Optimization Methods for Futures Trading

Dominick Dupuy

November 27th, 2024

Abstract

This paper discusses different methods of floating-point rounding in the context of futures trading. Current implementations of this operation prove to be computationally exhaustive and fail to guarantee optimal solutions. As a result, we explore other optimization methods such as Stochastic Gradient Descent (SGD), Adam Optimization, and Pyomo-based constrained optimization, which are particularly helpful for non-convex landscapes.

The results demonstrate that Adam and Greedy methods consistently deliver the best performance across metrics, balancing accuracy and computational efficiency. Pyomo provides competitive accuracy but incurs significantly higher computational costs, making it more suitable for scenarios requiring flexible, general-purpose optimization. Conversely, the simplicity and efficiency of the Greedy method make it ideal for convex objectives such as tracking error but may limit its applicability to more complex problems involving volatility or portfolio diversification.

This study highlights the trade-offs between accuracy, computational cost, and problem complexity in optimization methods. Future research could focus on leveraging multithreaded approaches in C++ or integrating portfolio-level constraints on volatility and diversification into existing optimization frameworks.

1 Introduction

As strategies are implemented, a recurring problem in executing said strategy is whether the ideal positions are reflected in the realized positions. This is an especially pressing problem in the world of futures, where they can only be traded in whole numbers. Most strategies assume the best positions without taking this constraint into consideration. The current method of optimization is a greedy algorithm; however, its computational efficiency is far from perfect.

Optimization in this context can become very involved. An idealized portfolio has to meet several criteria before being executed. It must meet capital constraints while maintaining risk at a minimum and accurately reflecting the desired positions. For simplicity, we will consider the tracking error—the square difference between the ideal and realized positions—as the main focus of this paper. It is important to recognize that this function is convex. Intuitively, this makes sense because the squared difference magnifies deviations symmetrically on either side of the ideal value, and its second derivative is always positive, indicating a single global minimum.

Mathematically, the tracking error can be expressed as:

$$\text{TE} = \sum_{i=1}^{n} (x_{\text{realized},i} - x_{\text{ideal},i})^2,$$

where $x_{\text{realized},i}$ and $x_{\text{ideal},i}$ are the realized and ideal positions for asset i.

Taking the first and second derivatives of the tracking error with respect to $x_{\text{realized},i}$ demonstrates convexity:

$$\frac{\partial \text{TE}}{\partial x_{\text{realized},i}} = 2(x_{\text{realized},i} - x_{\text{ideal},i}),$$

$$\frac{\partial^2 \text{TE}}{\partial x_{\text{realized},i}^2} = 2.$$

Since the second derivative is a constant positive value, the function is convex. This property ensures that optimization algorithms, such as gradient descent, converge efficiently to a global minimum, where the tracking error is minimized. Additionally, the absolute value of deviations, $|x_{\text{realized},i} - x_{\text{ideal},i}|$, also exhibits a convex shape but lacks differentiability at zero, making the squared difference more suitable for gradient-based methods.

2 Optimization Methods

2.1 Greedy Algorithm (Current Method)

The current system uses a greedy algorithm, which increments or decrements the number of positions in a certain asset and tests it with the constraints and the best attempt. If the new position is found to be better than the current best attempt, then it is brought to the next iteration, where it keeps trying until no better solution is found.

2.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variation of Gradient Descent that improves computational efficiency by operating on smaller random subsets (minibatches) of the data, rather than the entire dataset, during each iteration. This stochastic approach introduces variability, which helps escape local minima and better explore the non-convex optimization landscape.

SGD iteratively updates the positions based on the gradient of the tracking error (TE) with respect to the current positions. For each minibatch of assets, the following steps are performed:

- **Gradient Calculation:** The gradient of the tracking error for a minibatch is calculated as:
$$\nabla \text{TE} = \frac{x_{\text{curr\_batch}} - x_{\text{ideal\_batch}}}{2\|\text{TE}\|},$$
where $x_{\text{curr\_batch}}$ and $x_{\text{ideal\_batch}}$ are the current and ideal positions for the assets in the minibatch, and $\|\text{TE}\|$ is the norm of the tracking error for numerical stability.

- **Penalty Term (Optional):** A penalty term is introduced to enforce capital constraints:
$$\nabla \text{Penalty} = -2\lambda \frac{(C - \text{used\_capital})p_{\text{batch}}}{C},$$
where λ is the penalty weight, C is the capital limit, and p_{batch} represents the prices of the assets in the minibatch.

