Metody Numeryczne - Zadanie XXII

Mateusz Kamiński

December 2025

1 Wstęp

Celem zadania było numeryczne znalezienie minimum funkcji Rosenbrocka:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

z wykorzystaniem algorytmu Levenberga-Marquardta (LM).

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. Zastosowano procedurę sprawdzania wartości funkcji po każdym kroku:

1. Jeśli $f(x_{\text{test}}) < f(x_k)$, krok jest akceptowany, a λ zmniejszane (dzielone przez 8), aby przybliżyć metodę do Newtonowskiej.

2. Jeśli $f(x_{\text{test}}) > f(x_k)$, krok jest odrzucany, a λ zwiększane (mnożone przez 8), co „skręca” kierunek poszukiwań w stronę gradientu.

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

```python
import matplotlib.pyplot as plt

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

def rosenbrock(x: num, y: num) -> num:
    return (1 - x)**2 + 100*(y - x**2)**2

def gradient(x: num, y: num) -> vector:
    df_dx = -2*(1 - x) - 400*(y - x**2)*x
    df_dy = 200*(y - x**2)
    return np.array([df_dx, df_dy], dtype=num)

def hessian(x: num, y: num) -> matrix:
    h11 = 2 - 400*x*(y - x**2) + 800*x**2
    h12 = -400*x
    h22 = 200
    return np.array([[h11, h12], [h12, h22]], dtype=num)

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

    for i in range(2):
        hess_local[i, i] *= (1 + lam)

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

    x_new = x - delta[0]
    y_new = y - delta[1]

    return x_new, y_new

def levenberg_marquardt(x: num, y: num, lam: num, tolerance: float =
1e-10, max_iterations: int = 5000) -> tuple[num, num, vector]:
    path = [(x, y)]

    for k in range(max_iterations):
```

```
        grad = gradient(x, y)

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

        hess = hessian(x, y)
        f = rosenbrock(x, y)

        while True:
            x_test, y_test = levenberg_marquardt_step(x, y, lam, grad, hess)
            f_test = rosenbrock(x_test, y_test)

            if f_test > f:
                # krok odrzucony
                lam *= 8
            else:
                # krok zaakceptowany
                x = x_test
                y = y_test
                path.append((x, y))
                lam /= 8
                break

            if lam > 1e12:
                break

    return x, y, np.array(path)

def plot_all_paths(all_paths: list[vector]):
    x = np.linspace(0, 2, 400)
    y = np.linspace(-1, 4, 400)
    X, Y = np.meshgrid(x, y)
    Z = rosenbrock(X, Y)

    plt.figure(figsize=(6, 5))
    plt.contour(X, Y, Z, levels=100, cmap="grey")

```

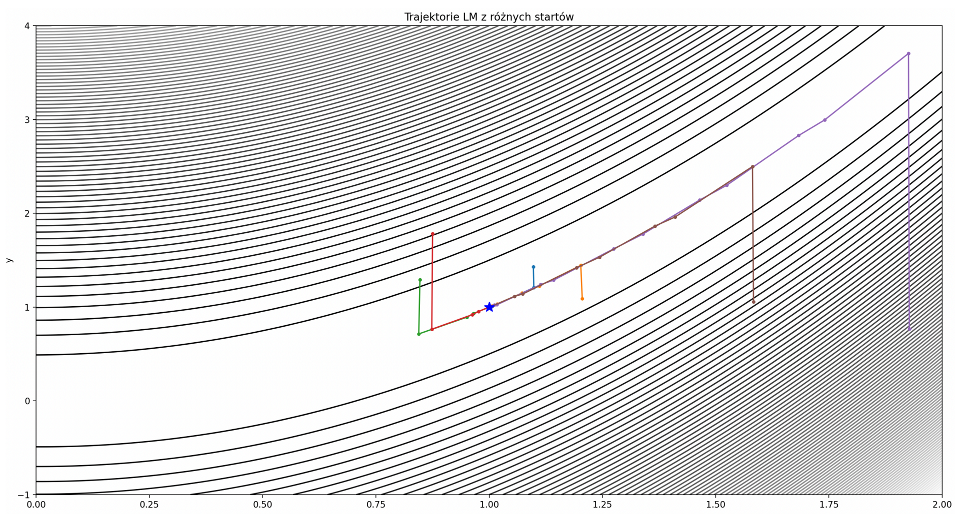
```python
    # Rysujemy wszystkie trajektorie
    for path in all_paths:
        plt.plot(path[:, 0], path[:, 1], "-o", markersize=3)

    # Minimum
    plt.plot(1, 1, "b*", markersize=12)

    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("Trajektorie LM z różnych startów")
    plt.tight_layout()
    plt.show()


def main():
    np.random.seed(0)
    all_paths = []

    for i in range(6):
        x, y = np.random.uniform(0, 2, size=2)

        x_min, y_min, path = levenberg_marquardt(x, y, 1/1024)
        all_paths.append(path)

        print(f"Start {i + 1}: ({x:.3f}, {y:.3f})")
        print(f"  Koniec: ({x_min:.6f}, {y_min:.6f})")
        print(f"  Kroki: {len(path)}")
        print()

    plot_all_paths(all_paths)

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

4 Wyniki i analiza

4.1 Trajektorie poszukiwań

Wylosowano 6 punktów startowych z rozkładu X, Y $\sim$ U(0, 2). Algorytm we wszystkich przypadkach zbiegł do globalnego minimum $(1.000000, 1.000000)$ z wysoką precyzją. Poniższa tabela przedstawia liczbę iteracji potrzebną do osiągnięcia zbieżności:

Punkt startowy (x_0, y_0)	**Punkt końcowy**	**Liczba kroków**
$(1.098, 1.430)$	$(1.000, 1.000)$	7
$(1.206, 1.090)$	$(1.000, 1.000)$	10
$(0.847, 1.292)$	$(1.000, 1.000)$	8
$(0.875, 1.784)$	$(1.000, 1.000)$	8
$(1.927, 0.767)$	$(1.000, 1.000)$	16
$(1.583, 1.058)$	$(1.000, 1.000)$	13



Rysunek 1: Trajektorie algorytmu LM dla różnych punktów startowych na tle poziomic funkcji Rosenbrocka.