

# Zadanie Numeryczne 18

Mateusz Wojtyna

## 1. Wstęp

Należało znaleźć minimum funkcji będącej wielomianem interpolacyjnym z zadania 7:

$$f(x) = x^7 - 2x^5 + 3x^4 - x + 1$$

Wykorzystano metodę Brenta.

## 2. Opis

Na początku ograniczamy minimum funkcji trzema punktami  $a < b < c$  tak, że  $f(a) > f(b)$  oraz  $f(b) < f(c)$ . Następnie wybieramy punkt  $d$  na podstawie jakiejś metody i odpowiednio przesuwamy punkty  $a, b, c$ . Ustalamy znowu punkt  $d$  biorąc pod uwagę nowe  $a, b, c$ , i tak dalej. Iterację zatrzymujemy jeżeli  $|c - a| < \varepsilon(|b| + |d|)$ .

### 2.1. Metoda Brenta

Metoda Brenta wykorzystuje interpolację paraboliczną, aby znaleźć minimum. Przez punkty  $(a, f(a)), (b, f(b)), (c, f(c))$  przeprowadzamy parabolę, a jako punkt  $d$  bierzemy jej minimum:

$$d = \frac{1}{2} \cdot \frac{a^2(f(c) - f(b)) + b^2(f(a) - f(c)) + c^2(f(b) - f(a))}{a(f(c) - f(b)) + b(f(a) - f(c)) + c(f(b) - f(a))}$$

Jeżeli nie zachodzi  $a < d < c$  to ustawiamy  $d = \frac{c+a}{2}$ . Metoda ta jest zbieżna szybciej niż liniowo, ale wolniej niż kwadratowo.

### 3. Kod

Program napisano w Pythonie z użyciem pakietu *NumPy*.

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

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

def f(x: float):
    coeffs = [1, -1, 0, 0, 3, -2, 0, 1]
    p = coeffs[-1]

    for k in range(len(coeffs) - 2, -1, -1):
        p = p * x + coeffs[k]

    return p

def initial_interval(x_0: num, delta=0.1, limit=100):
    a = x_0
    fa = f(a)
    b = a + delta
    fb = f(b)

    if fb > fa:
        # zmiana kierunku
        delta *= -1
        b = a + delta
        fb = f(b)
        assert fa > fb

    for _ in range(limit):
        c = b + delta
        fc = f(c)

        if fc > fb:
            return a, b, c

        a = b
        fa = fb
        b = c
        fb = fc
        delta *= 2
```

```

raise RuntimeError("Nie znaleziono przedziału")

def brent(a: num, b: num, c: num, limit=100, eps=1e-6):
    prev = abs(c - a)
    prev2 = prev

    for i in range(limit):
        fa = f(a)
        fb = f(b)
        fc = f(c)
        old_b = b
        d = 0

        accept = False
        denominator = a * (fc - fb) + b * (fa - fc) + c * (fb - fa)
        if abs(denominator) > 1e-16:
            nominator = a * a * (fc - fb) + b * b * (fa - fc) + c * c * (fb - fa)
            d = nominator / (2 * denominator)

        if a < d < c and max(abs(d - a), (c - d)) < 0.5 * prev2:
            accept = True

        if not accept:
            d = (c + a) / 2

        if abs(c - a) < eps * (abs(old_b) + abs(d)):
            return d, i + 1

        fd = f(d)
        if fd < fb:
            if d < b:
                c = b
                b = d
            else:
                a = b
                b = d
        else:
            if d < b:
                a = d
            else:
                c = d

    prev2 = prev
    prev = abs(c - a)

```

```
raise RuntimeError("Nie znaleziono punktu (brent)")

def main():
    a, b, c = initial_interval(0)
    print("Punkty: ", a, b, c)
    value, steps = brent(a, b, c)
    print(f"{value} po {steps} krokach")

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

## 4. Wyniki ( $\varepsilon = 10^{-6}$ )

Dla punktów  $a = 0.2$ ,  $b = 0.4$ ,  $c = 0.8$ :

$$x_{\min} \approx 0.502926, \text{ iteracje} = 17$$