- **Gradient Normalization:** The total gradient is normalized to ensure consistent step sizes:
$$\nabla_{\text{total}} = \frac{\nabla \text{TE} + \nabla \text{Penalty}}{\|\nabla \text{TE} + \nabla \text{Penalty}\| + \epsilon},$$
where ϵ is a small value for numerical stability.

- **Position Update:** The positions are updated using the learning rate α:
$$x_{\text{curr\_batch}}^{(t+1)} = x_{\text{curr\_batch}}^{(t)} - \alpha \nabla_{\text{total}}.$$

The iterative process continues for a fixed number of iterations or until the tracking error converges below a predefined threshold.

2.3 Pyomo Optimization

Pyomo is a Python-based optimization modeling language used for solving constrained optimization problems. In this approach, we use Pyomo to minimize the tracking error between current and ideal asset positions while adhering to a capital limit.

The key components of this optimization strategy are as follows:

- **Decision Variables:**
Let curr_pos_i represent the decision variable for the position in asset i, where $i \in \{1, 2, \ldots, n\}$. These variables are initialized based on the current portfolio.

- **Objective Function:**
  The objective is to minimize the tracking error, defined as:

$$\text{Tracking Error} = \sum_{i=1}^{n} (\text{curr\_pos}_i - \text{ideal\_pos}_i)^2$$

- **Constraints:**

  1. **Capital Constraint:** Ensure the total cost of the portfolio does not exceed the available capital:

$$\sum_{i=1}^{n} (\text{curr\_pos}_i \cdot \text{price}_i) \leq \text{Capital\_Limit}$$

     where price_i is the price of asset i.

  2. **Non-Negativity Constraint (Optional):** Enforce non-negative positions if short-selling is disallowed:

$$\text{curr\_pos}_i \geq 0 \quad \forall i \in \{1, 2, \ldots, n\}$$

- **Optimization Process:** The optimization is performed using the `ipopt` solver, which employs an interior-point algorithm for nonlinear programming. The Pyomo model is defined as follows:

  1. Initialize the Pyomo model and define the decision variables.
  2. Specify the objective function and constraints.
  3. Solve the optimization problem using a solver:

$$\min_{\text{curr\_pos}} \text{Tracking Error}$$

     subject to the defined constraints.

The optimized positions are extracted and rounded to the nearest integer values to reflect the discrete nature of financial trades. This approach ensures that the portfolio aligns closely with the ideal positions while adhering to practical constraints.

2.4 Adam Optimization

Adam Optimization combines the benefits of momentum-based optimization and RMSProp, which uses an adaptive learning rate. Adam maintains two moving averages: the gradient (m_t) and the squared gradient (v_t). These moving averages are corrected for their bias to stabilize the optimization process in its early stages.

Formally, Adam Optimization can be defined as follows:

- **Gradient Update:**

$$g_t = \text{TE}'(w_t)$$

  where g_t is the gradient of the tracking error with respect to w_t (current weights).

- **First Moment Estimate (Mean):**

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

where m_t is the exponentially weighted average of past gradients, and β_1 is the decay rate for the first moment.

- **Second Moment Estimate (Variance):**

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where v_t is the exponentially weighted average of squared gradients, and β_2 is the decay rate for the second moment.

- **Bias Correction:**

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- **Weight Update:**

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where α is the learning rate, and ϵ is a small constant added for numerical stability.

Adam Optimization is particularly effective for non-convex objective functions and has been shown to converge faster and more stably than other gradient-based methods.

3 Methodology

The findings of this paper are intended to be extrapolated to futures trading; however, this implementation only supports equities for simplicity. Stocks do not have to be traded in whole numbers, but we will add this constraint artificially to make use of the optimization methods described. Equities also tend to be much cheaper than even the cheapest futures, so we set the simulated capital to be a mere $1,000 for more significant results since, at this scale, determining whether a position should be scaled up or down is more meaningful.

The strategy used to test the tracking error of these optimization methods is irrelevant, but for curious readers, the strategy is very simple: it finds the top 10 best-performing stocks in a day by percent change and invests in those stocks for the next day.

