CMO Assignment 1

SWAPNIL MITESHKUMAR JOSHI SWAPNILJOSHI@IISC.AC.IN
SR number : 25846

Question 1

Consider the quadratic function

$$f(x) = \frac{1}{2}x^T Qx + b^T x$$

where Q is a symmetric positive definite matrix and b is a vector in $\mathbb{R}^2$. Fix $b = (1, 1)^T$. Query the oracle with your five digit SR number to obtain five different Q matrices.

1.0 Methodology

To minimize the quadratic function, we applied **gradient descent with exact line search**. At each iteration k, the update rule is

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k),$$

where the gradient is

$$\nabla f(x) = Qx + b.$$

The exact line search step size has the closed-form expression

$$\alpha_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_k)^T Q \nabla f(x_k)},$$

which guarantees optimal progress along the descent direction.
In parallel, we also computed the analytical minimizer by solving the stationarity condition

$$Qx^\star + b = 0 \quad \Rightarrow \quad x^\star = -Q^{-1}b.$$

Finally, to study convergence, we plotted the error norm

$$\|x^{(k)} - x^\star\|$$

on a logarithmic scale for each oracle-provided Q matrix.

1.1 Minimizing f(x) using Exact Line Search

Using the exact line search method, we were able to obtain the following results:

```
Case 1
x* from exact line search: [-0.54234548 -0.53568389]
f(x*) from exact line search: -0.5390146833611191

Case 2
x* from exact line search: [-1.10374588 -0.14205034]
f(x*) from exact line search: -0.6228981124440631


Case 3
x* from exact line search: [-0.57790436 -0.3919162 ]
f(x*) from exact line search: -0.48491028097904076

Case 4
x* from exact line search: [-1.17275546 -2.87247559]
f(x*) from exact line search: -2.0226155422252656

Case 5
x* from exact line search: [ 0.39798347 -1.40464231]
f(x*) from exact line search: -0.503329424062969
```

1.2 Analytically Solving for x* and Plotting Convergence

The analytical solution for the minimizer x^* of the quadratic function is found by setting the gradient $\nabla f(x) = Qx + b$ to zero, which yields $x^* = -Q^{-1}b$.

Analytical x^* for each Case

```
Case 1: [-0.54234548 -0.53568389]

Case 2: [-1.10374589 -0.14205034]

Case 3: [-0.57790436 -0.3919162 ]

Case 4: [-1.17275547 -2.87247561]

Case 5: [ 0.39798348 -1.40464233]
```

The convergence of the gradient descent algorithm to this analytical solution is shown in the plot below, where the error $||x^{(k)} - x^*||$ is plotted against the number of iterations on a log scale.

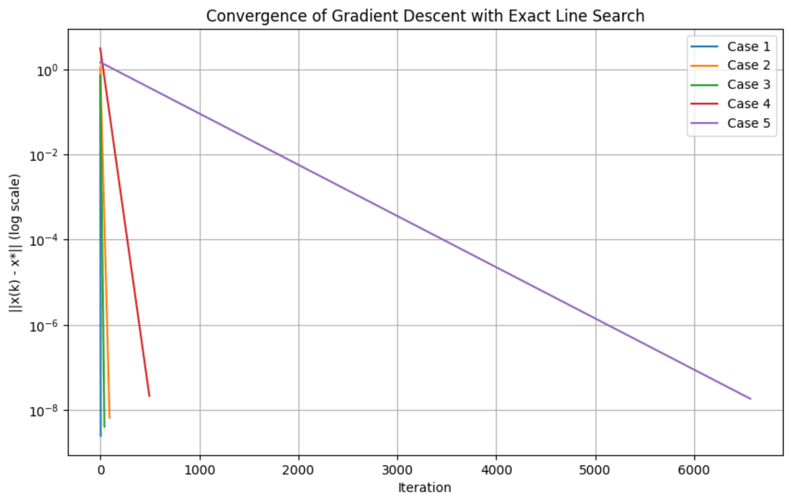


Figure 1: Convergence of Gradient Descent with Exact Line Search.

