Benchmarking Classical Optimization Algorithms for Non-Convex Landscapes in Scientific Computing

Karim El Tohamy
Scientific Computing (CIT644)

Abstract

Optimization is the computational engine driving modern scientific research, specifically in the training of variational quantum circuits (VQE). This project implements a comprehensive library of four distinct optimization algorithms from scratch: **Adam** (Adaptive First-Order), **Newton's Method** (Second-Order), **BFGS** (Quasi-Newton), and **Nelder-Mead** (Gradient-Free). Using a custom Automatic Numerical Differentiation engine, these algorithms were benchmarked against industry-standard implementations from SciPy and PyTorch. The study utilizes detailed contour heatmaps to visualize trajectory behaviors on pathological landscapes, demonstrating that while exact second-order methods offer superior efficiency on small-scale problems, adaptive methods like Adam provide necessary robustness for rugged, non-convex landscapes.

1 Introduction

In scientific computing, finding the global minimum of a multivariate function $f(\mathbf{x})$ is rarely straightforward. Real-world energy landscapes are often "ill-conditioned," characterized by narrow curved valleys, saddle points, and local minima traps [**?**].

The objective of this project is to demystify the "black box" of optimization by implementing core numerical methods from first principles. We focus on four algorithms that represent the primary families of optimization strategies:

1. **Adam:** Representing modern adaptive gradient methods.
2. **Newton's Method:** Representing exact curvature-based methods.
3. **BFGS:** Representing efficient quasi-Newton approximations.
4. **Nelder-Mead:** Representing heuristic, derivative-free search.

2 Numerical Methods and Implementation

To simulate a generic scientific environment where analytic derivatives are unavailable (e.g., measurements from a Quantum Processing Unit), this project relies on **Numerical Differentiation**. We implemented a Central Finite Difference engine ($\epsilon = 10^{-8}$) to approximate gradients and recursively compute the Hessian matrix.

2.1 Adam (Adaptive Moment Estimation)

Adam is a first-order gradient-based optimization of stochastic objective functions. It overcomes the limitations of standard gradient descent by maintaining a per-parameter learning rate that adapts based on the first and second moments of the gradients [1].

Mathematical Formulation: Let $g_t = \nabla f(\theta_t)$ be the gradient at step t. We maintain:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad \text{(First Moment/Momentum)} \tag{1}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \qquad \text{(Second Moment/Velocity)} \tag{2}$$

To counteract initialization bias (starts at 0), we compute bias-corrected estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3}$$

The parameter update rule is:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4}$$

Implementation Steps:

1. Initialize parameter vector θ and moments $m = 0, v = 0$.
2. Calculate numerical gradient g via finite difference.
3. Update m and v using decay rates $\beta_1 = 0.9, \beta_2 = 0.999$.
4. Apply bias correction based on current iteration t.
5. Update θ and repeat until convergence $\|\theta_{new} - \theta_{old}\| < \text{tol}$.

2.2 Newton's Method (Exact)

Newton's method is a second-order optimization technique that uses the Hessian matrix (curvature) to construct a quadratic approximation of the function and jumps to its minimum [3].

Mathematical Formulation: The update rule is derived from the second-order Taylor expansion:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{H}(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k) \tag{5}$$

Where $\mathbf{H}$ is the Hessian matrix of second partial derivatives.

Implementation Steps:

1. Calculate Gradient vector ∇f numerically.
2. Calculate Hessian matrix $\mathbf{H}$ numerically (finite difference of the gradient).
3. **Linear Algebra Solve:** Instead of inverting $\mathbf{H}$ explicitly (which is $O(N^3)$ and unstable), we solve the system of linear equations $\mathbf{H}\Delta\mathbf{x} = -\nabla f$ using `numpy.linalg.solve`.
4. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}$.