To quantify the performance of different optimization methods, we present the following metrics for measuring tracking error:

- **Mean Absolute Tracking Error (MATE):**

$$\text{MATE} = \frac{1}{N} \sum_{i=1}^{N} |x_{\text{realized},i} - x_{\text{ideal},i}|$$

- **Root Mean Square Tracking Error (RMSTE):**

$$\text{RMSTE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{\text{realized},i} - x_{\text{ideal},i})^2}$$

- **Maximum Tracking Error (MaxTE):**

$$\text{MaxTE} = \max(|x_{\text{realized}} - x_{\text{ideal}}|)$$

- **Tracking Error Volatility (TEV):**

$$\text{TEV} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{TE}_i - \text{TEM})^2}$$

where TEM is the mean tracking error.

- **Portfolio Level Tracking Error (PLTE):**

$$\text{PLTE} = \frac{1}{N}\sum_{i=1}^{N}(w_{\text{realized},i} - w_{\text{ideal},i})^2$$

where w denotes the weight for an asset.

4 Results and Discussion

The performance of the optimization methods was evaluated across five metrics: Mean Absolute Tracking Error (MATE), Root Mean Square Tracking Error (RMSTE), Maximum Tracking Error (MaxTE), Portfolio Level Tracking Error (PLTE), and computational time. The results are summarized below.

4.1 Mean Absolute Tracking Error (MATE)

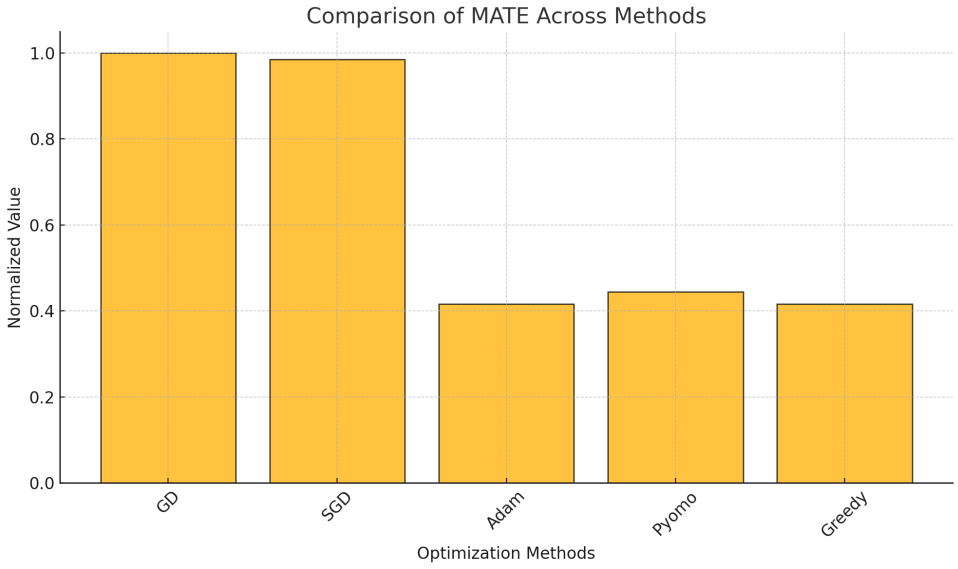

Figure 1: Comparison of Mean Absolute Tracking Error (MATE) across optimization methods. Lower values indicate better alignment between realized and ideal positions.

Figure 1 shows that Adam and Greedy methods achieved the lowest MATE values (0.8006 and 0.8, respectively), indicating precise alignment with ideal positions. Pyomo performed comparably at 0.8547, while GD and SGD showed higher errors, reflecting less effective optimization.

