# CS 303 - Coding Assignment 1 - 23110049

Github Link => https://github.com/Aryan-IIT/coding1_assignment_cs303

## Problem 1

**Expected Deliverables**

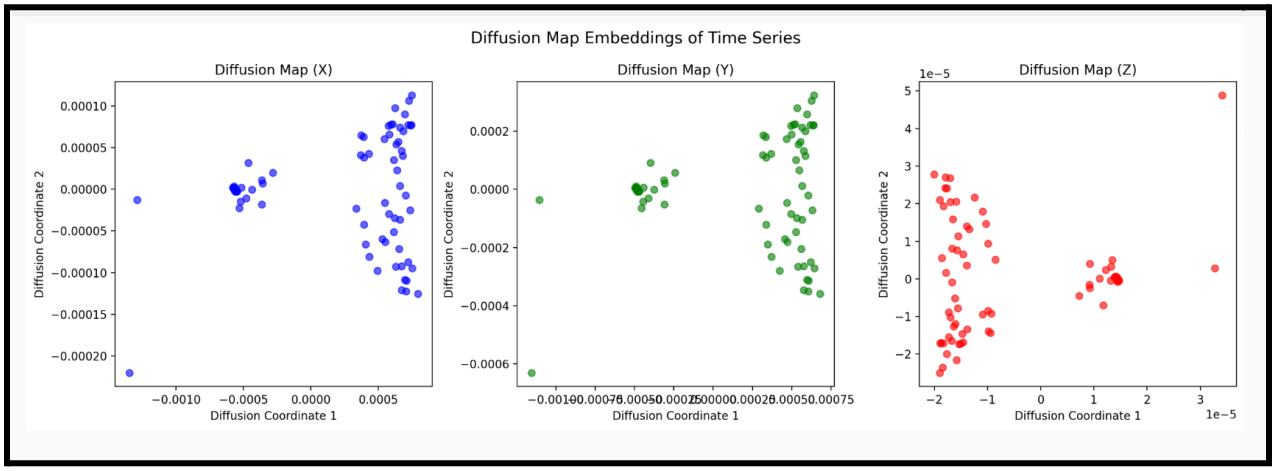
• Python Notebook implementing Diffusion Maps for clustering. => Link
• Include plots showing low-dimensional embeddings of time-series segments in the above notebook
• Short Report explaining the choice of distance metric and why Diffusion Maps outperform PCA/t-SNE.

# Low Dimensional Embeddings of Time Series

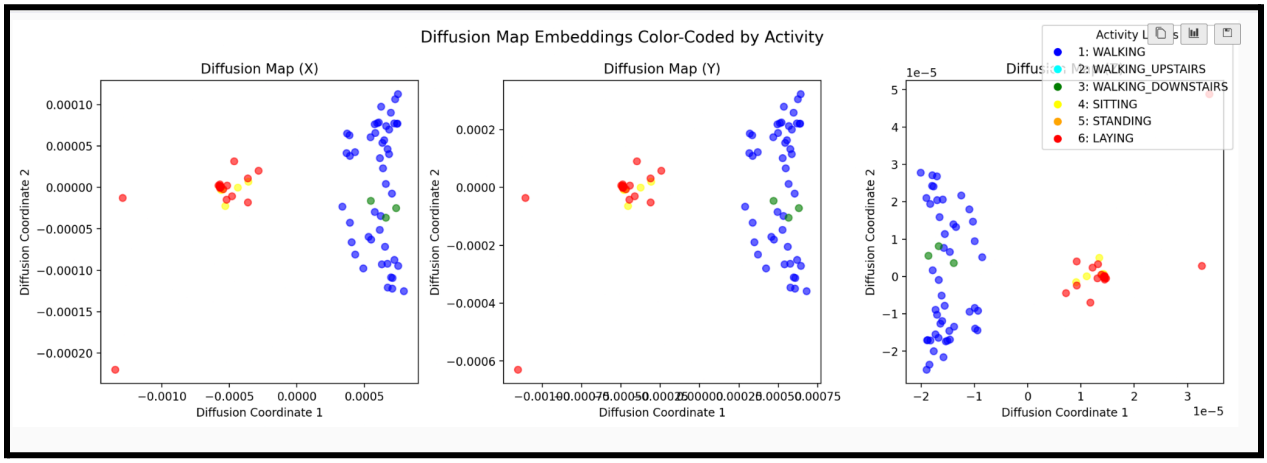
In this case, I have taken 100 x 3 samples for each axis.
Reasoning: Faster Compute and Clearer Visualization.

Additionally, the design choice I have followed is to perform all processes individually on the axes. We could have done them together as well.

Embedding space

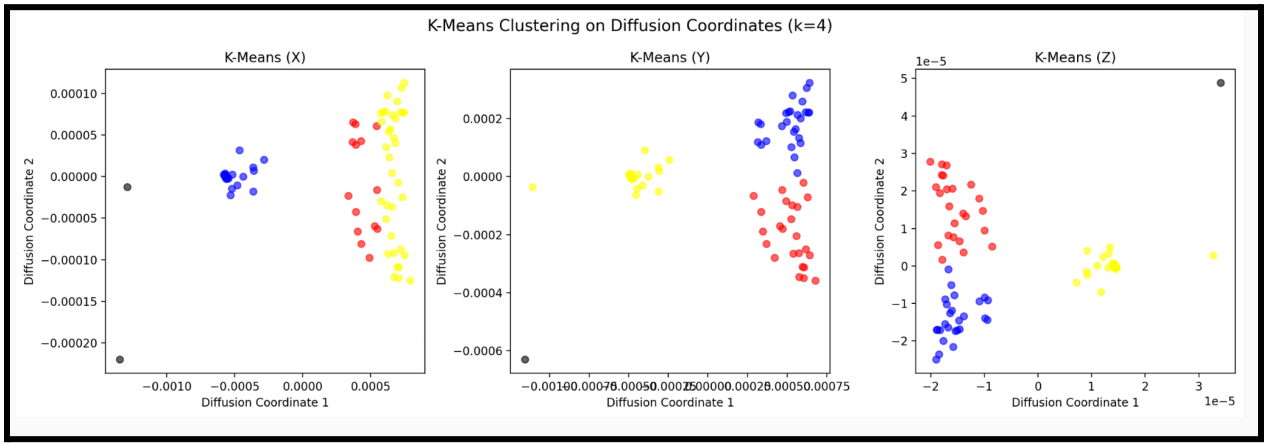



Ground Truth




Here, we can observe that there is a clear separation between walking+walking_downstairs vs sitting and laying.

K-means based clustering.




K-means does a reasonable job, as well.