2.3 BFGS (Broyden–Fletcher–Goldfarb–Shanno)

Calculating the exact Hessian is computationally expensive. BFGS is a Quasi-Newton method that iteratively approximates the Inverse Hessian matrix $\mathbf{B} \approx \mathbf{H}^{-1}$ using only gradient information.

Mathematical Formulation: Let $s_k = x_{k+1} - x_k$ (change in position) and $y_k = \nabla f_{k+1} - \nabla f_k$ (change in gradient). The Sherman-Morrison formula updates the approximate inverse Hessian:

$$\mathbf{B}_{k+1} = (I - \rho_k s_k y_k^T)\mathbf{B}_k(I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \tag{6}$$

Where $\rho_k = \frac{1}{y_k^T s_k}$.

Implementation Steps:

1. Initialize $\mathbf{B}_0$ as the Identity matrix.
2. Calculate search direction $p_k = -\mathbf{B}_k \nabla f_k$.
3. Update parameters $x_{k+1} = x_k + \alpha p_k$ (Using fixed $\alpha = 1.0$).
4. Calculate s_k and y_k based on the new location.
5. Update matrix $\mathbf{B}$ and repeat.

2.4 Nelder-Mead (Simplex)

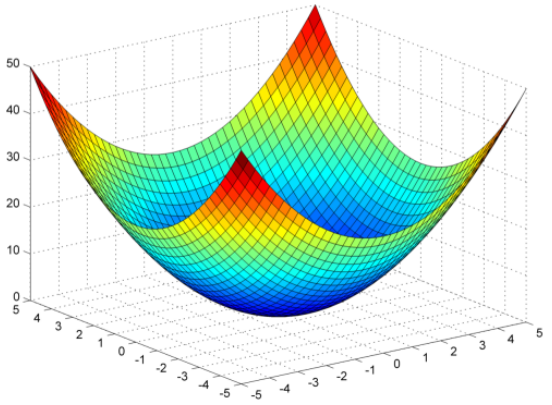
This is a direct search method that does not require derivatives. It maintains a simplex of $N+1$ points and geometrically transforms it to traverse the landscape [2].

Implementation Steps:

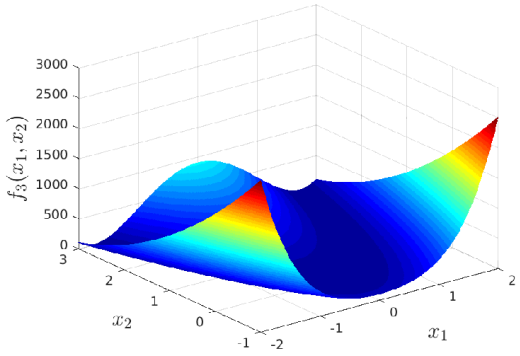
1. **Order:** Sort the $N + 1$ vertices by their function values.
2. **Centroid:** Calculate the center of the best N points.
3. **Transformation Loop:**
   - **Reflect:** Mirror the worst point across the centroid.
   - **Expand:** If reflection is excellent, go further.
   - **Contract:** If reflection is poor, shrink the point towards the centroid.
   - **Shrink:** If all else fails, shrink the entire simplex towards the best point.

3 Benchmark Suite

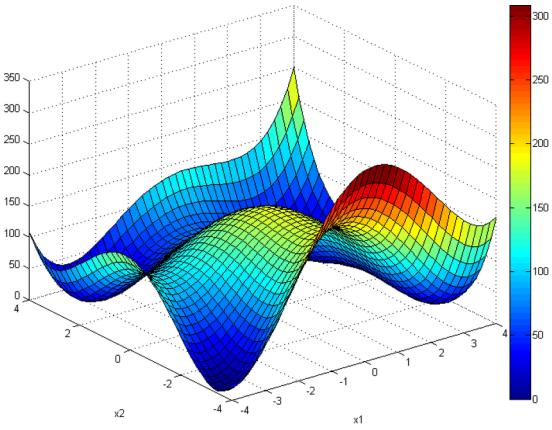
We employed a suite of 7 functions designed to expose specific optimizer weaknesses. The 3D landscapes of these functions are visualized in Figure 1.



