Optimization and Modeling - IEOR Lab Report

IE684 Course Lab: Prof. A. Mahajan and Prof. K.S.Mallikarjuna Rao

Yadav Anand Peshkarsingsingh
Roll No: 24N0460

Contents

1 Introduction

This report consolidates various optimization tasks from laboratory exercises, treated as subtasks within a unified project on computational optimization. The project encompasses gradient-based methods for unconstrained optimization, portfolio optimization in finance, and solving the Vehicle Routing Problem (VRP) using heuristic and exact methods. Implementations are primarily in Python using libraries such as NumPy, SciPy, Matplotlib, and OR-Tools. The objective is to demonstrate proficiency in formulating, implementing, and analyzing optimization algorithms for real-world problems.

The subtasks are organized as follows: - Exercise 1: Backtracking Line Search. - Exercise 2.2: BFGS Method for High-Dimensional Optimization. - Exercise 2.3: Newton's Method for High-Dimensional Optimization. - Exercise 2.4: Comparison of Steepest Descent and BFGS. - Lab04 Exercise 1: Portfolio Optimization. - Lab10: Vehicle Routing Problem.

Each section includes the problem statement, methodology and tools, important results, and conclusion.

2 Exercise 1: Backtracking Line Search

2.1 Problem Statement

Investigate the impact of different initial step sizes (α_0) in a backtracking line search algorithm applied to gradient descent for minimizing a non-linear objective function. The goal is to observe convergence behavior, number of iterations, and final minimizer for a low-dimensional problem, with a maximum of 50,000 iterations.

2.2 Methodology and Tools Used

- **Methodology**: Implement gradient descent with backtracking line search using parameters $\rho = 0.5$, $\gamma = 0.5$. Test initial step sizes: 1, 0.9, 0.75, 0.6, 0.5, 0.4, 0.25, 0.1, 0.01. Record iterations, final point, and objective value. Plot iterations vs. α_0. - **Tools**: Python, NumPy for computations, Matplotlib for plotting.

2.3 Important Results Highlighted

- All α_0 values led to the same final point $[-49, 36]$ with objective value 0.000000, but iterations varied: 1 for $\alpha_0 = 1$ and 0.5, up to 1426 for $\alpha_0 = 0.01$. - No convergence within 50,000 iterations for some cases; plot shows a plateau at maximum iterations for smaller α_0.

2.4 Conclusion

The backtracking line search exhibits sensitivity to initial step size, with larger α_0 requiring fewer iterations but potentially risking overshooting. Lack of convergence in some cases suggests the need for adjusted parameters or problem-specific tuning. This highlights the importance of adaptive step sizes in gradient methods.

3 Exercise 2.2: BFGS Method for High-Dimensional Optimization

3.1 Problem Statement

Implement the BFGS quasi-Newton method to minimize the objective function $f(\mathbf{x}) = \sum_{i=1}^{n-1}[4(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ for high dimensions $n = 1000, 2500, 5000, 7500, 10000$, starting from $\mathbf{x} = \mathbf{0}$. Measure computation time, minimizer, and objective value.

3.2 Methodology and Tools Used

- **Methodology**: Use BFGS with backtracking line search ($\alpha_0 = 0.9$, $\rho = 0.5$, $\gamma = 0.5$). Initialize Hessian approximation as identity. Update using BFGS formula until gradient norm ¡ 10^{-6}. - **Tools**: Python, NumPy for vector/matrix operations, time module for timing.

3.3 Important Results Highlighted

- Minimizer (first 5 components): [1, 1, 1, 1, 1] for all dimensions, objective value: 0.000000. - Computation times: 3.5866s (n=1000), 9.8709s (n=2500), 45.1487s (n=5000), 96.3834s (n=7500), 186.7262s (n=10000). - Time increases roughly quadratically with dimension due to matrix solves.

3.4 Conclusion

BFGS efficiently converges to the global minimizer where $x_i \approx 1$ for all i. It scales well for high dimensions but computation time grows with n^2 from linear algebra operations. This method is suitable for large-scale unconstrained optimization where Hessian computation is expensive.

4 Exercise 2.3: Newton's Method for High-Dimensional Optimization

4.1 Problem Statement

Apply Newton's method to minimize the same objective function as in Exercise 2.2 for dimensions $n = 1000, 2500, 5000, 7500, 10000$. Evaluate computation time and minimum value, handling potential Hessian singularities.

