Metody Numeryczne - Zadanie XXIV

Mateusz Kamiński

January 2026

1 Wstęp

Celem zadania było znalezienie minimów funkcji dwóch zmiennych:

$$f(x, y) = 0.25x^4 + y^2 - 0.5x^2 + 0.125x + 0.0625(x - y)$$

co po uproszczeniu daje postać:

$$f(x, y) = 0.25x^4 - 0.5x^2 + 0.1875x + y^2 - 0.0625y$$

Obliczenia przeprowadzono dla 128 losowych punktów startowych z obszaru $[-3, 3] \times [-3, 3]$ przy użyciu algorytmu Levenberga-Marquardta.

2 Opis

Algorytm Levenberga-Marquardta stanowi hybrydę metody Newtona oraz metody najszybszego spadku. Kluczowym elementem jest modyfikacja macierzy Hesjanu poprzez wprowadzenie parametru $\lambda > 0$:

$$\tilde{H}_{ii} = (1 + \lambda)\frac{\partial^2 f}{\partial x_i^2}, \quad \tilde{H}_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} \;\; \text{dla} \;\; i \neq j$$

3 Implementacja

W programie wyznaczono analitycznie gradient oraz macierz Hesjanu:

$$\nabla f = \left[x^3 - x + 0.1875, 2y - 0.0625\right]^T$$

$$H = \begin{pmatrix} 3x^2 - 1 & 0 \\ 0 & 2 \end{pmatrix}$$

Zastosowano procedurę adaptacyjną parametru λ:

1. Jeśli $f(x_{\text{test}}) < f(x_k)$, krok jest akceptowany, a λ jest zmniejszane.
2. Jeśli $f(x_{\text{test}}) > f(x_k)$, krok jest odrzucany, a λ zwiększane.

```python
import numpy as np
from numpy.typing import NDArray

num = np.float64
vector = NDArray[np.float64]
matrix = NDArray[np.float64]

def f(x: vector):
    x, y = x
    return 0.25 * x**4 - 0.5 * x**2 + 0.1875 * x + y**2 - 0.0625 * y

def gradient(x: vector) -> vector:
    x, y = x
    return np.array([
        x**3 - x + 0.1875,
        2*y - 0.0625
    ], dtype=num)

def hessian(x: vector) -> matrix:
    x, y = x
    return np.array([
        [3*x**2 - 1, 0],
        [0, 2]
    ], dtype=num)

def levenberg_marquardt_step(x: vector, lam: num, grad: vector, hess: matrix) -> vector:
    hess_local = hess.copy()

    for i in range(len(x)):
        hess_local[i, i] *= (1 + lam)
```

```

    delta = np.linalg.solve(hess_local, grad)

    x_new = x - delta

    return x_new

def levenberg_marquardt(x: vector, lam: num, tolerance: float = 1e-10, max_iterations: int = 5000) -> vector:
    for k in range(max_iterations):
        grad = gradient(x)

        # warunek stopu potrzebny
        if np.linalg.norm(grad) < tolerance:
            break

        hess = hessian(x)
        f_x = f(x)

        while True:
            x_test: vector = levenberg_marquardt_step(x, lam, grad, hess)
            f_test = f(x_test)

            if f_test > f_x:
                # krok odrzucony
                lam *= 8
            else:
                # krok zaakceptowany
                x = x_test
                lam /= 8
                break

            if lam > 1e12:
                raise RuntimeError("Nie znaleziono minimum")

    return x, k + 1

def main():
```

```python
    np.random.seed(0)
    starts = np.random.uniform(-3, 3, size=(128, 2))

    results = []

    for x_start in starts:
        try:
            x_min, iterations = levenberg_marquardt(x_start, 1/1024)
            f_val = f(x_min)
            results.append({
                "start": x_start,
                "x_min": x_min,
                "f_val": f_val,
                "iterations": iterations,
                "success": True
            })
        except RuntimeError:
            continue

    results.sort(key=lambda r: r["f_val"])

    unique_solutions = []
    if results:
        unique_solutions.append(results[0])

        for r in results[1:]:
            is_new = True
            for u in unique_solutions:
                if np.allclose(r["x_min"], u["x_min"], atol=1e-5):
                    is_new = False
                    break
            if is_new:
                unique_solutions.append(r)

    # 3. Print Summary
    print(f"{'Rank':<5} | {'x_min':<30} | {'f(x)':<12} | {'Count'}")
    print("-" * 65)

    for i, sol in enumerate(unique_solutions):
```

```python
        # Count how many starting points led to this specific minimum
        count = sum(1 for r in results if np.allclose(r["x_min"], sol["x_min"], atol=1e-5))

        x_str = f"[{sol['x_min'][0]:.4f}, {sol['x_min'][1]:.4f}]"
        print(f"{i+1:<5} | {x_str:<30} | {sol['f_val']:<12.6f} | {count}")

    print(f"\nUdanych poszukiwań: {len(results)} / {len(starts)}")

if __name__ == "__main__":
    main()
```

4 Wyniki i analiza

4.1 Znalezione minima

Po przetestowaniu 128 punktów startowych, algorytm zbiegał do jednego z dwóch minimów lokalnych lub punktu siodłowego.

Typ punktu	Punkt (x, y)	Wartość $f(x, y)$	Liczba trafień
Minimum	$(-1.0831, 0.0312)$	-0.446566	51
Minimum	$(0.8882, 0.0312)$	-0.073298	55
Siodło	$(0.1949, 0.0312)$	0.016935	4