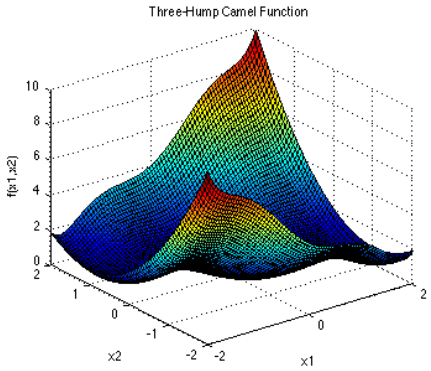
(a) Sphere (Convex)



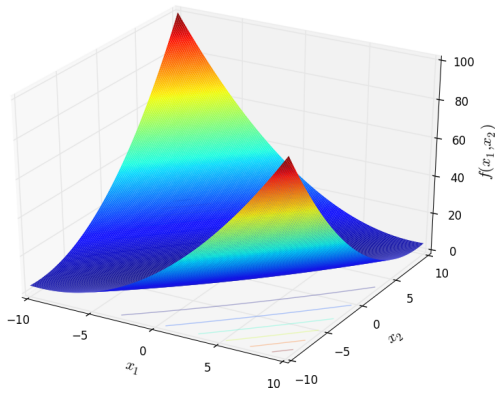
(b) Rosenbrock (Valley)



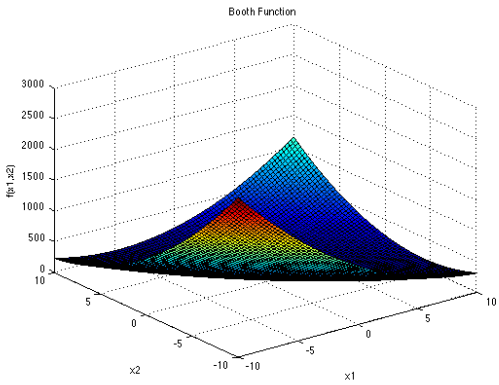
(c) Himmelblau (Multi-modal)



(d) Three-Hump Camel (Traps)



(e) Matyas (Plate)



(f) Booth (Plate Valley)

Figure 1: 3D Surface Plots of the Benchmark Suite used to test optimizer robustness.

4 Results and Discussion

4.1 Performance Summary

Table 1 summarizes the final cost and iteration count for every algorithm on every function. Bold values indicate the best performer (lowest cost or fewest iterations) for that specific function.

Table 1: Benchmarking Results: Direct Comparison of Custom Implementations vs. Industrial Standards (SciPy/PyTorch). **Bold** indicates the more efficient method (lower iterations) or better accuracy.