Ari score

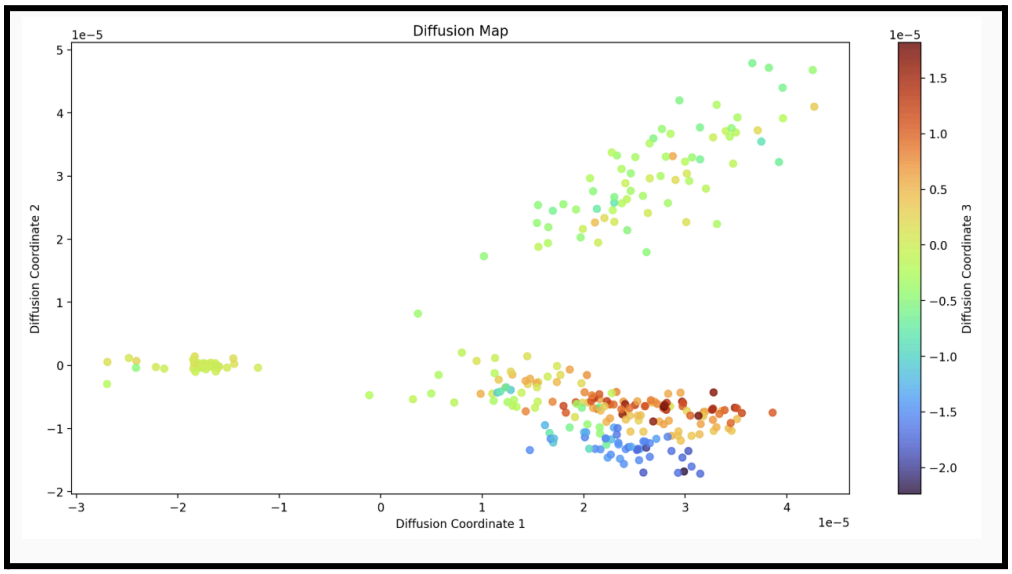
| | Method | X | Y | Z |
|---|---|---|---|---|
| 0 | Raw Features | 0.009892 | -0.035978 | 0.054406 |
| 1 | PCA | 0.096773 | 0.069976 | 0.093558 |
| 2 | t-SNE | 0.015082 | 0.015082 | 0.016618 |
| 3 | Diffusion Map | 0.494786 | 0.452742 | 0.452742 |

Silhouette Score

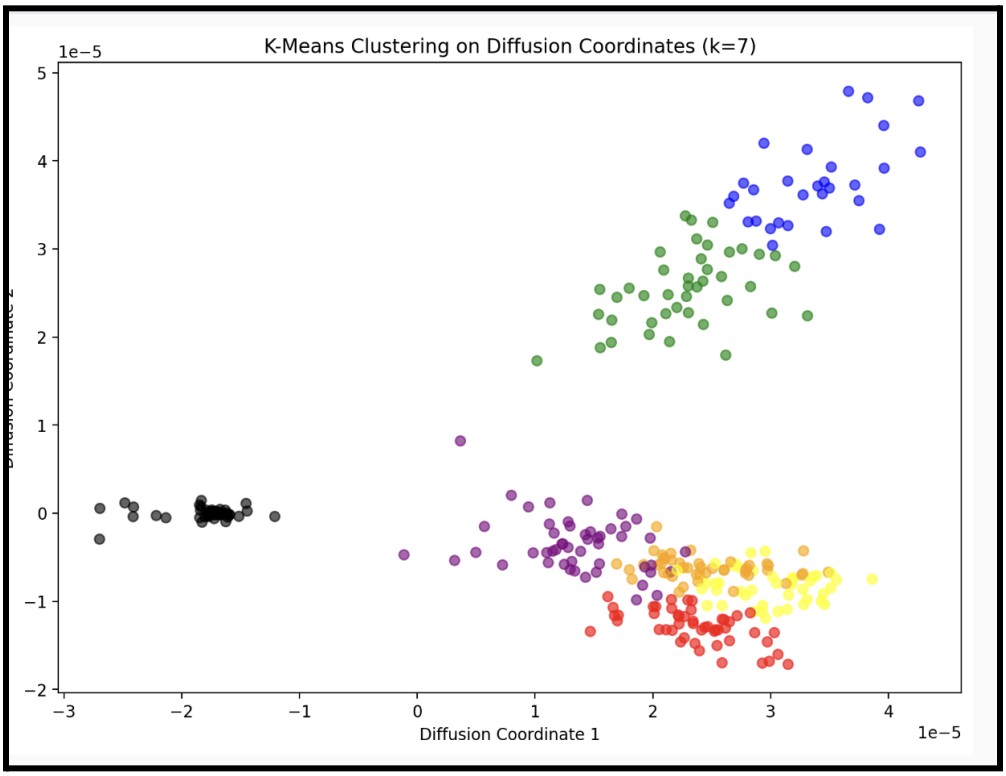
| | Method | X | Y | Z |
|---|---|---|---|---|
| 0 | Raw Features | 0.139534 | 0.238763 | 0.287741 |
| 1 | PCA | 0.488685 | 0.434136 | 0.508392 |
| 2 | t-SNE | 0.229370 | 0.229370 | 0.228949 |
| 3 | Diffusion Map | 0.703534 | 0.723222 | 0.778572 |

Now, let us try the same with n = 500 x 3. Here, let us perform dtw and all other processes for all axes together. Notebook

