Newton-Raphson Method in 3D Space Using Hessian Matrix in Python

Kerem Adalı

March 21, 2025

Abstract

This report presents a mini-project that applies the Newton-Raphson method in 3D space using the Hessian matrix to locate critical points of the function

$$f(x, y) = 2\sin(x) + 3\cos(y).$$

We discuss the impact of different initial vectors and step size parameters on convergence, provide a step-by-step explanation of the algorithm, showcase code snippets and visualizations, and conclude with insights learned from the implementation.

1 Introduction

The goal of this project is to implement a multivariate version of the Newton-Raphson method in Python to find the critical points of the function

$$f(x, y) = 2\sin(x) + 3\cos(y),$$

which can be interpreted as a surface in a three-dimensional space (x, y, and z). In the context of optimization, the Newton-Raphson method leverages first and second derivatives (via the Hessian matrix) to iteratively approximate solutions.

1.1 Algorithm Overview

The Newton-Raphson method is widely used for finding the roots of a real-valued function as well as for optimization problems. The method typically uses the formula:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha H^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k),$$

where:

- $\mathbf{x}_k$ is the current approximation,
- $\nabla f(\mathbf{x}_k)$ is the gradient vector,
- $H(\mathbf{x}_k)$ is the Hessian matrix,
- α is the step size parameter.

Advantages:

- Rapid convergence near the solution.
- Uses second derivative information to adjust step direction.

Disadvantages:

- Requires computation and inversion of the Hessian.
- Convergence is not guaranteed if the initial guess is far from the solution or the Hessian is singular.

2 Implementation

The solution is implemented in Python. The following code snippet outlines the core function that performs the Newton-Raphson iterations:

```python
def newton_method(initial_guess, alpha, tol=1e-6, max_iter=1000):
    x1, y1 = initial_guess
    history = [(x1, y1)]

    for i in range(max_iter):
        # Compute gradient
        grad_x = 2 * np.cos(x1)
        grad_y = -3 * np.sin(y1)
        gradient = np.array([grad_x, grad_y])

        # Compute Hessian
        d2f_dx2 = -2 * np.sin(x1)
        d2f_dy2 = -3 * np.cos(y1)
        hessian = np.array([[d2f_dx2, 0],
                            [0, d2f_dy2]])

        if np.linalg.norm(gradient) < tol:
            return History((x1, y1), i, history, initial_guess, alpha)

        try:
            delta = -alpha * np.dot(np.linalg.inv(hessian), gradient)
        except np.linalg.LinAlgError:
            delta = -alpha * gradient

        x1 += delta[0]
        y1 += delta[1]
        history.append((x1, y1))

    return History((x1, y1), max_iter, history, initial_guess, alpha)
```

Listing 1: Newton-Raphson Implementation Using Hessian

The code includes:

- Calculation of the gradient and Hessian matrix for the function $f(x, y)$.
- A try-except block to handle cases when the Hessian matrix is singular, in which case the algorithm falls back to gradient descent.

- A history log for tracking the iterative steps, which is later used for visualizations.

Visualizations are generated using Matplotlib. The function `visualize()` plots the convergence paths on a 3D surface, while `visualize_2d()` creates 2D contour plots that help in comparing the effect of different initial guesses and step sizes.

3 Discussion

The experiments performed in the project were designed to explore how different initial vectors and step size parameters affect convergence.

