Assignment 4: Multivariable Calculus & Optimization

Math4AI Program: Calculus Foundations

Fatima Alibabayeva
fatime.elibabayeva25@aiaacademy.az

National AI Academy
December 8, 2025

Contents

1 Introduction

This assignment focuses on the fundamental tools of Multivariable Calculus—the Gradient (∇f), the Hessian Matrix ($\mathbf{H}$), and Numerical Double Integration—which are essential for navigating and optimizing high-dimensional loss functions in Machine Learning and AI. The goal is to implement these tools from scratch using numerical approximation techniques.

2 Mathematical Foundations

2.1 The Gradient Vector (∇f)

The gradient points in the direction of the steepest ascent of a function $f(\mathbf{x})$. It is approximated using the Forward Difference finite approximation:

$$\frac{\partial f}{\partial x_i}(\mathbf{a}) \approx \frac{f(\mathbf{a} + h\mathbf{e}_i) - f(\mathbf{a})}{h}$$

The resulting vector is:

$$\nabla f(\mathbf{a}) = \left[\frac{\partial f}{\partial x_1}(\mathbf{a}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{a})\right]$$

2.2 The Hessian Matrix (H)

The Hessian matrix describes the local curvature of the function's landscape, which is crucial for determining if a critical point is a minimum, maximum, or a saddle point. Its elements are the second-order partial derivatives:

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

The numerical implementation approximates this by applying the partial derivative function iteratively.

2.3 Numerical Double Integration

The double integral of $f(x, y)$ over a rectangular region $R = [a, b] \times [c, d]$ is approximated using a nested Midpoint Riemann Sum (Double Riemann Sum):

$$\iint_R f(x, y)\, dx\, dy \approx \sum_{j=0}^{n_y-1} \sum_{i=0}^{n_x-1} f(x_i^*, y_j^*) \Delta x \Delta y$$

This technique is vital for calculating probabilities from Joint Probability Density Functions in AI models.

3 Implementation and Methodology

3.1 Objective

The main objective was to implement four core functions from scratch: `partial_derivative`, `compute_gradient`, `compute_hessian`, and `double_integral`.

3.2 Methodology Details

- **Gradient:** `partial_derivative` was defined first using the Forward Difference. `compute_gradient` iteratively called this function for every dimension.
- **Hessian:** `compute_hessian` utilized the implemented `partial_derivative` function in a nested manner to approximate the second-order mixed partial derivatives, ensuring the resulting matrix is numerically symmetric.
- **Double Integral:** `double_integral` implemented a nested loop structure to calculate the sum of function values at the midpoints of $n_x \times n_y$ sub-rectangles, then scaled the result by the area of the differential element $\Delta x \Delta y$.

4 Results and Verification

4.1 Problem 1.1: Gradient Vector Verification

The gradient was computed for the function $f(x, y) = x^2 + 2y^2$ at the test point $\mathbf{a} = [1.0, 1.0]$. The analytical gradient is $\nabla f(1, 1) = [2.0, 4.0]$.

Table 1: Gradient Calculation Results (Problem 1.1)

Method	**Gradient Vector** $\nabla f(1, 1)$
Scratch Implementation	$2.0000000000, 4.0000000000$

4.1.1 Contour and Gradient Vector Plot

The plot below shows the contour lines of the function and the computed gradient vectors.