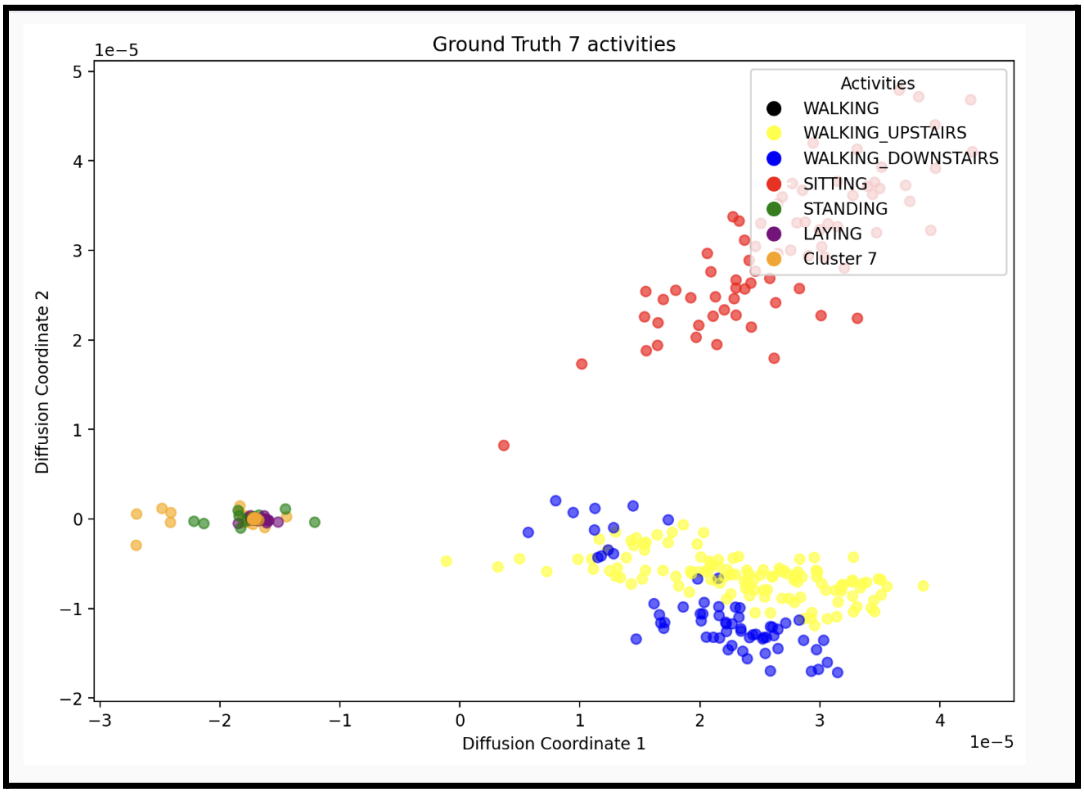
Embedding space




Kmeans



Ground Truth




ARI Score = 0.3926
Silhouette Score = 0.6223

Here, We can see that even with diffusion maps, if we use all axis at once, the silhouette and ari score drops. Though visibly we can see differences, k-means proves to be an ineffective clustering method.

One way to improve this is to maybe use 4 or 5 as the embedding dimension or use other clustering techniques.

## Why do diffusion maps work better than PCA and others? Why did we use DTW as a distance metric?

For time-series clustering, DTW as a metric is generally better. Because

DTW is effective as it allows sequences to be compared even if they vary in speed.It finds an optimal alignment by minimizing the distance between corresponding points in two time series. This flexibility makes it suitable for (HAR) human activity recognition, where variations exist in motion patterns.

Whereas, Euclidean is a simpler metric that measures pointwise differences.It fails to capture temporal misalignment, and hence making it less suitable for activity recognition.

### DTW (Dynamic Time Wrapping) - Pairwise For x, y and z direction respectively

Dynamic Time Warping (DTW) is computed using the following recurrence relation:

$$D(i,j) = d(x_i, y_j) + \min \begin{cases} D(i-1,j) & \text{(insertion)} \\ D(i,j-1) & \text{(deletion)} \\ D(i-1,j-1) & \text{(match)} \end{cases}$$

where $d(x_i, y_j)$ is the Euclidean distance between points.

The total warping cost is:

$$DTW(X,Y) = D(N,M)$$

where $X = (x_1, x_2, ..., x_N)$ and $Y = (y_1, y_2, ..., y_M)$.

The main idea is to compute the distance from the matching of similar elements between time series.

Example, imagine two people saying the same sentence, one speaking quickly and one speaking more slowly. If I were to graph the sound waves of each sentence, the shapes would look similar in terms of rises and falls, but they would not line up perfectly all the time axis due to the difference in speak. If I tried to compare these sound waves using a straightforward, point-to-point approach (like Euclidean distance), the comparison would be unfair.

It's like trying to synchronize two singers when one is slightly ahead or behind the other; they're singing the same notes, but not at the same time, so a direct comparison at any given moment wouldn't make much sense. Hence, DTW is better.

Diffusion maps outperform PCA and t-SNE for dimensionality reduction in time-series data by **capturing intrinsic geometric structures** while **preserving both local and global relationships.** This can be observed/understood by the formula written above.

The reason why PCA is not good enough, is becuase it assumes linearity. And t-SNE, also struggles with global structure and is computationally expensive, whereas for diffusion maps they leverage the diffusion process, making them more robust to noise and variations. This advantage makes them particularly effective for clustering tasks in Human Activity Recognition (HAR) datasets like this assignment's dataset.

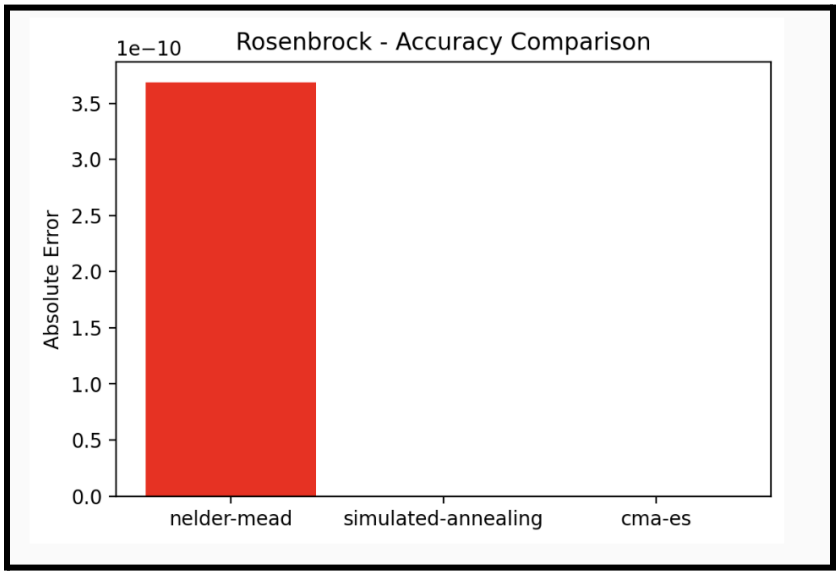
# Problem 2

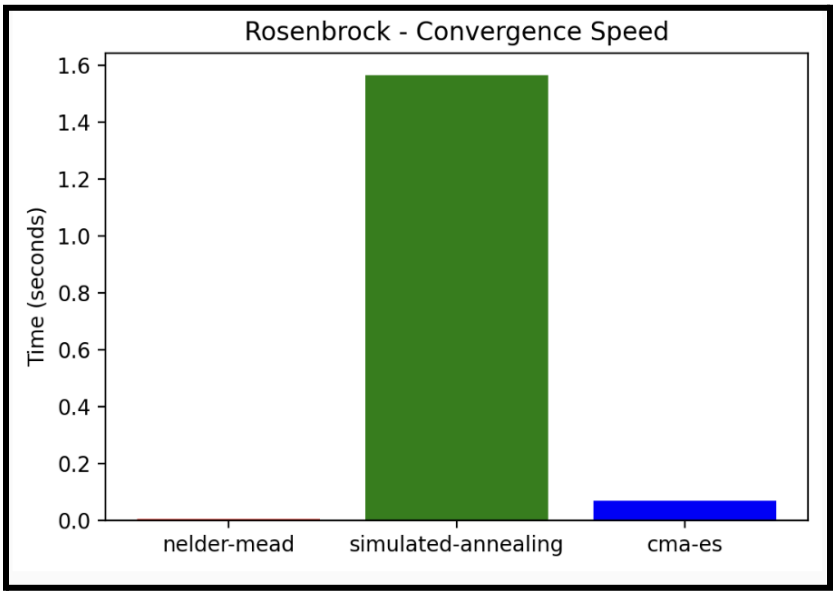
Expected Deliverables

- Python Notebook implementing and comparing all three optimization techniques. => Notebook
- Include Plots and analysis of function convergence in the above notebook
- Short Report discussing the trade-offs of each method.