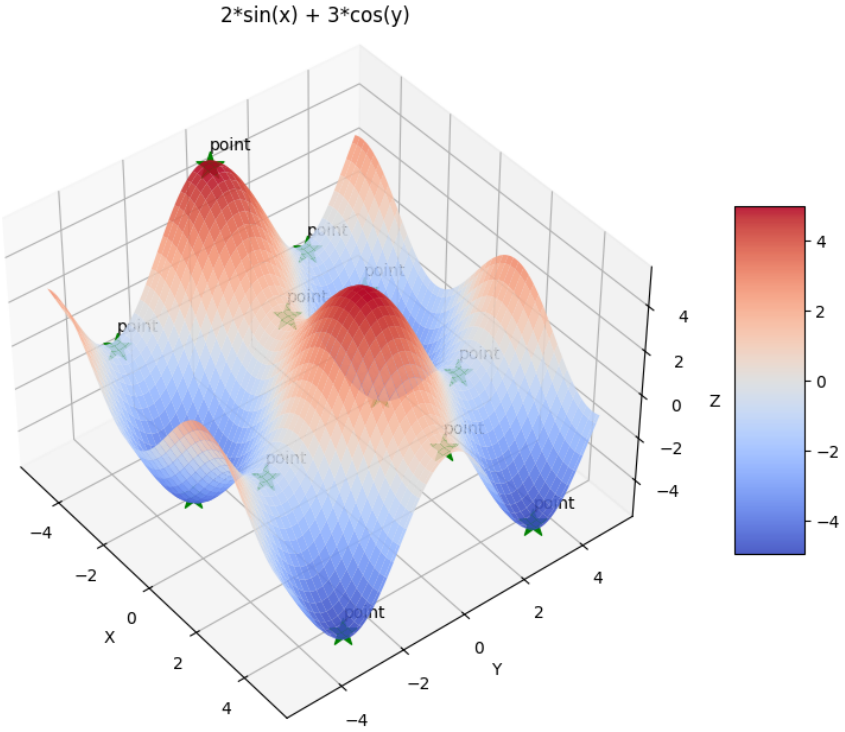

Figure 1: All the points found by Newton Method for 2*sinx + 3*cosy for x ∈ [-5,5] and y ∈ [-5,5]

3.1 Impact of Different Initial Vectors

In one set of experiments, several initial guesses were used (e.g., $[-2.0, 3.0]$, $[0.0, 2.0]$, $[3.0, 1.0]$, $[1.0, -2.0]$) with a fixed step size. The following observations were made:

- **Convergence to the same critical point:** Despite the varied starting points, the algorithm converged to similar critical points. This demonstrates the method's robustness when the initial guess is within a basin of attraction.
- **Iteration count variability:** The number of iterations required for convergence varied with the choice of initial guess. Some starting points were closer to the optimum, leading to faster convergence.