4.2 Root Mean Square Tracking Error (RMSTE)

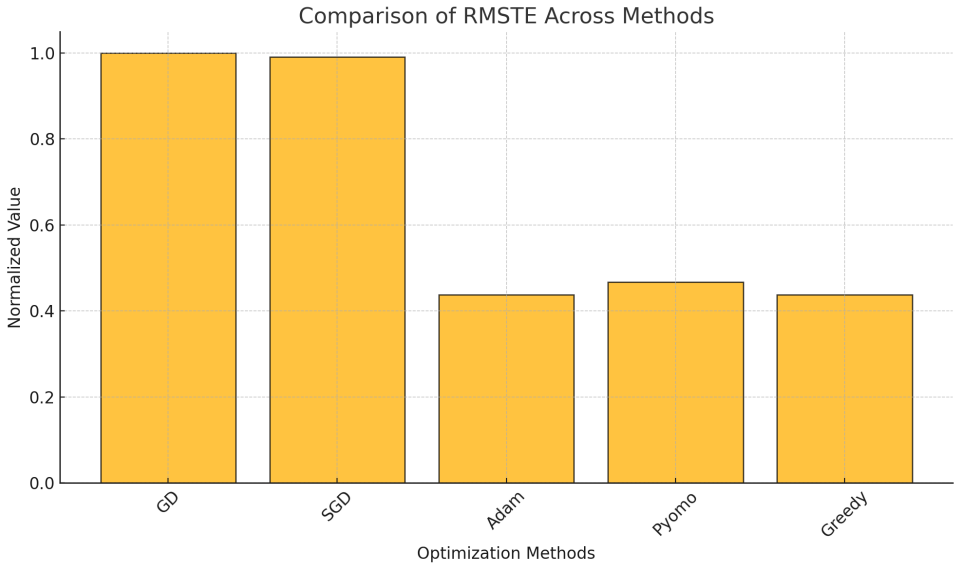

Figure 2: Comparison of Root Mean Square Tracking Error (RMSTE) across optimization methods. Lower values indicate better precision in minimizing tracking deviations.

Figure 2 demonstrates that Adam (1.2431) and Greedy (1.2421) outperform other methods, with Pyomo (1.327) following closely. GD and SGD exhibited significantly higher RMSTE values, suggesting less precise tracking.

4.3 Maximum Tracking Error (MaxTE)

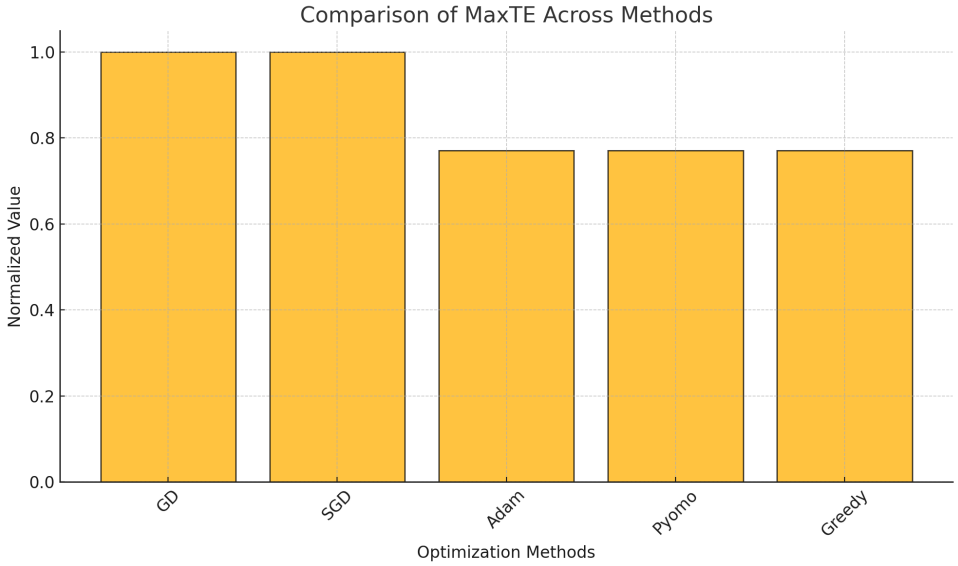

Figure 3: Comparison of Maximum Tracking Error (MaxTE) across optimization methods. Lower values reflect better consistency in achieving desired positions for the most significant deviations.

As shown in Figure 3, Adam, Pyomo, and Greedy achieved significantly lower MaxTE (30.3391), indicating better control over extreme deviations, while GD and SGD showed poorer performance (39.3391).