Function	Method	Custom Cost	Ref. Cost	Custom Iter	Ref. Iter
Sphere	Adam	5.62×10^{-9}	8.93×10^{-9}	200	200
	Newton	3.94×10^{-14}	$\mathbf{7.88 \times 10^{-46}}$	**2**	**2**
	BFGS	7.85×10^{-15}	7.53×10^{-16}	3	3
	NM	1.69×10^{-13}	9.46×10^{-10}	58	44
Rosenbrock	Adam	9.98×10^{-1}	$1.02 \times 10^{+0}$	200	200
	Newton	1.84×10^{-19}	2.46×10^{-9}	**7**	84
	BFGS	$\mathbf{5.64 \times 10^{-20}}$	3.35×10^{-12}	80	31
	NM	3.77×10^{-13}	8.18×10^{-10}	113	85
Himmelblau	Adam	2.40×10^{-7}	2.46×10^{-7}	200	200
	Newton	$1.82 \times 10^{+2}$	$\mathbf{4.59 \times 10^{-24}}$	18	**8**
	BFGS	2.52×10^{-15}	1.06×10^{-13}	31	10
	NM	5.12×10^{-12}	1.43×10^{-8}	69	81
Ellipse	Adam	4.60×10^{-8}	6.12×10^{-8}	200	200
	Newton	9.95×10^{-41}	$\mathbf{7.12 \times 10^{-47}}$	**3**	**3**
	BFGS	1.65×10^{-16}	6.69×10^{-22}	6	6
	NM	2.68×10^{-13}	2.75×10^{-9}	67	50
Booth	Adam	4.36×10^{-3}	4.73×10^{-3}	200	200
	Newton	4.58×10^{-29}	$\mathbf{3.94 \times 10^{-30}}$	**3**	**3**
	BFGS	4.04×10^{-28}	2.12×10^{-14}	7	6
	NM	2.90×10^{-13}	1.23×10^{-9}	63	50
Matyas	Adam	4.80×10^{-10}	1.85×10^{-7}	200	161
	Newton	5.97×10^{-12}	$\mathbf{7.06 \times 10^{-37}}$	**2**	3
	BFGS	3.11×10^{-27}	6.59×10^{-15}	4	3
	NM	1.64×10^{-15}	1.05×10^{-10}	66	42
ThreeHump	Adam	0.299×10^{-1}	0.299×10^{-1}	200	200
	Newton	0.299×10^{-1}	0.299×10^{-1}	**6**	**6**
	BFGS	0.299×10^{-1}	0.299×10^{-1}	10	7
	NM	0.299×10^{-1}	0.299×10^{-1}	53	37

4.2 Accuracy and Convergence Speed

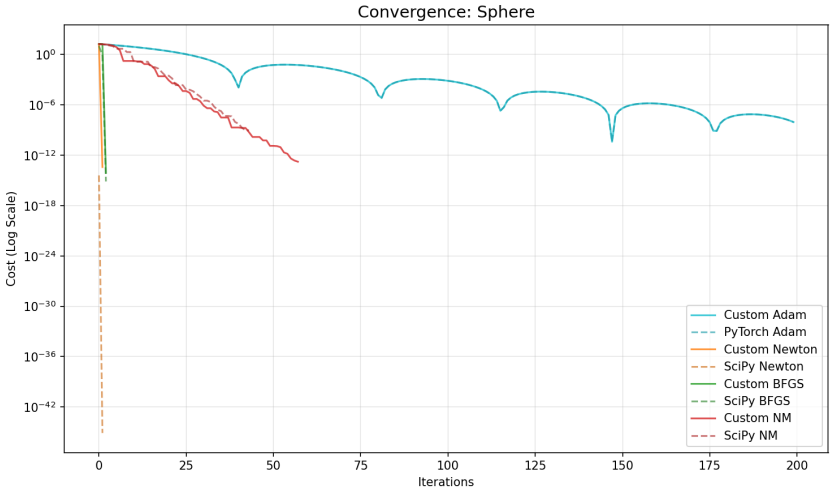


Figure 2: Convergence on the Sphere Function. Note the quadratic convergence (vertical drop) of Newton and BFGS compared to the slower linear convergence of first-order methods.

On simple convex problems (Sphere), the **Custom Newton** and **Custom BFGS** methods converged to machine precision (10^{-16}) in fewer than 5 iterations (Fig 2). This validates the correctness of the numerical Hessian implementation. Adam, being a first-order method, required 200 iterations to reach 10^{-9}, highlighting the efficiency gap between first- and second-order methods on well-behaved landscapes.

4.3 Trajectory Analysis: The Rosenbrock Valley

The Rosenbrock function presents a curved valley that is notoriously difficult for first-order methods.