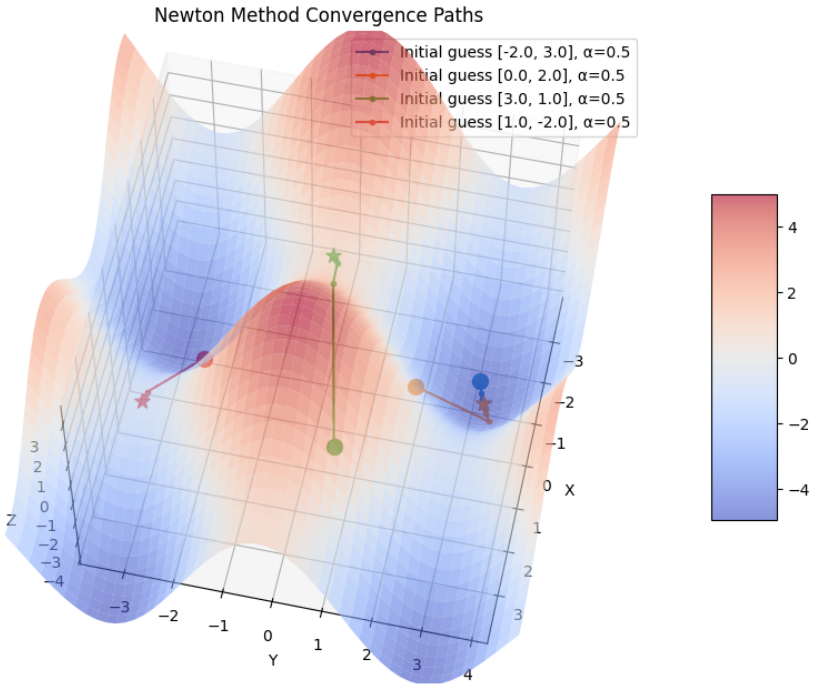

Figure 2: Newton Method steps for Different Initial Vectors

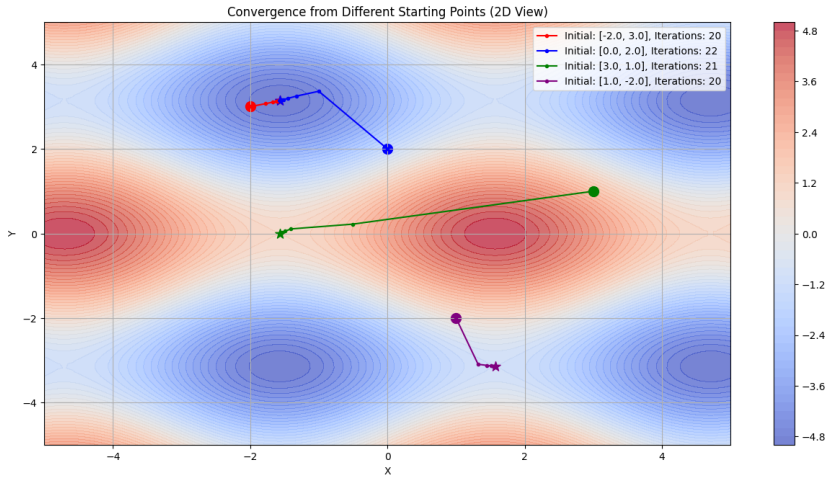

Figure 3: Newton Method steps for Different Initial Vectors, 2D

These observations were reinforced by the 2D and 3D visualization plots, which graphically illustrate the convergence paths from different starting points.

3.2 Impact of Different Step Size Parameters

Another experiment was conducted by fixing the initial guess (e.g., $[1.0, 1.0]$) and varying the step size parameter α (e.g., 0.1, 0.5, 1.0, 1.5):

- **Small Step Size ($\alpha = 0.1$):** Led to a slow convergence rate. The algorithm took more iterations to reach a solution.

- **Moderate Step Size ($\alpha = 0.5$ to 1.0):** Provided a balance between convergence speed and stability.
- **Large Step Size ($\alpha = 1.5$):** Risked overshooting the optimum, occasionally leading to oscillatory behavior or divergence in more delicate regions of the function.