1.3 Explanation of convergence behavior

The plots show linear convergence of gradient descent with exact line search: the error $\|x^{(k)} - x^\star\|_2$ decreases by a fixed factor at each iteration, which appears as an (approximately) straight line on the semilogarithmic plot.
For a quadratic $f(x) = \frac{1}{2}x^TQx + b^Tx$ with $Q \succ 0$, the convergence factor is governed by the condition number $\kappa(Q) = \lambda_{\max}(Q)/\lambda_{\min}(Q)$. The condition number is the ratio of the largest to the smallest eigenvalue of the matrix.
In particular one can bound the contraction factor by

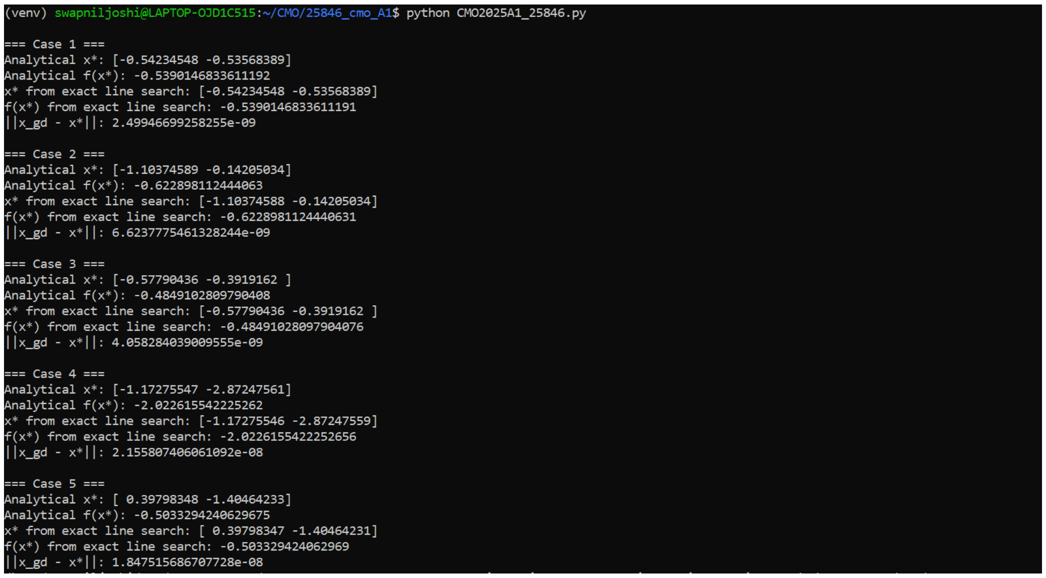
$$\rho = \frac{\kappa(Q) - 1}{\kappa(Q) + 1} \in (0, 1),$$

so that the error (in the Q-norm) satisfies $\|e_{k+1}\|_Q \leq \rho\,\|e_k\|_Q$. Thus a small κ (well-conditioned Q) yields a small ρ and fast convergence (steep line), while a large κ yields ρ close to 1 and slow convergence (flat line).
Two additional remarks explain the shape of the curves:

- With exact line search on quadratics successive gradients are orthogonal, $g_{k+1}^T g_k = 0$. This orthogonality causes the iterates to alternate directions and often produces a zig-zag trajectory, especially when eigenvalues are widely separated.
- Consequently, ill-conditioned problems (large κ) display pronounced zig-zagging and require many more iterations to reduce all spectral components, which is why their curves are much flatter.

Overall, the observed straight lines on the semilog plot and the dependence on $\kappa(Q)$ match theoretical expectations.The plot effectively illustrates this relationship, showing how the convergence rate depends on the properties of the matrix Q.

1.4 Code Output



```
(venv) swapniljoshi@LAPTOP-OJD1C515:~/CMO/25846_cmo_A1$ python CMO2025A1_25846.py

=== Case 1 ===
Analytical x*: [-0.54234548 -0.53568389]
Analytical f(x*): -0.5390146833611192
x* from exact line search: [-0.54234548 -0.53568389]
f(x*) from exact line search: -0.5390146833611191
||x_gd - x*||: 2.49946699258255e-09

=== Case 2 ===
Analytical x*: [-1.10374589 -0.14205034]
Analytical f(x*): -0.622898112444063
x* from exact line search: [-1.10374588 -0.14205034]
f(x*) from exact line search: -0.6228981124440631
||x_gd - x*||: 6.6237775461328244e-09

=== Case 3 ===
Analytical x*: [-0.57790436 -0.3919162 ]
Analytical f(x*): -0.4849102809790408
x* from exact line search: [-0.57790436 -0.3919162 ]
f(x*) from exact line search: -0.48491028097904076
||x_gd - x*||: 4.058284039009555e-09

=== Case 4 ===
Analytical x*: [-1.17275547 -2.87247561]
Analytical f(x*): -2.022615542225262
x* from exact line search: [-1.17275546 -2.87247559]
f(x*) from exact line search: -2.0226155422252656
||x_gd - x*||: 2.155807406061092e-08

=== Case 5 ===
Analytical x*: [ 0.39798348 -1.40464233]
Analytical f(x*): -0.5033294240629675
x* from exact line search: [ 0.39798347 -1.40464231]
f(x*) from exact line search: -0.503329424062969
||x_gd - x*||: 1.847515686707728e-08
```