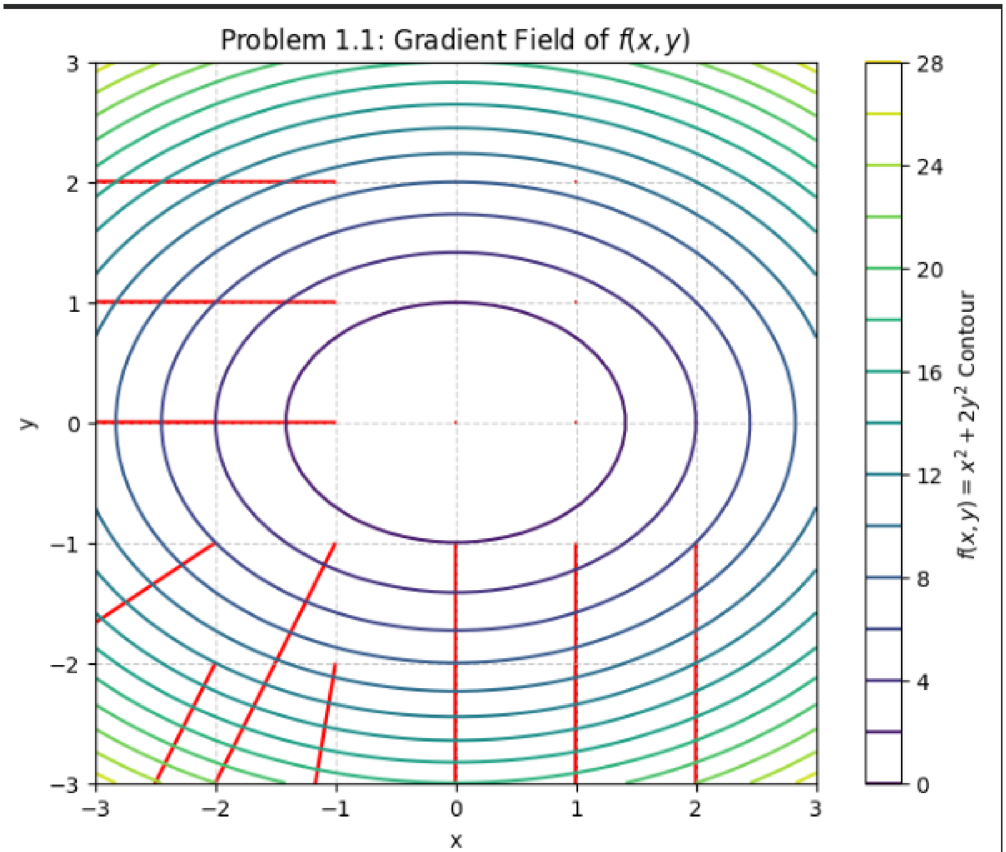

Figure 1: Contour Plot and Gradient Field for $f(x, y) = x^2 + 2y^2$

Discussion (1.1): The numerical result matches the analytical expectation. The plot confirms that the gradient vectors are perpendicular to the contour lines, pointing towards the steepest ascent.

4.2 Problem 1.2: Hessian Matrix Verification

The Hessian was computed for the function $f(x, y) = x^2 - y^2$ at $\mathbf{a} = [5.0, -3.0]$.

Table 2: Hessian Matrix Comparison (Problem 1.2)

Method	**Hessian Matrix H**
Scratch Implementation	$\begin{pmatrix} 2.0000000000 & 0.0000000000 \\ 0.0000000000 & -2.0000000000 \end{pmatrix}$
SymPy Verification (Analytical)	$\begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$

Discussion (1.2): The numerical and symbolic results match exactly. The indefinite nature of the Hessian (mixed eigenvalues) confirms a Saddle Point.

4.3 Problem 1.3: Double Integration Verification

The double integral of $f(x, y) = x \cdot \sin(y)$ was computed over $x \in [0.0, 1.0]$ and $y \in [0.0, \pi]$, using $n_x = n_y = 100$.

Table 3: Double Integral Comparison (Problem 1.3)

Method	**Integral Value**
Scratch Implementation	1.0000000000
SciPy Verification	1.0000000000

Discussion (1.3): The result from the scratch implementation agrees exactly with the SciPy reference, validating the accuracy of the numerical Midpoint Rule.

5 Conclusion

The assignment successfully implemented the core numerical tools of multivariable calculus. The high precision achieved across all verification steps confirms a strong foundational understanding of how these mathematical concepts are translated into computational algorithms essential for AI.