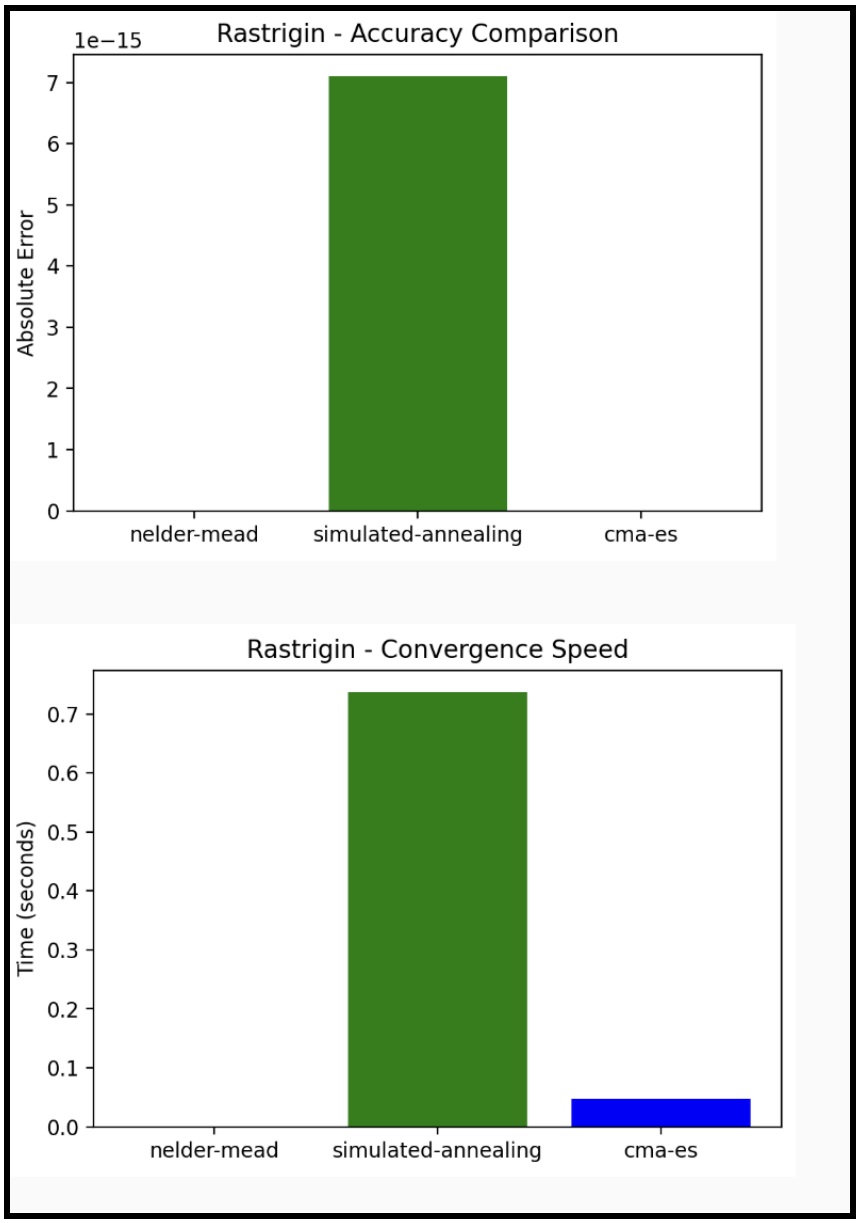
The plots are based on our setup and implementation, these may vary by a little from implementation to other implementations.