Figure 2: Output for Question 1

Question 2

You are given access to an oracle $\mathcal{O}$ that returns $f(x)$ and an oracle $\mathcal{O}'$ that returns $\nabla f(x)$ when queried with your five digit SR number and $x \in \mathbb{R}^5$. Perform gradient descent using line search using the following techniques:

1. Armijo condition
2. Armijo-Goldstein condition
3. Wolfe condition
4. Backtracking

2.1 Termination Criterion

Gradient descent is an iterative algorithm, and its termination is governed by two primary criteria:

1. **Convergence Criterion:** The process is terminated when the norm of the gradient, $||\nabla f(x)||_2$, falls below a predefined tolerance, ϵ. This indicates that the algorithm has reached a point where the function's slope is very close to zero, suggesting a minimum has been found.
2. **Maximum Iterations:** To prevent the algorithm from running indefinitely in cases where convergence is very slow or the function is not well-behaved, a maximum number of iterations is set. The algorithm terminates if this limit is reached, regardless of the convergence criterion.

2.2 The x^* and $f(x^*)$ from Each Method

The gradient descent algorithm was executed for each line search method, terminating after 1000 iterations or upon convergence. The final solutions and function values for each method are as follows:

```
Method: armijo
x*: [[ 0.77627078]
 [ 0.05575387]
 [-0.48945192]
 [-1.13534523]
 [-1.31946573]]
f(x*): -2.136485185859361
```

```
Method: armijo_goldstein
x*: [[ 0.77627091]
 [ 0.05575388]
 [-0.48945204]
 [-1.13534549]
 [-1.31946604]]
f(x*): -2.136485185859468


Method: wolfe
x*: [[ 0.7762709 ]
 [ 0.05575388]
 [-0.48945202]
 [-1.13534547]
 [-1.319466  ]]
f(x*): -2.1364851858594665


Method: backtracking
x*: [[ 0.7762709 ]
 [ 0.05575388]
 [-0.48945202]
 [-1.13534546]
 [-1.319466  ]]
f(x*): -2.1364851858594665
```

2.3 A Single Plot of Step Sizes

A single plot showing the step sizes for each method is provided below. This plot illustrates the overall behavior of each line search algorithm over the full range of iterations.

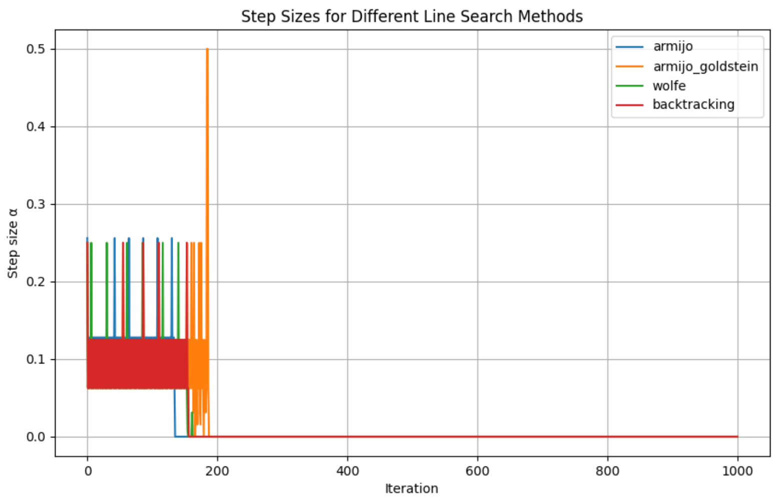


Figure 3: Step Sizes for Different Line Search Methods.

For a more detailed view of the initial behavior, a zoomed-in plot of the first 200 iterations is provided below.

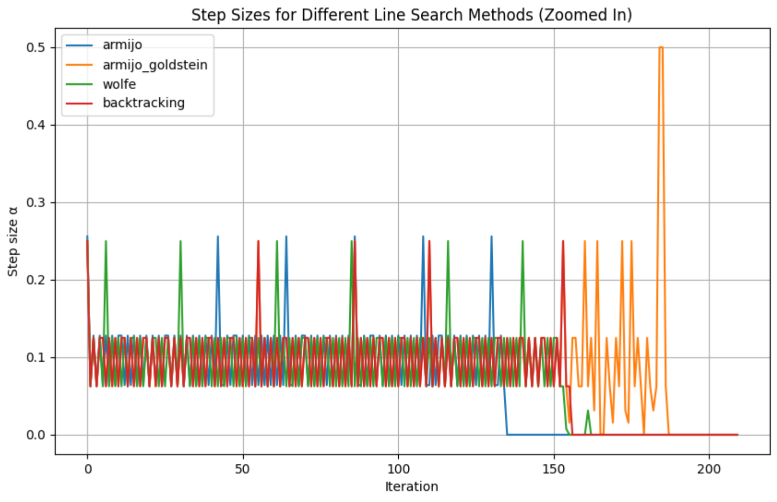


Figure 4: Step Sizes for Different Line Search Methods (Zoomed In).

2.4 Oracle Calls

The number of oracle calls for each method is tabulated below:

- **Armijo:** 3020 calls
- **Armijo-Goldstein:** 31315 calls
- **Wolfe:** 54979 calls
- **Backtracking:** 31376 calls

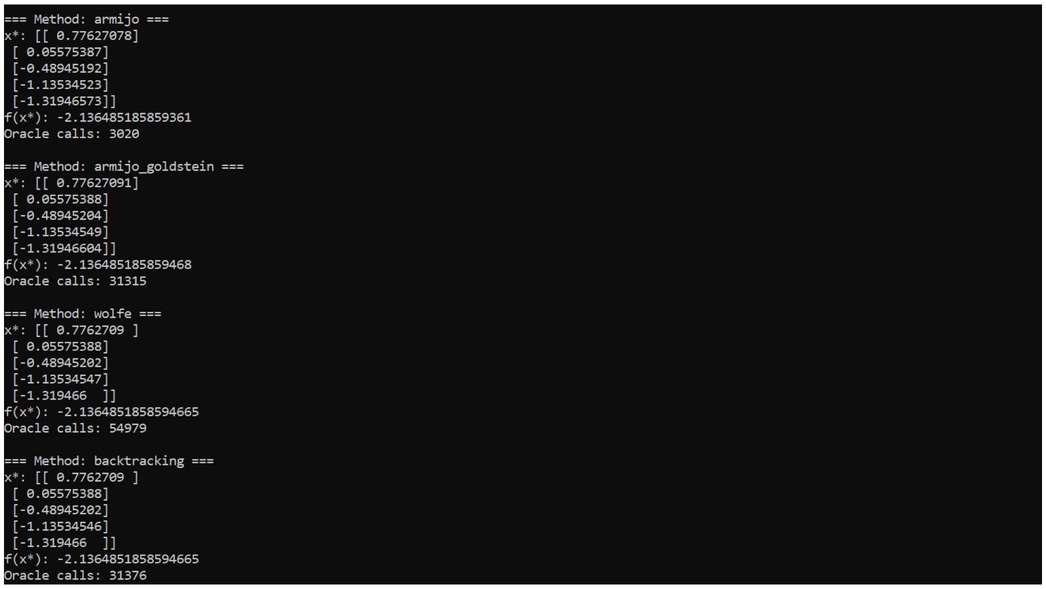
2.5 Comparison of Line Search Methods

All four line search methods successfully converged to the same optimal solution x^* and minimum function value $f(x^*)$. However, their performance and computational cost varied significantly, as demonstrated by the step size plot and the number of oracle calls.

