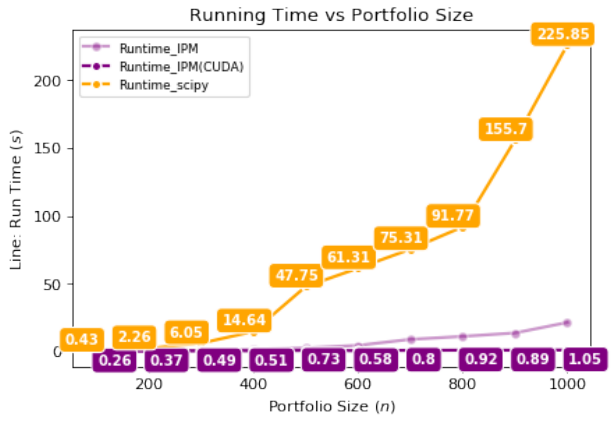
model developed by NVIDIA. CUDA computation platform allows programmers to use CPU to sets up the parameters and data for the computation and GPU to do the hard computation. Compared with traditional CPUs that commonly only have 4 or 8 arithmetic logic units (ALUs), GPU has 1000s of ALUs so it is good for repetitive arithmetic operations.

To make the result reproducible across different environment and platforms, we used Google Colaboratory to run GPU tasks. This is a free Jupyter notebook environment that allows people to access powerful computing resources (such as GPUs). To make the most use of GPU, we further optimize our computation without generating a new matrix during each iteration but modifying the original matrix in place to reduce communication overhead. This optimization allows us to use less CPU power during computation.

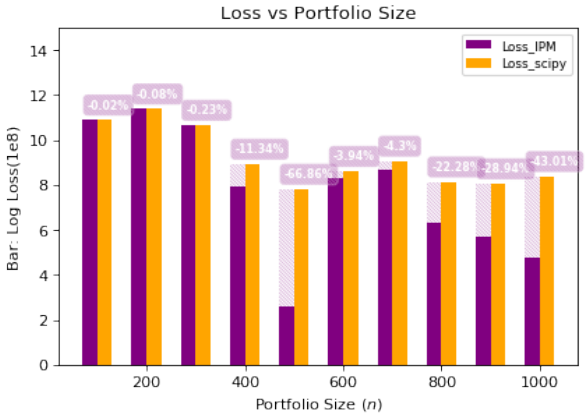
Our hypothesis is confirmed after measuring the runtime of CUDAs version. Also, the speedup becomes even more significant as the number of stocks increases.

7 Result

Below we present the running time comparison of different methods against the portfolio size n. For $n = 1,000$ stocks, our IPM method is 10 times faster compared to `SciPy` optimizer. When CUDA is enabled, the improvements increased to staggering 200 times. We expect the difference will be even larger when n increases.



Our method is not only faster, but it is also more accurate. Below we show the comparison of loss (measured in variance of the resulting portfolio) for different n. when portfolio size increase, our method(IPM) tends to generate a portfolio with lower variance as compared to `SciPy` counterpart.



8 Conclusion

In this paper, we introduce the Interior Point Method (IPM) to improve both the speed and accuracy of convex optimization routines in Python. Multiple optimization techniques have been discussed, including *Cython* and parallel programming, but the IPM with CUDA is the optimal way to improve convex optimization experience in python, at least in the portfolio optimization setting. Our result suggests over 200 times speed improvement and improved accuracy. We also believe that our method could be applied in scenarios other than portfolio optimization.

References

[1] S. Marc, *Markowitz revisited: Mean variance models in financial portfolio analysis*, March 2001, SIAM Review.

[2] J. Gondzio and A. Grothey, *Parallel interior point solver for structured quadratic programs: Application to financial planning problems*, [*Technical Report MS-03-001, School of Mathematics*], School of Mathematics, University of Edinburgh, Edinburgh EH9 3JZ, Scotland, UK, April 2003, Annals of Operations Research.

[3] L. Tao, et al., *GPU Acceleration of Interior Point Methods in Large Scale SVM Training*, 2013, 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications.