Optymalizacja
Sprawozdanie z laboratorium 11

Jakub Grześ

14.06.2024

1 Treść zadań

Zadanie 1

Wyznacz punkty krytyczne każdej z poniższych funkcji. Scharakteryzuj każdy znaleziony punkt jako minimum, maksimum lub punkt siodłowy. Dla każdej funkcji zbadaj, czy posiada minimum globalne lub maksimum globalne na zbiorze $\mathbb{R}^2$.

$$f_1(x, y) = x^2 - 4xy + y^2 \tag{1}$$
$$f_2(x, y) = x^4 - 4xy + y^4 \tag{2}$$
$$f_3(x, y) = 2x^3 - 3x^2 - 6xy(x - y - 1) \tag{3}$$
$$f_4(x, y) = (x - y)^4 + x^2 - y^2 - 2x + 2y + 1 \tag{4}$$

Zadanie 2

Należy wyznaczyć najkrótszą ścieżkę robota pomiędzy dwoma punktami $x(0)$ i $x(n)$. Problemem są przeszkody usytuowane na trasie robota, których należy unikać. Zadanie polega na minimalizacji funkcji kosztu, która sprowadza problem nieliniowej optymalizacji z ograniczeniami do problemu nieograniczonej optymalizacji.

Macierz $X \in \mathbb{R}^{(n+1)\times 2}$ opisuje ścieżkę złożoną z $n+1$ punktów $x(0), x(1), x(2), \ldots, x(n)$. Każdy punkt posiada 2 współrzędne, $x(i) \in \mathbb{R}^2$. Punkty początkowy i końcowy ścieżki, $x(0)$ i $x(n)$, są ustalone.

Punkty z przeszkodami (punkty o 2 współrzędnych), $r(i)$, dane są w macierzy przeszkód $R \in \mathbb{R}^{k\times 2}$.

W celu optymalizacji ścieżki robota należy użyć metody największego spadku. Funkcja celu użyta do optymalizacji $F(x(0), x(1), \ldots, x(n))$ zdefiniowana jest jako:

$$F(x(0), x(1), \ldots, x(n)) = \lambda_1 \sum_{i=0}^{n} \sum_{j=1}^{k} \frac{1}{\epsilon + \|x(i) - r(j)\|_2^2} + \lambda_2 \sum_{i=0}^{n-1} \|x(i+1) - x(i)\|_2^2 \tag{5}$$

Symbole użyte we wzorze mają następujące znaczenie:

- Stałe λ_1 i λ_2 określają wpływ każdego członu wyrażenia na wartość $F(X)$.
  - λ_1 określa wagę składnika zapobiegającego zbytniemu zbliżaniu się do przeszkody
  - λ_2 określa wagę składnika zapobiegającego tworzeniu bardzo długich ścieżek
- n jest liczbą odcinków, a $n+1$ liczbą punktów na trasie robota.
- k jest liczbą przeszkód, których robot musi unikać.
- Dodanie ϵ w mianowniku zapobiega dzieleniu przez zero.

1. Wyprowadzenie gradientu

Wyprowadź wyrażenie na gradient ∇F funkcji celu F względem $x(i)$:

$$\nabla F = \left[\frac{\partial F}{\partial x(0)}, \ldots, \frac{\partial F}{\partial x(n)} \right] \tag{6}$$

Wzór wyraź poprzez wektory $x(i)$ i ich składowe, wektory $r(j)$ i ich składowe, ϵ, λ_1, λ_2, n i k (niekoniecznie wszystkie).

Wskazówka: $\frac{\partial \|x\|^2}{\partial x} = 2x$.

2. Algorytm największego spadku

Opisz matematycznie i zaimplementuj kroki algorytmu największego spadku z przeszukiwaniem liniowym, który służy do minimalizacji funkcji celu F. Do przeszukiwania liniowego (ang. *line search*) użyj metody złotego podziału (ang. *golden section search*). W tym celu załóż, że F jest unimodalna (w rzeczywistości tak nie jest) i że można ustalić początkowy przedział, w którym znajduje się minimum.