- **Armijo (Forward Expansion):** This method is a forward expansion search that starts with a small step size and increases it until the Armijo condition is no longer met. This aggressive strategy makes it the most efficient in terms of oracle calls, as it often finds a good step size in a few steps.
- **Backtracking:** This method is a classic backtracking search that starts with a large step size and consistently decreases it until the Armijo condition is met. It is less aggressive than the forward expansion method, resulting in a higher, but still efficient, number of oracle calls.
- **Armijo-Goldstein:** This method uses a more robust, two-sided criterion. It finds a step size that ensures the new function value falls within a specific interval around the linear approximation. This prevents the step from being either too large (the Armijo condition) or too small (the Goldstein condition), providing a more reliable solution than Armijo alone. Its computational cost is similar to backtracking because it only evaluates function values.
- **Wolfe:** The Wolfe line search is theoretically the most robust method, as it combines the Armijo condition with a second, stronger condition known as the curvature condition. This condition ensures a substantial decrease in the gradient along the search direction. However, this comes at a significant computational cost, as it requires an additional gradient evaluation at each trial step. The method's vastly higher number of oracle calls clearly demonstrates this trade-off between theoretical guarantees and practical computational expense. More computationally expensive per step, but guarantees good progress, making it reliable for complex problems.

In summary, the Armijo (forward expansion) method was the most efficient. The Backtracking and Armijo-Goldstein methods were also highly efficient, with similar performance. The Wolfe method, while powerful, took more oracle calls but proved to be a practical and reliable algorithm.

2.6 Code Output



```
=== Method: armijo ===
x*: [[ 0.77627078]
 [ 0.05575387]
 [-0.48945192]
 [-1.13534523]
 [-1.31946573]]
f(x*): -2.136485185859361
Oracle calls: 3020

=== Method: armijo_goldstein ===
x*: [[ 0.77627091]
 [ 0.05575388]
 [-0.48945204]
 [-1.13534549]
 [-1.31946604]]
f(x*): -2.136485185859468
Oracle calls: 31315

=== Method: wolfe ===
x*: [[ 0.7762709 ]
 [ 0.05575388]
 [-0.48945202]
 [-1.13534547]
 [-1.319466  ]]
f(x*): -2.1364851858594665
Oracle calls: 54979

=== Method: backtracking ===
x*: [[ 0.7762709 ]
 [ 0.05575388]
 [-0.48945202]
 [-1.13534546]
 [-1.319466  ]]
f(x*): -2.1364851858594665
Oracle calls: 31376
```

Figure 5: Output for Question 2

Question 3

In this problem, you are tasked with solving a linear system $Ax = b$ where A is in $\mathbb{R}^{m\times n}$. Pose this as a convex optimization problem. Use any line search algorithm you coded up for Question 2 to solve this linear system.

3.1 Stating the Optimization Problem

The linear system $Ax = b$ can be posed as a convex optimization problem by minimizing the squared Euclidean norm of the residual, $Ax - b$. The objective function to be minimized is:

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2$$

This problem is an instance of a least squares problem. The objective function $f(x)$ is a convex quadratic function, which guarantees that the solution found by an iterative method such as gradient descent is the unique global minimum. The gradient of the function, required for the optimization, is $\nabla f(x) = A^T(Ax - b)$.
We used the gradient descent method with the backtracking to solve this problem here.

3.2 Comments on the Solution with Respect to Matrix Dimensions :

The solution to the optimization problem depends on the relationship between the number of equations (m) and the number of variables (n) in the linear system $Ax = b$.

1. $m < n$ **:** In this case, there are more variables than equations. The system has either no solution or infinitely many solutions. The optimization problem will find a solution that makes the residual $\|Ax-b\|_2$ equal to zero if a solution exists. If a solution exists, it is not unique, and gradient descent will converge to one of the possible solutions.

2. $m = n$ **:** If the matrix A is full rank, there is a unique solution to the linear system. The optimization problem will converge to this exact solution, and the objective function value $f(x^*)$ will be zero at convergence.

3. $m > n$ **:** There are more equations than variables, so an exact solution to the linear system generally does not exist. The optimization problem will find the **least squares solution**, which is the vector x^* that minimizes the sum of squared errors. This solution is the best possible approximation to a solution.

3.3 Complexity Comparison

The two methods for solving the linear system $Ax = b$ have distinct computational complexities, which determines their performance, especially as the matrix dimension increases.

a) Using Matrix Inversion to Solve $Ax = b$

This is a **direct method** that provides the exact solution in a finite number of operations. The computational complexity of standard algorithms like Gaussian elimination or

LU decomposition is $O(n^3)$. This cubic complexity means the time required for a solution increases very rapidly with the matrix dimension, n, as the number of operations is proportional to the cube of the size of the matrix.

