Metody Numeryczne - Zadanie XXIII

Mateusz Kamiński

December 2025

1 Wstęp

Celem zadania było numeryczne znalezienie minimum czterowymiarowej funkcji Rosenbrocka:

$$f(x_1, x_2, x_3, x_4) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 + 100(x_3 - x_2^2)^2 + 100(x_4 - x_3^2)^2$$

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 $\lambda > 0$:

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

3 Implementacja

W programie wyznaczono analitycznie gradient oraz macierz Hesjanu dla przypadku wielowymiarowego. 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

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

def rosenbrock(x: vector) -> num:
    f = (1 - x[0])**2

    for i in range(len(x) - 1):
        f += 100*(x[i + 1] - x[i]**2)**2

    return f

def gradient(x: vector) -> vector:
    n = len(x)
    grad = np.zeros(n, dtype=num)

    grad[0] = -2 * (1 - x[0]) - 400 * x[0] * (x[1] - x[0] ** 2)

    for i in range(1, n - 1):
        grad[i] = 200 * (x[i] - x[i - 1] ** 2) - 400 * x[i] * (x[i + 1] - x[i] ** 2)

    grad[-1] = 200 * (x[-1] - x[-2] ** 2)

    return grad

def hessian(x: vector) -> matrix:
    n = len(x)
    H = np.zeros((n, n), dtype=num)

    for i in range(n - 1):
        H[i, i] += 800 * x[i] ** 2 - 400 * (x[i + 1] - x[i] ** 2)
        H[i, i + 1] = -400 * x[i]
        H[i + 1, i] = -400 * x[i]
        H[i + 1, i + 1] += 200

```

```
39      H[0, 0] += 2
40
41      return H
42
43  def levenberg_marquardt_step(x: vector, lam: num, grad: vector, hess:
    matrix) -> vector:
44      hess_local = hess.copy()
45
46      for i in range(len(x)):
47          hess_local[i, i] += lam
48
49      delta = np.linalg.solve(hess_local, grad)
50
51      x_new = x - delta
52
53      return x_new
54
55  def levenberg_marquardt(x: vector, lam: num, tolerance: float =
    1e-10, max_iterations: int = 5000) -> vector:
56      for k in range(max_iterations):
57          grad = gradient(x)
58
59          # warunek stopu potrzebny
60          if np.linalg.norm(grad) < tolerance:
61              break
62
63          hess = hessian(x)
64          f = rosenbrock(x)
65
66          while True:
67              x_test: vector = levenberg_marquardt_step(x, lam, grad,
                hess)
68              f_test = rosenbrock(x_test)
69
70              if f_test > f:
71                  # krok odrzucony
72                  lam *= 8
73              else:
74                  # krok zaakceptowany
```

```
75                 x = x_test
76                 lam /= 8
77                 break
78
79             if lam > 1e12:
80                 break
81
82     return x, k + 1
83
84 def main():
85     np.random.seed(0)
86
87     for i in range(6):
88         x = np.random.uniform(-2, 2, size=4)
89
90         x_min, iterations = levenberg_marquardt(x, 1/1024)
91
92         print(f"Start {i + 1}: {x}")
93         print(f"Koniec: {x_min}")
94         print(f"Iteracje: {iterations}")
95         print()
96
97
98 if __name__ == "__main__":
99     main()
```

4 Wyniki i analiza

4.1 Zbieżność algorytmu

Wylosowano 6 punktów startowych z rozkładu jednorodnego w zakresie $[-2, 2]$. Algorytm we wszystkich przypadkach zbiegł do globalnego minimum $(1.0, 1.0, 1.0, 1.0)$. Poniższa tabela przedstawia wyniki dla poszczególnych startów:

Punkt startowy (x_0)	**Punkt końcowy**	**Liczba kroków**
$(0.20, 0.86, 0.41, 0.18)$	$(1.0, 1.0, 1.0, 1.0)$	34
$(-0.31, 0.58, -0.25, 1.57)$	$(1.0, 1.0, 1.0, 1.0)$	35
$(1.85, -0.47, 1.17, 0.12)$	$(1.0, 1.0, 1.0, 1.0)$	33

$(0.27, 1.70, -1.72, -1.65)$	$(1.0, 1.0, 1.0, 1.0)$	38
$(-1.92, 1.33, 1.11, 1.48)$	$(1.0, 1.0, 1.0, 1.0)$	41
$(1.91, 1.20, -0.15, 1.12)$	$(1.0, 1.0, 1.0, 1.0)$	27