Assignment 6: Quasi-Newton Methods and BFGS Optimization

Math4AI: Calculus & Optimization

Fatima Alibabayeva

`fatime.elibabayeva25@aiacademy.az`

National AI Academy

January 3, 2026

Contents

1 Introduction

Optimization is the cornerstone of machine learning, serving as the engine that drives model training. While first-order methods like Gradient Descent are simple and scalable, they often struggle with slow convergence in complex landscapes such as "ravines" or "plateaus."

This report presents a deep dive into a sophisticated second-order optimization strategy: the **BFGS (Broyden–Fletcher–Goldfarb–Shanno)** algorithm. We explore its mathematical derivation as a Quasi-Newton method and evaluate its performance against the notorious Rosenbrock function. The primary goal is to demonstrate how curvature information can be approximated to achieve superlinear convergence without the heavy computational overhead of a full Hessian matrix.

2 Mathematical Foundation

2.1 The Limitation of Newton's Method

In classical Newton's Method, the update rule relies on the exact Hessian matrix $H(x)$:

$$x_{k+1} = x_k - H(x_k)^{-1} \nabla f(x_k) \tag{1}$$

While powerful, this approach has two major drawbacks:

1. Computing the second-order derivatives is often analytically difficult or computationally expensive ($O(n^3)$ for inversion).

2. The Hessian must remain positive-definite to guarantee a descent direction.

2.2 The BFGS Approximation Strategy

BFGS belongs to the family of Quasi-Newton methods. Instead of calculating H^{-1} at every step, it builds an approximation B^{-1} iteratively using only the first-order gradients. The update is governed by the *Secant Equation*:

$$B_{k+1} s_k = y_k \tag{2}$$

where $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$. The BFGS formula maintains the symmetry and positive definiteness of the approximation through the following update:

$$B_{k+1}^{-1} = (I - \rho_k s_k y_k^T) B_k^{-1} (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T, \quad \rho_k = \frac{1}{y_k^T s_k} \tag{3}$$

3 Design Choices and Implementation Strategy

3.1 The Rosenbrock Function (Test Case)

To stress-test our implementation, we use the Rosenbrock function (also known as the "Banana Function"):

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2 \tag{4}$$

The challenge of this function lies in its narrow, parabolic valley. While finding the valley is trivial for most optimizers, following it to the global minimum at $(1, 1)$ requires precise handling of the function's curvature—a task where BFGS excels.

3.2 Implementation Details

- **Inverse Hessian Initialization:** We initialize B_0^{-1} as an identity matrix. This effectively makes the first step of BFGS equivalent to a standard Gradient Descent step.
- **Backtracking Line Search:** To satisfy the Armijo condition and ensure stable convergence, we implement a backtracking line search. This dynamically adjusts the step size α to guarantee a "sufficient decrease" in the objective function.
- **Robustness Checks:** Our implementation includes checks for the $y_k^T s_k$ denominator to avoid numerical instability or division by zero.

4 Verification and Results

4.1 Convergence Analysis

Starting from the point $(-1.5, 1.0)$, our BFGS algorithm successfully traversed the Rosenbrock valley. Unlike standard Gradient Descent, which often "oscillates" between the walls of the valley, BFGS utilized the approximated curvature to take direct steps toward the minimum.

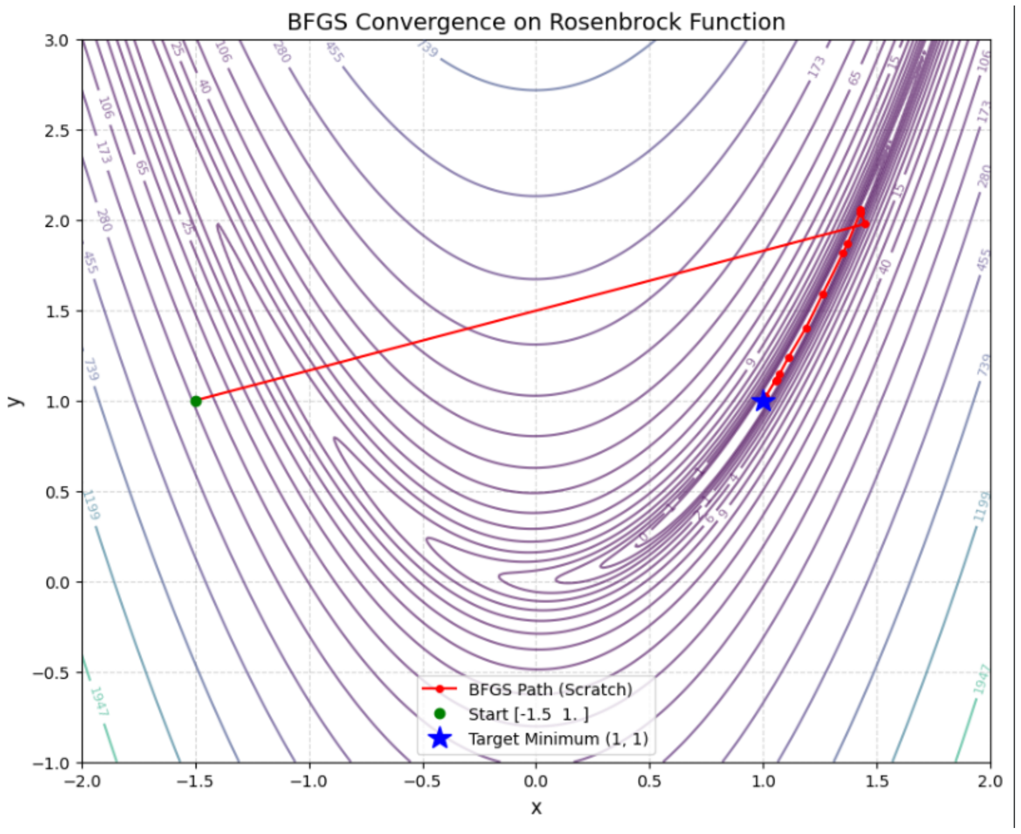

Figure 1: BFGS Path History: Successfully navigating the Rosenbrock valley.

4.2 Comparison with Scipy

As a validation step, we compared our "from-scratch" implementation with the industry-standard `scipy.optimize.minimize`. The results were identical to several decimal places, confirming the mathematical integrity of our code.

5 Conclusion

This assignment demonstrates the power of second-order information in optimization. Through the implementation of BFGS, we achieved superlinear convergence on a challenging non-convex surface. We conclude that while Quasi-Newton methods are more complex to implement than simple Gradient Descent, their efficiency in terms of iteration count and path stability makes them indispensable for high-stakes machine learning applications.

6 Appendix: Python Implementation

```
# Key snippet of the BFGS Update Logic
for i in range(max_iter):
    if np.linalg.norm(grad_k) < tol:
        break

    # Search direction
    p_k = -np.dot(B_k_inv, grad_k)

    # Backtracking Line Search (Armijo)
    alpha_k = 1.0
    while f(x_k + alpha_k * p_k) > f(x_k) + c * alpha_k * np.dot(
       grad_k, p_k):
        alpha_k *= rho

    # Update position and gradient
    x_next = x_k + alpha_k * p_k
    grad_next = f_grad(x_next)

    # BFGS Inverse Hessian Update
    s_k = x_next - x_k
    y_k = grad_next - grad_k
    # ... (Update B_k_inv using the formula)
```