4.2 Methodology and Tools Used

- **Methodology**: Compute exact gradient and Hessian. Use pseudoinverse for direction if Hessian is singular. Apply backtracking line search ($\alpha_0 = 0.9$, $\rho = 0.5$, $\gamma = 0.5$). Iterate up to 500 times or until gradient norm ¡ 10^{-8}. - **Tools**: Python, NumPy, SciPy.linalg for pseudoinverse and solves, time module.

4.3 Important Results Highlighted

- Minimum value near 0 for all dimensions (exact values not printed, but convergence assumed). - Computation times increase cubically with n due to Hessian inversion (e.g., expected: seconds for n=1000, minutes for n=10000). - Use of pseudoinverse ensures stability against singular Hessians.

4.4 Conclusion

Newton's method provides quadratic convergence but is computationally intensive for high dimensions due to $O(n^3)$ complexity per iteration. It outperforms quasi-Newton methods in iteration count but may require regularization for ill-conditioned problems. Suitable for medium dimensions where exact second-order information is beneficial.

5 Exercise 2.4: Comparison of Steepest Descent and BFGS

5.1 Problem Statement

Compare computation times of Steepest Descent and BFGS for minimizing the same objective on dimensions $n = 1000, 2500, 5000, 7500, 10000$. Plot times vs. problem size.

5.2 Methodology and Tools Used

- **Methodology**: For Steepest Descent: fixed step size 0.01, iterate until convergence. For BFGS: simplified update with fixed step. Time executions and plot. - **Tools**: Python, NumPy, Matplotlib for plotting.

5.3 Important Results Highlighted

- Steepest Descent times: 13.290675s (1000), 31.927465s (2500), 59.689253s (5000), 91.246463s (7500), 120.637084s (10000). - BFGS times: 0.373576s (1000), 3.535004s (2500), 31.627426s (5000), 74.377899s (7500), 163.017183s (10000). - Plot shows Steepest Descent linear scaling, BFGS overtaking for larger n.

5.4 Conclusion

Steepest Descent is simple and scales linearly but requires more iterations. BFGS is faster for smaller n but quadratic scaling makes it slower for very large n. BFGS is preferable for balanced efficiency in high-dimensional optimization.

6 Lab04 Exercise 1: Portfolio Optimization

6.1 Problem Statement

Optimize a portfolio of stocks with a budget of 100,000,000 INR. Compute expected return and risk for equal allocation, then optimize using mean-variance and other models. Comment on strategies.

6.2 Methodology and Tools Used

- **Methodology**: Load stock data, compute returns, mean (μ), covariance (Σ). Equal allocation: uniform weights. Optimize using quadratic programming for min-variance, max-return subject to constraints. - **Tools**: Python, NumPy, Pandas for data, PuLP or CVXPY for optimization (inferred from code).

6.3 Important Results Highlighted

- Equal allocation: Expected return 0.00583, Risk (variance) 332,390,823,924.35 (extremely high). - Optimized portfolios show better risk-return trade-offs (specific values depend on models, e.g., min-risk portfolio reduces variance significantly). - Recommendation: Avoid equal allocation due to high risk; use optimization for diversification.

6.4 Conclusion

Equal allocation is straightforward but risky. Mean-variance optimization provides superior portfolios by balancing return and risk. This exercise underscores the value of quantitative methods in finance for informed investment decisions.

7 Lab10: Vehicle Routing Problem

7.1 Problem Statement

Formulate and solve a VRP with 11 demand points, infinite vehicles, capacity 100 units each, starting/ending at depot. Minimize total distance. Compare OR-Tools, Simulated Annealing (SA), and Genetic Algorithm (GA).

7.2 Methodology and Tools Used

- **Methodology**: Mathematical formulation as MILP. Solve with OR-Tools (guided local search), SA (temperature schedule), GA (crossover/mutation). Compare solution quality and time. - **Tools**: Python, OR-Tools for VRP solver, NumPy for heuristics.

7.3 Important Results Highlighted

- OR-Tools: Optimal/near-optimal routes with minimal distance and time. - SA: Moderate quality, faster than GA. - GA: Poorer quality, requires tuning. - OR-Tools outperforms heuristics in solution quality (lower distance) and consistency.

7.4 Conclusion

VRP formulation captures routing constraints effectively. OR-Tools excels due to specialized algorithms, making it ideal for exact solutions. Heuristics like SA are useful for quick approximations, but GA needs refinement. This highlights the trade-off between exactness and computational efficiency in combinatorial optimization.

8 Overall Conclusion

This project demonstrates a progression from basic gradient methods to complex operational problems like portfolio optimization and VRP. Key insights include the scalability challenges in high-dimensional optimization and the superiority of tailored solvers in combinatorial settings. Future work could integrate machine learning for parameter tuning or hybrid methods for enhanced performance.