Plots

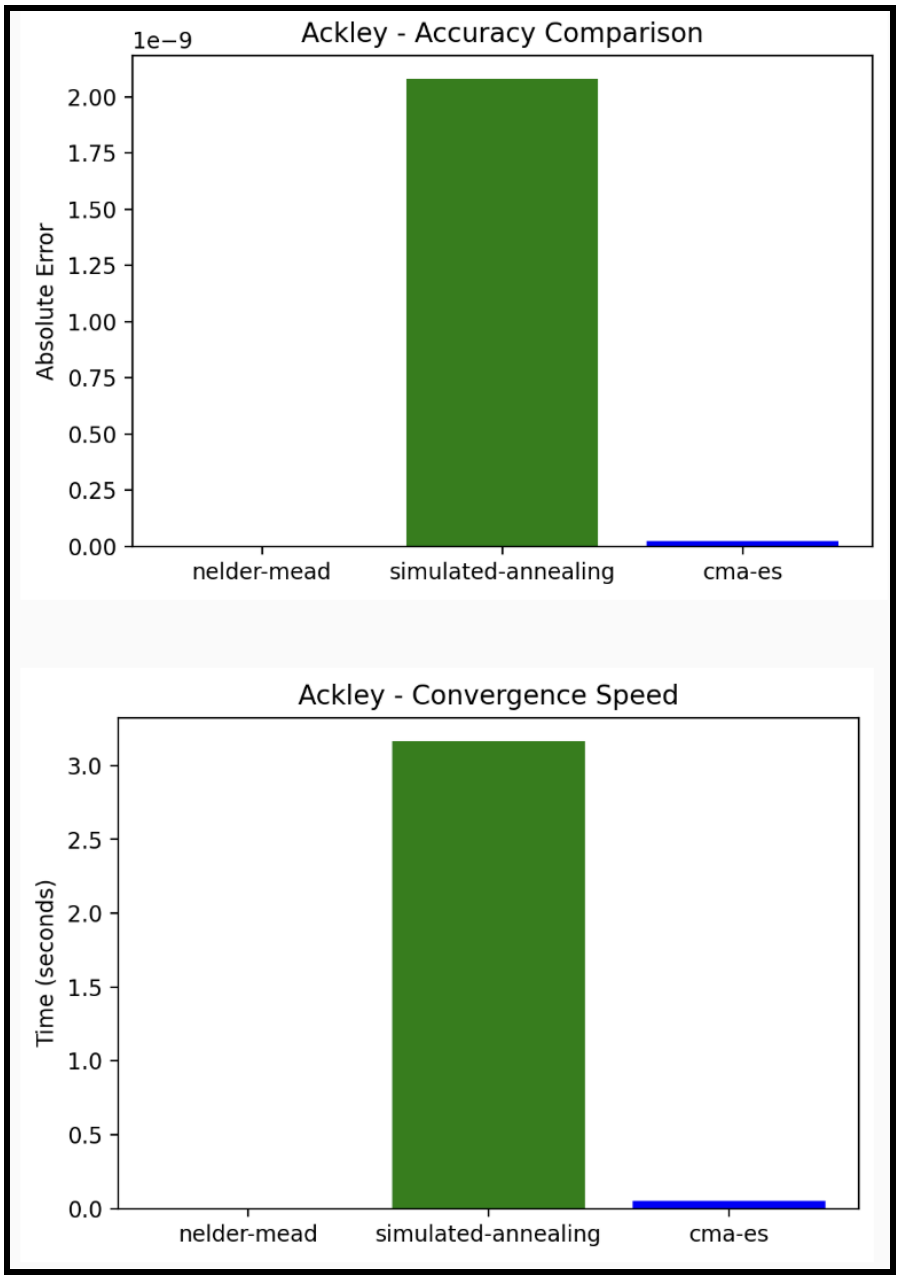


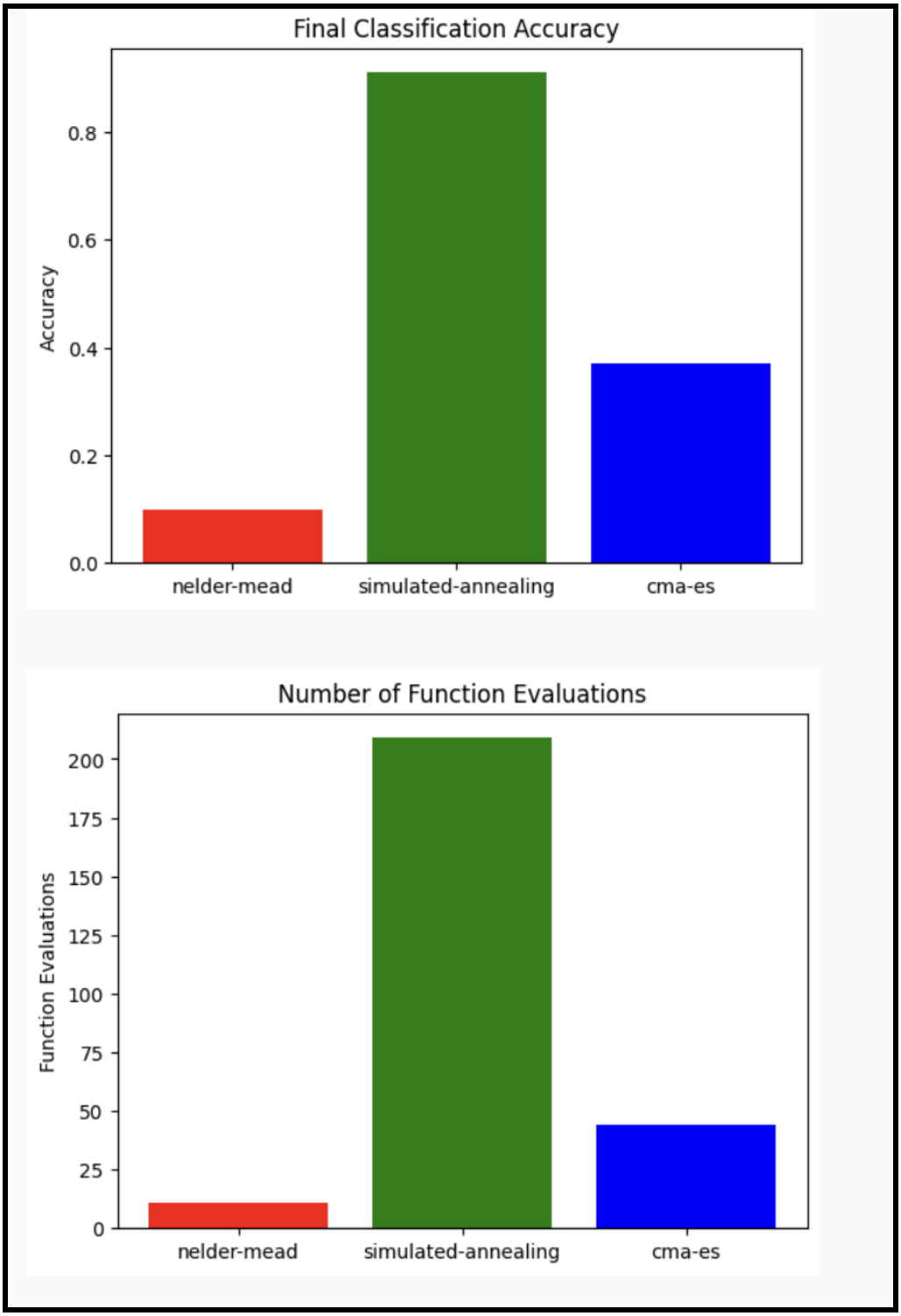

Here, we can see Rosenbrock converges the fastest with Nelder-mead and also gives the highest accuracy with it or best function value achieved.