4.4 Portfolio Level Tracking Error (PLTE)

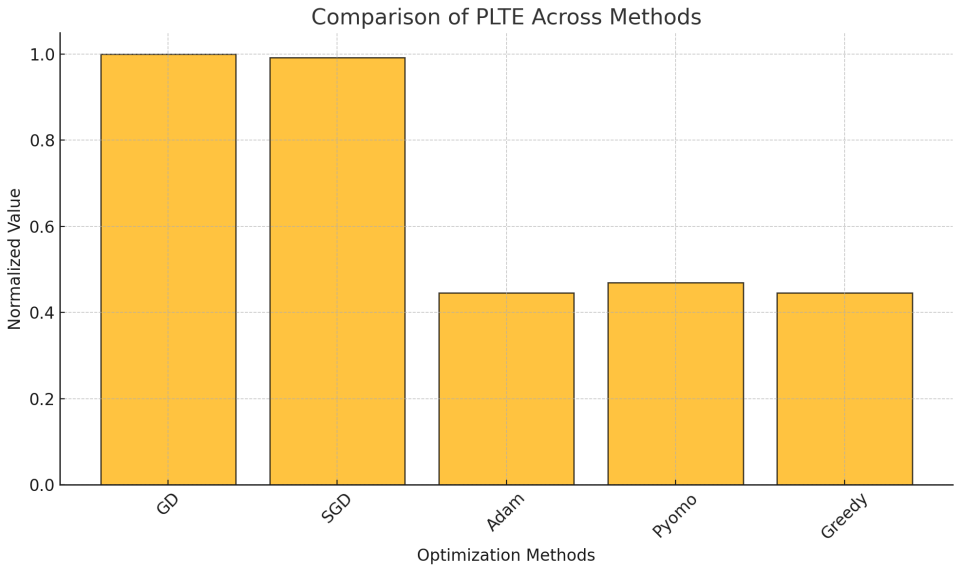

Figure 4: Comparison of Portfolio Level Tracking Error (PLTE) across optimization methods. Lower values indicate better overall portfolio alignment with ideal positions.

Figure 4 illustrates that Adam (4754.3908) and Greedy (4750.4992) lead in portfolio-level accuracy, followed closely by Pyomo (5013.3455). GD and SGD showed higher tracking errors, reflecting less accurate portfolio alignment.

4.5 Computational Time

Figure 5 compares the computational times. Greedy (0.004s) and Adam (0.006s) were the fastest, while Pyomo required 0.301s, reflecting its general-purpose nature. GD and SGD fell in between, with times of 0.056s and 0.105s, respectively.

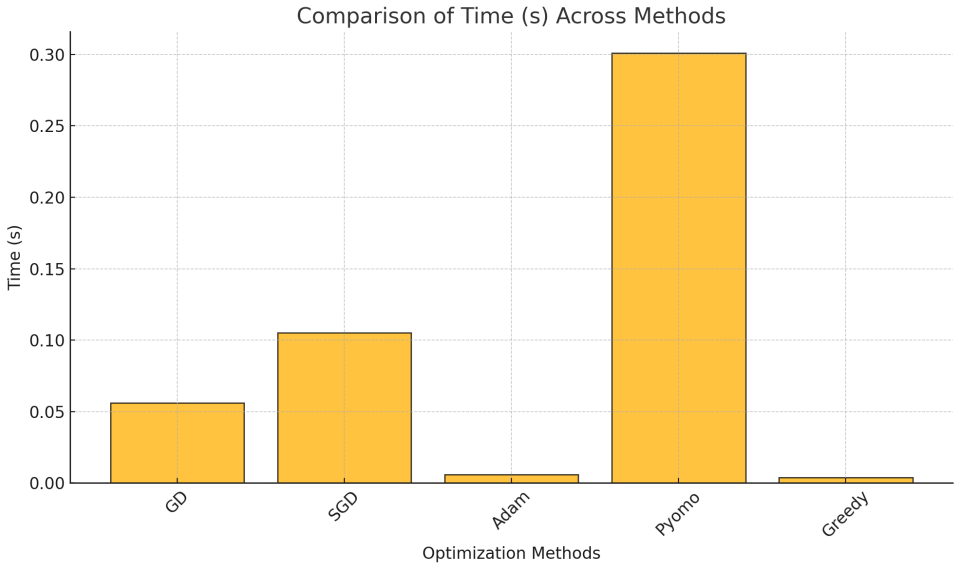

Figure 5: Comparison of computational time (in seconds) across optimization methods. Lower values indicate higher computational efficiency.

4.6 Summary of Results

In summary, Adam and Greedy optimizations consistently demonstrated the best performance across all metrics, balancing accuracy and computational efficiency. Pyomo provided competitive accuracy but saw significantly higher computational costs. GD and SGD were less effective overall, showing higher errors across all metrics.

The current method (Greedy) stands out for its simplicity and efficiency, achieving fast and accurate results due to the convex nature of the objective function—the tracking error. However, this simplicity may become a limitation if more complex objective functions are introduced. For instance, if the optimization needed to account for how changes in the number of positions in a certain asset affect the volatility or diversification of a portfolio, the Greedy approach might not perform as well. In such scenarios, more advanced methods like Adam or Pyomo would likely be better equipped to handle the increased complexity.

Overall, while the Greedy method remains highly effective for simple convex problems, its applicability to more sophisticated optimization tasks may be limited.