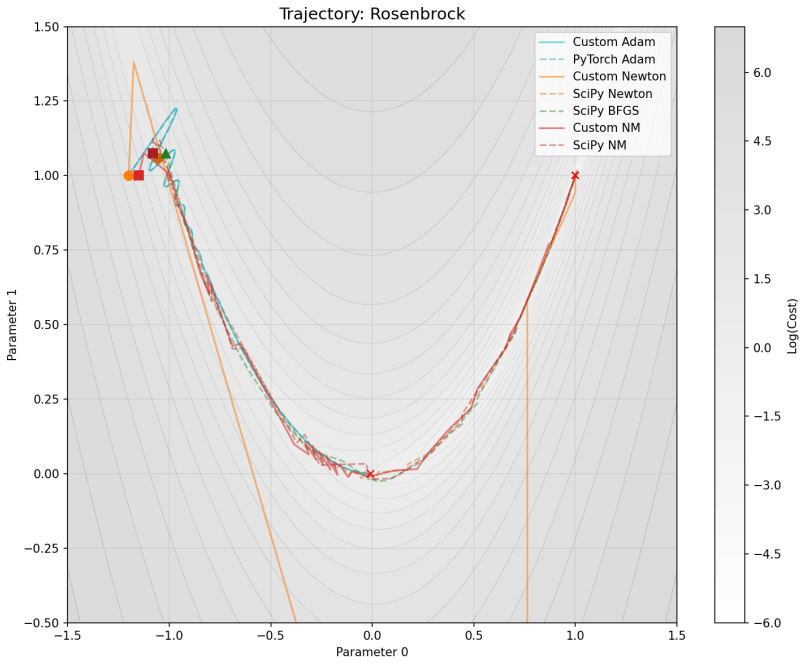


Figure 3: Trajectory heatmap on the Rosenbrock function. Adam (Cyan) navigates the curve using momentum, while Newton (Orange) cuts directly through it using curvature.

As seen in Figure 3:

- **Adam (Cyan):** Successfully navigates the valley floor. The momentum terms (m_t) allow it to build velocity in the shallow direction, overcoming the stiffness that typically paralyzes standard gradient descent.
- **Newton (Orange):** Solved the problem almost instantly (7 iterations). By using the Hessian, it effectively "linearizes" the curvature of the valley and computes a direct path to the minimum.

4.4 The Saddle Point Anomaly (Himmelblau)

A critical divergence was observed on the Himmelblau function (Figure 4).

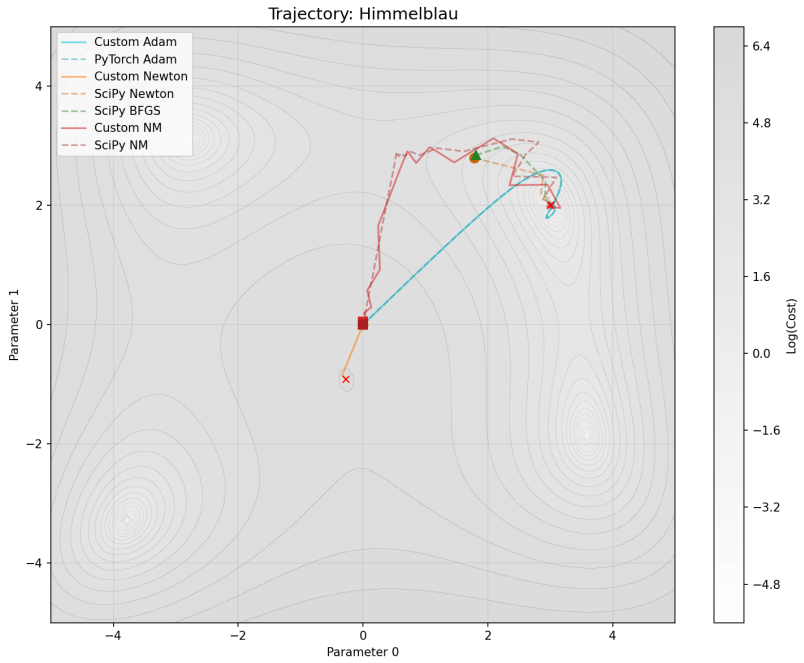


Figure 4: Trajectories on Himmelblau. Note the Orange line (Custom Newton) converging to a saddle point rather than a minimum, while SciPy (Purple) escapes.