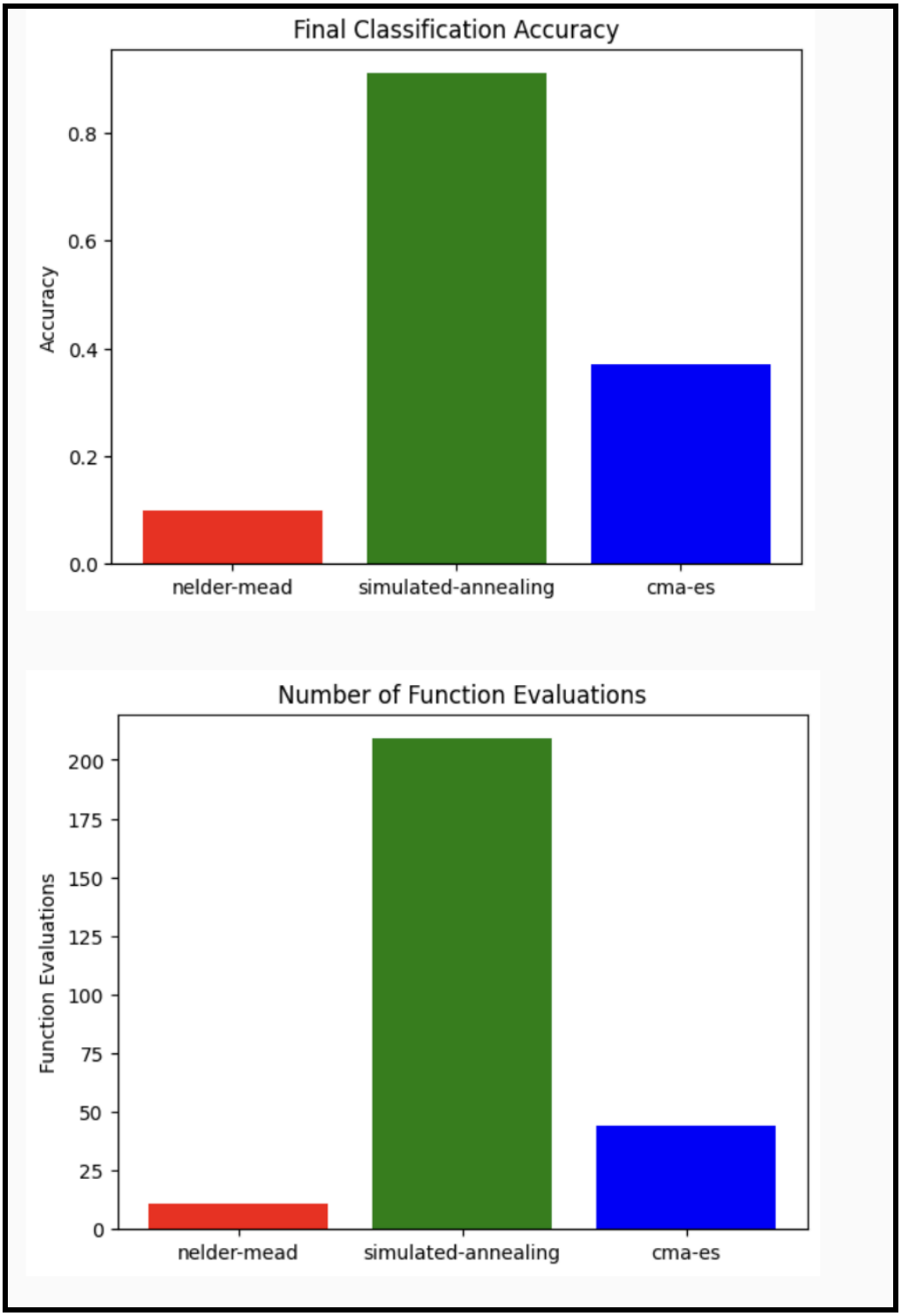



For Rastrigin, simulated annealing takes the most amount of time yet it gives the best accuracy.

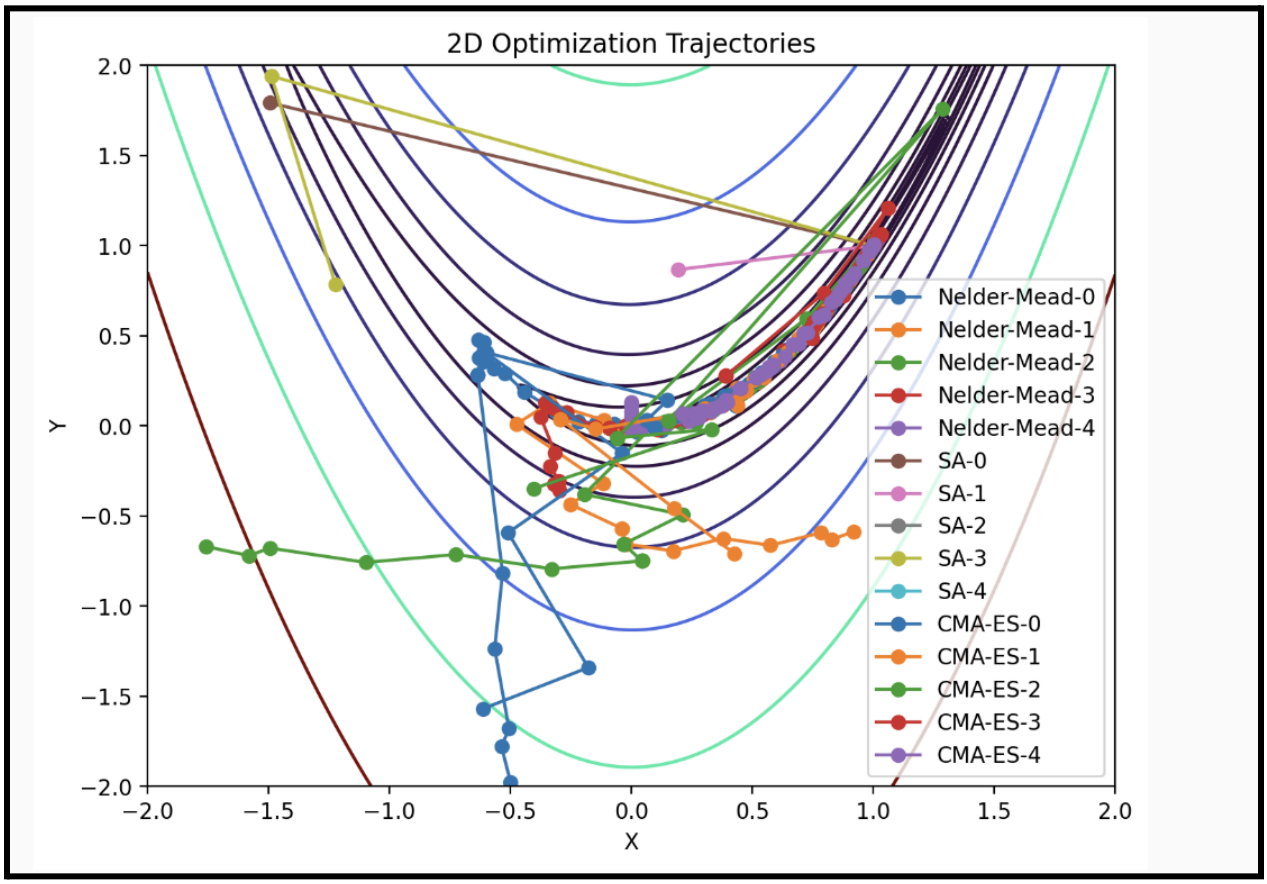



For Ackely, simulated annealing takes the most amount of time yet it gives the best accuracy.