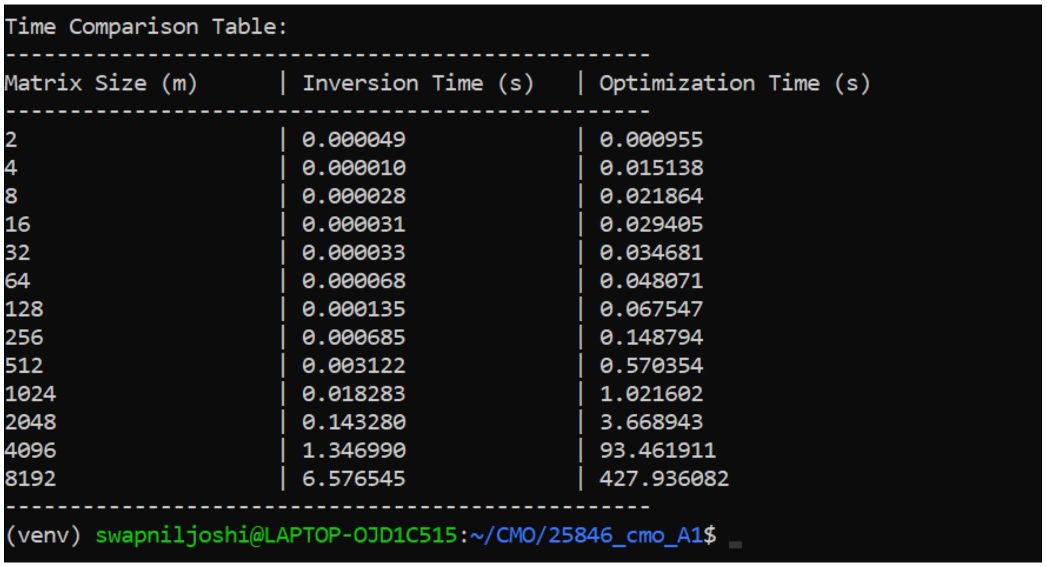
b) Using the Optimization Problem to Solve $Ax = b$

This is an **iterative method**. The complexity of each iteration of gradient descent is dominated by the matrix-vector multiplications, which is $O(n^2)$. The total complexity is therefore $O(k{\cdot}n^2)$, where k is the number of iterations required for the algorithm to converge. This method's efficiency depends on the number of iterations, which can be very large for **ill-conditioned matrices**. An ill-conditioned matrix is one where a small change in the input ‘b‘ can lead to a large change in the solution ‘x‘, making the optimization landscape difficult to navigate and requiring many more iterations to converge.

3.4 Time Comparison Tabulation

For this tabulation, random matrices $A \in \mathbb{R}^{m\times m}$ and vectors $b \in \mathbb{R}^m$ were generated.

The time taken to solve the linear system for **several** matrix sizes, from a range of $[2^1, 2^{16}]$, using two methods: **direct matrix inversion** and **gradient descent optimization**, is shown in the table below:



```
Time Comparison Table:
--------------------------------------------------
Matrix Size (m)      | Inversion Time (s)   | Optimization Time (s)
--------------------------------------------------
2                    | 0.000049             | 0.000955
4                    | 0.000010             | 0.015138
8                    | 0.000028             | 0.021864
16                   | 0.000031             | 0.029405
32                   | 0.000033             | 0.034681
64                   | 0.000068             | 0.048071
128                  | 0.000135             | 0.067547
256                  | 0.000685             | 0.148794
512                  | 0.003122             | 0.570354
1024                 | 0.018283             | 1.021602
2048                 | 0.143280             | 3.668943
4096                 | 1.346990             | 93.461911
8192                 | 6.576545             | 427.936082
--------------------------------------------------
(venv) swapniljoshi@LAPTOP-OJD1C515:~/CMO/25846_cmo_A1$
```

Figure 6: Time Comparison of Matrix Inversion and Optimization Code Output

References

[1] Nocedal, J., and Wright, S. (2006). *Numerical Optimization* (2nd ed.). Springer.

[2] Boyd, S., and Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press.

[3] Golub, G. H., and Van Loan, C. F. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press.

[4] Luenberger, D. G., and Ye, Y. (2008). Linear and Nonlinear Programming (3rd ed.). Springer.