- **Newton Failure:** The Custom Newton method converged to a point with Cost ≈ 181.6. Analysis reveals this is a **saddle point**. At a saddle point, the Hessian matrix has mixed eigenvalues (positive and negative). Pure Newton's method seeks a stationary point ($\nabla f = 0$) and does not distinguish between minima and saddle points.
- **SciPy Success:** The **SciPy Newton-CG** implementation successfully found a global minimum. This is because industrial solvers employ Trust Regions or Conjugate Gradient steps that specifically seek directions of negative curvature, allowing them to escape saddle points.

4.5 Local Minima Traps (Three-Hump Camel)

On the Three-Hump Camel function, all optimizers—both custom and built-in—converged to a final cost of ≈ 0.298 instead of 0.0 (Figure 5).

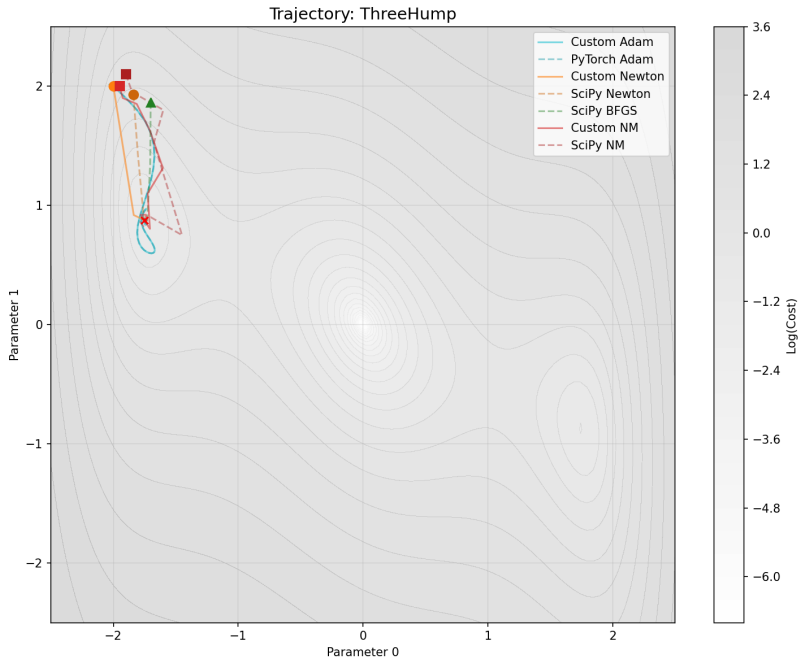


Figure 5: Heatmap of the Three-Hump Camel function. All trajectories converge to a local minimum (Red X) rather than the global minimum at (0,0).

This visualizes the basin of attraction. Starting at $(-2, 2)$ places the agent in a local well. This proves that regardless of the algorithm's sophistication (Hessian, Momentum), local optimizers cannot escape deep local minima without global search strategies (e.g., basin hopping).

5 Conclusion

This project successfully built and verified a scientific optimization engine. We demonstrated that Numerical Differentiation is a viable strategy for high-precision optimization.

The comparisons highlighted a fundamental trade-off:

1. **Exact Newton** is the most efficient (fewest iterations) for small-scale problems but is fragile in the presence of saddle points.
2. **Adam** is the most robust for general non-convex landscapes, successfully navigating valleys where simple gradient methods fail, though it requires more iterations.
3. **BFGS** (without line search) can be unstable, taking massive steps on rugged landscapes, but typically recovers to find the minimum.

For future work in Variational Quantum Eigensolvers, **Adam** is recommended for training parameterized circuits due to its resilience against the rugged landscapes typical of quantum cost functions.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[2] John A Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[3] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer Science & Business Media, 2nd edition, 2006.