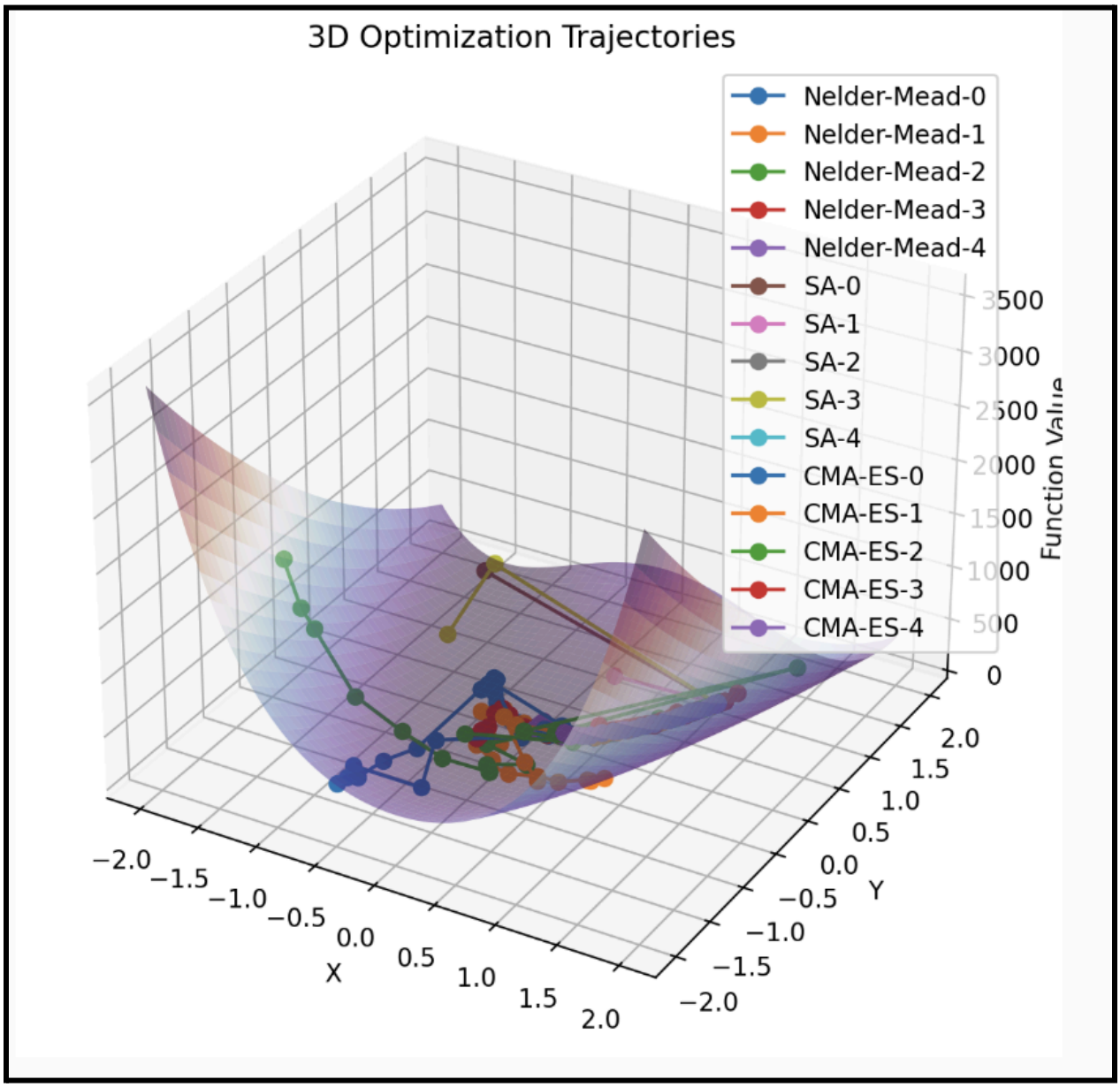

Final Classification Accuracy
Accuracy
0.0
0.2
0.4
0.6
0.8
nelder-mead
simulated-annealing
cma-es
Number of Function Evaluations
Function Evaluations
0
25
50
75
100
125
150
175
200
nelder-mead
simulated-annealing
cma-es


Final Classification Accuracy
Accuracy
0.0
0.2
0.4
0.6
0.8
nelder-mead
simulated-annealing
cma-es
Number of Function Evaluations
Function Evaluations
0
25
50
75
100
125
150
175
200
nelder-mead
simulated-annealing
cma-es

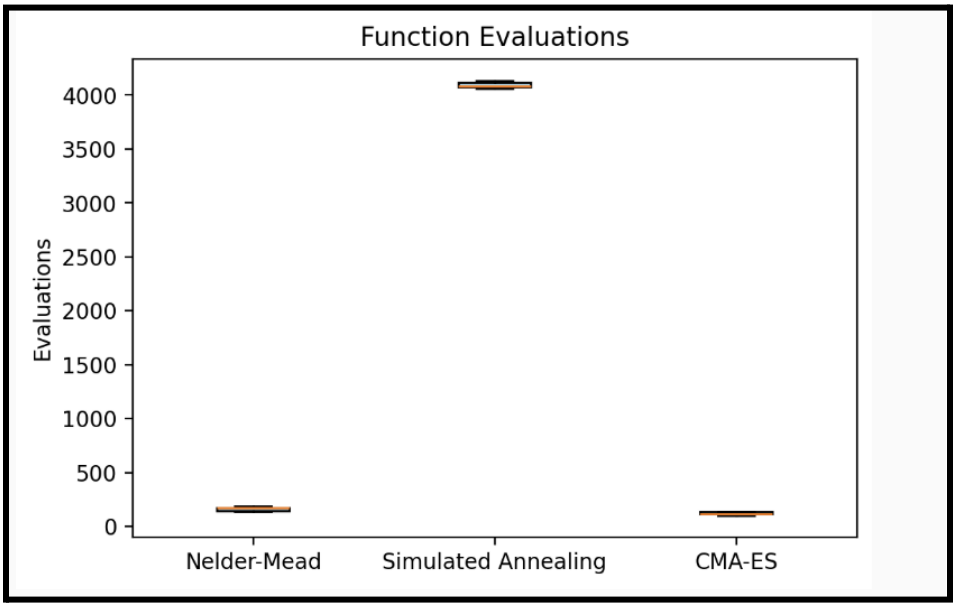
Now, I performed the three optimization strategies for the **Rosenbrock function.**
We used 5 sets of  initial points generated uniformly at random between -2 and 2.



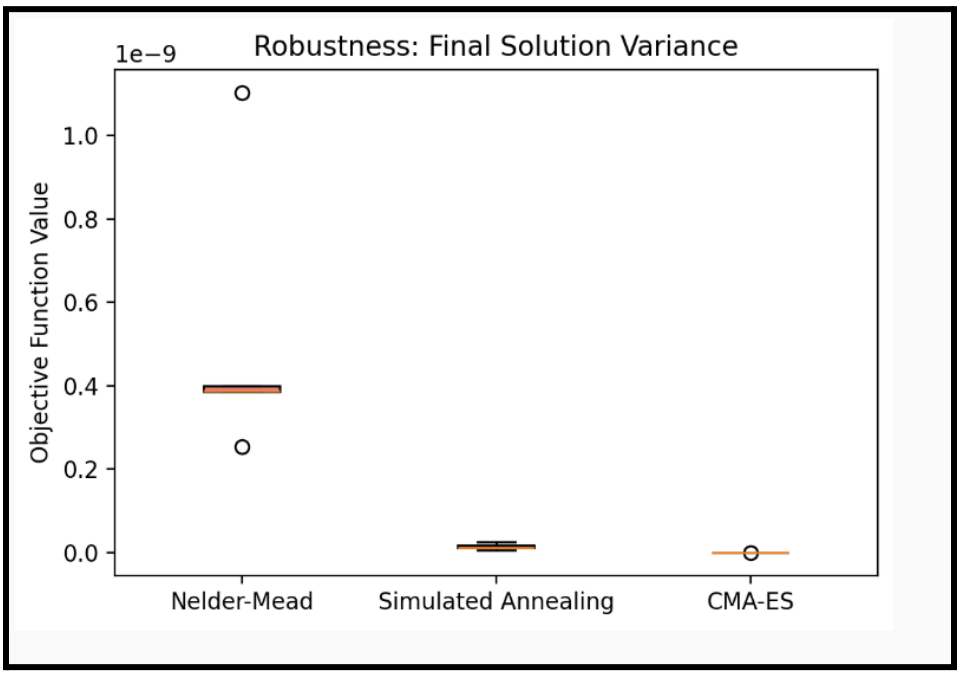
The same plot, better visualized in 3D=>