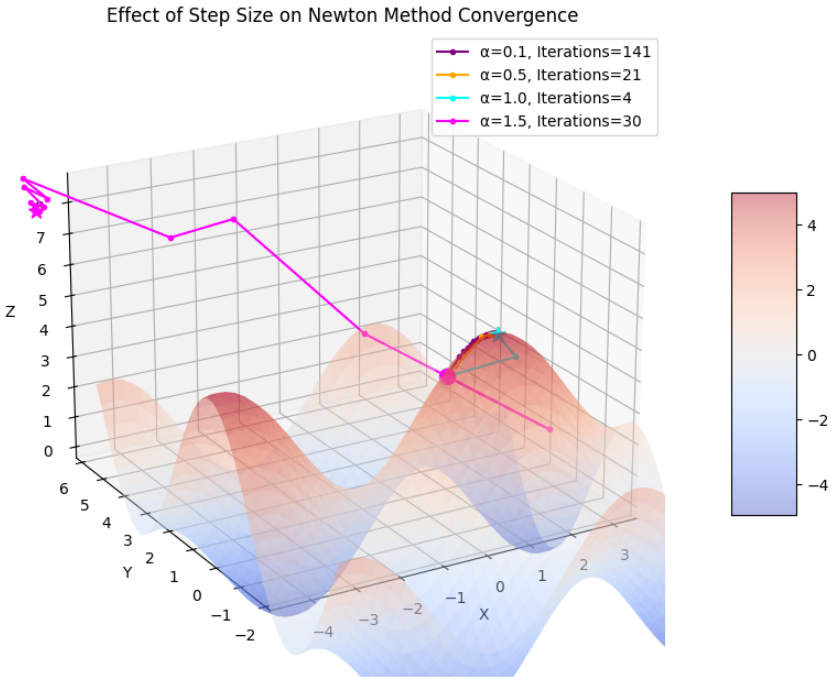

Figure 4: Newton Method steps for Different Step Size Parameters

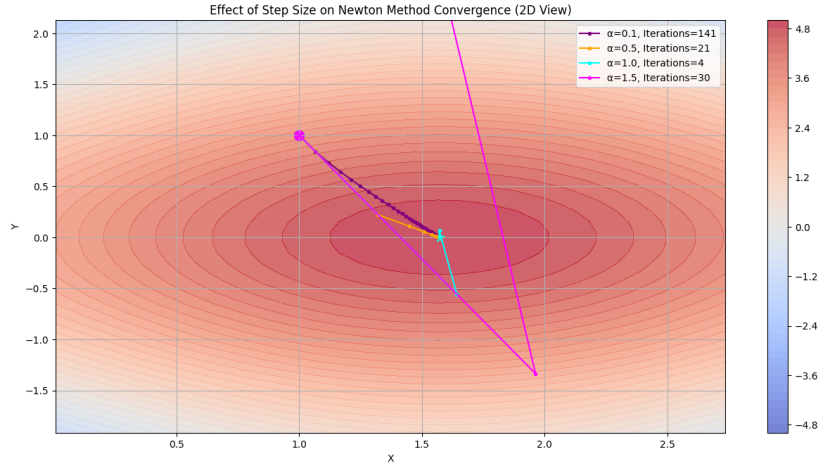

Figure 5: Newton Method steps for Different Step Size Parameters, 2D

The 2D contour plots clearly show how the paths differ with varying α, emphasizing that the choice of step size is crucial for achieving both convergence speed and stability.

3.3 Discussion of Special Cases

A set of experiments was also performed near points where the Hessian could become singular. These experiments used initial guesses close to the singularity region, and the algorithm occasionally switched to gradient descent. This fallback ensured continued progress even when second derivative information was unreliable.

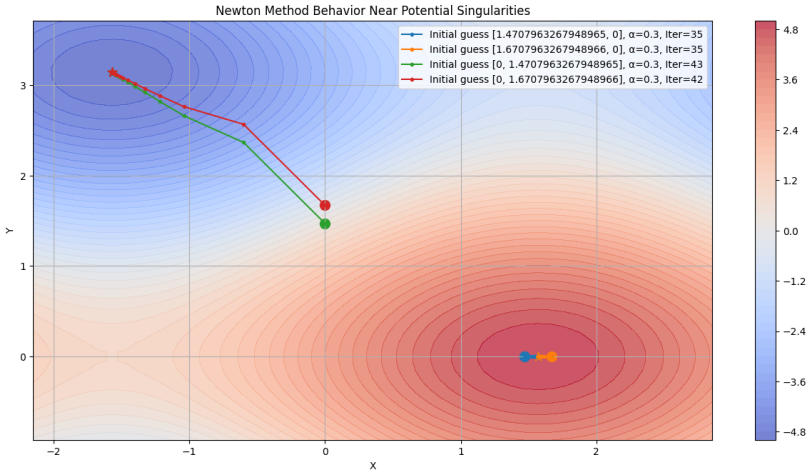

Figure 6: Newton Method near potential singularities

4 Conclusion

This project has demonstrated the application of the Newton-Raphson method in a three-dimensional space by leveraging the Hessian matrix to iteratively approach critical points of the function $f(x, y) = 2\sin(x) + 3\cos(y)$. The experiments highlighted two crucial aspects:

- **Sensitivity to Initial Vectors:** The choice of the starting point plays a pivotal role in determining the efficiency and success of the convergence. While well-chosen initial guesses rapidly converge to the desired solution, poorly selected starting points can lead to slow convergence or even divergence, especially in regions where the Hessian matrix is nearly singular.
- **Role of Step Size (α):** The step size parameter is equally critical, balancing the trade-off between convergence speed and stability. A smaller α promotes stable but slower convergence, whereas a larger α may accelerate the process at the risk of overshooting or instability. The experiments confirm that an optimal step size is necessary to harness the rapid local quadratic convergence of the method.

Lessons Learned:

1. The importance of tuning parameters (initial guess and step size) in iterative optimization methods.
2. The benefits of visualizing convergence paths to better understand algorithm behavior.
3. This optimization algorithm doesn't always minimize the function, but finds the convergent points. Thus stumbling upon another optimization problem.
4. Challenges include ensuring numerical stability and handling cases where the Hessian is nearly singular.