Few more evaluations

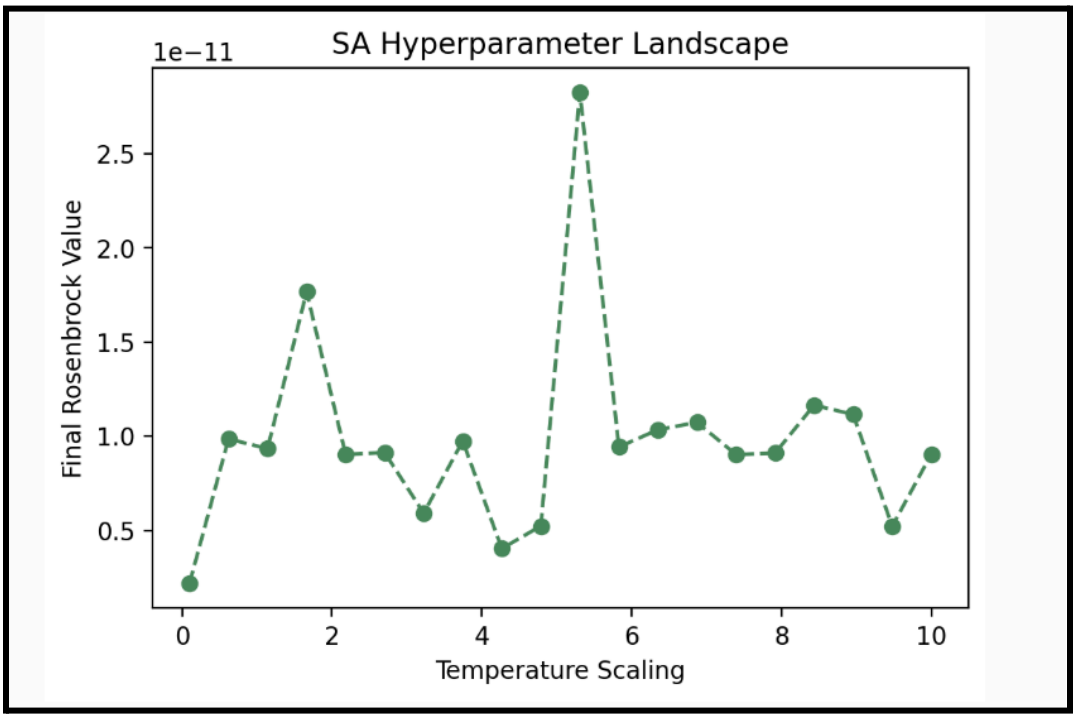



Nelder-mead and CMA-ES take less than 500 evaluations and are generally more efficient than Simulated annealing.




This plot shows the variance in the final objective function values achieved by each algorithm. Lower variance and lower values indicated more robustness in finding global minima.

For Simulated annealing we can see that the function’s (Rosenbrock's) value changes with temperature.



Report and Tradeoffs

1. **Nelder-Mead (Simplex Method)**

Nelder-Mead, also known as the Simplex Method as taught as a part of a worksheet also, is a simple optimization technique that does not require gradient information. It works well for small problems but becomes inefficient in higher dimensions. One major drawback is its tendency to get stuck in local minima, making it unreliable for complex landscapes.

2. **Simulated Annealing**

Simulated Annealing is a more flexible approach that can escape local minima by occasionally accepting worse solutions. This makes it effective for highly non-convex problems. However, it requires careful tuning of the cooling schedule and can be computationally expensive due to its stochastic nature.

3. **CMA-ES (Covariance Matrix Adaptation Evolution Strategy)**

CMA-ES (Covariance Matrix Adaptation Evolution Strategy) is a good method that adaptively improves its search strategy over time. It performs well in high-dimensional and multimodal optimization tasks but comes at a high computational cost, requiring many function evaluations.

Hyperparameter tuning

When I applied to tuning an SVM on the MNIST dataset, CMA-ES consistently found the best hyperparameters but required the most function evaluations. Simulated Annealing provided a good balance between exploration and efficiency. Nelder-Mead, while fast, often got trapped in suboptimal solutions, making it less effective for complex problems. This aligns with the tradeoffs of these strategies.

Conclusion

Nelder-Mead is suitable for simple, low-dimensional problems but struggles with complexity. Simulated Annealing balances exploration and efficiency, making it useful for difficult optimization landscapes. CMA-ES excels in handling high-dimensional and complex tasks but is computationally expensive.

Each method has trade-offs, and the best choice depends on problem complexity and available computational resources, generally speaking.

Summary sheet (Just for reference)

# Strategies for optimizations

## Nelder-Mead

A derivative-free optimization algorithm using a simplex of $n+1$ points for $n$-dimensional problems. Operations:

1. **Reflection**: $x_r = x_c + \alpha(x_c - x_w)$
2. **Expansion**: $x_e = x_c + \gamma(x_r - x_c)$
3. **Contraction**: $x_c = x_c + \rho(x_w - x_c)$
4. **Shrinkage**: Shrinks simplex towards the best point.

## Simulated Annealing

A probabilistic algorithm inspired by annealing in metallurgy. Key steps:

1. **Acceptance Probability**: $P(\Delta E) = \exp\left(-\frac{\Delta E}{T}\right)$
2. **Temperature Schedule**: $T_k = T_0 \cdot \alpha^k$, where $\alpha \in (0, 1)$.

## CMA-ES

A stochastic optimization algorithm adapting the covariance matrix of a multivariate normal distribution:

1. **Sampling**: $x_i \sim \mathcal{N}(m, \sigma^2 C)$
2. **Update Mean**: $m = \sum_{i=1}^{\lambda} w_i x_i$
3. **Update Covariance**:

$$C = (1 - c_1 - c_\mu)C + c_1 p_c p_c^\top + c_\mu \sum_{i=1}^{\lambda} w_i (x_i - m)(x_i - m)^\top$$