3. Znalezienie najkrótszej ścieżki

Znajdź najkrótszą ścieżkę robota przy użyciu algorytmu zaimplementowanego w poprzednim punkcie. Przyjmij następujące wartości parametrów:

- $n = 20$, $k = 50$
- $x(0) = [0, 0]$, $x(n) = [20, 20]$
- $r(i) \sim U(0, 20) \times U(0, 20)$
- $\lambda_1 = \lambda_2 = 1$
- $\epsilon = 10^{-13}$
- liczba iteracji $= 400$

Ponieważ nie chcemy zmieniać położenia punktu początkowego i końcowego, $x(0)$, $x(n)$, wyzeruj gradient funkcji F względem tych punktów.

Obliczenia przeprowadź dla 5 różnych losowych inicjalizacji punktów wewnątrz ścieżki $x(1), \ldots, x(n-1)$.

Narysuj przykładowy wykres wartości funkcji F w zależności od iteracji.

2 Rozwiązania zadań

2.1 Zadanie 1

2.1.1 Znalezienie minimów, maksimów lokalnych oraz punktów siodłowych

W implementacji korzystano z metod modułu `SymPy`.
Punkty krytyczne funkcji $f(x, y)$ to punkty, w których gradient funkcji jest równy zeru. Gradient funkcji f jest wektorem składającym się z pierwszych pochodnych cząstkowych funkcji f względem x i y:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

Aby znaleźć punkty krytyczne, rozwiązujemy układ równań:

$$\begin{cases} \frac{\partial f}{\partial x} = 0 \\ \frac{\partial f}{\partial y} = 0 \end{cases}$$

Dla każdej funkcji znajdziemy takie punkty oraz rozwiążemy układ równań. Zaimplementowano w tym celu funkcję `find_critical_points`. Wybiera ona tylko rzeczywiste rozwiązania powyższego układu równań.

```
def find_critical_points(f):
    f_x = sp.diff(f, x)
    f_y = sp.diff(f, y)

    critical_points = sp.solve([f_x, f_y], (x, y))

    if isinstance(critical_points, dict):
        critical_points = [tuple(critical_points.values())]

    return [point for point in critical_points if all(sp.im(part) == 0 for part in point)]
```

Te punkty spełniają warunek konieczny, ale niekoniecznie wystarczający. Sklasyfikować je można za pomocą hesjanu funkcji. Hesjan funkcji f jest macierzą drugich pochodnych cząstkowych funkcji f i jest zdefiniowany jako:

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}$$

Aby scharakteryzować znalezione punkty krytyczne, należy zbadać wartości własne hesjanu w tych punktach:

- Jeśli wszystkie wartości własne są dodatnie, punkt krytyczny jest lokalnym minimum.
- Jeśli wszystkie wartości własne są ujemne, punkt krytyczny jest lokalnym maksimum.
- Jeśli wartości własne mają różne znaki, punkt krytyczny jest punktem siodłowym.

Powyższe kroki wykonuje funkcja `classify_critical_points`.

```
def classify_critical_points(f):
    critical_points = find_critical_points(f)

    local_min = []
    local_max = []
    saddle = []

    hessian_matrix = sp.hessian(f, (x, y))

    for point in critical_points:
        hessian_at_point = hessian_matrix.subs({x: point[0], y: point[1]})
        eigenvalues = hessian_at_point.eigenvals()

        if all(sp.im(eigenvalue) == 0 for eigenvalue in eigenvalues):
            real_eigenvalues = [eigenvalue.evalf() for eigenvalue in eigenvalues]
            if all(eigenvalue > 0 for eigenvalue in real_eigenvalues):
                local_min.append(point)
            elif all(eigenvalue < 0 for eigenvalue in real_eigenvalues):
                local_max.append(point)
            else:
                saddle.append(point)

    return local_min, local_max, saddle
```

Na każdej funkcji wykonano odpowiednio obie metody. Wyniki zaprezentowano w poniższej tabeli.

Funkcja	**Minima Lokalna**	**Maksima Lokalna**	**Punkty Siodłowe**
f_1	Brak	Brak	$(0,0)$
f_2	$(-1,-1),(1,1)$	Brak	$(0,0)$
f_3	$(1,0)$	$(-1,-1)$	$(0,-1),(0,0)$
f_4	Brak	Brak	$(1,1)$

Tabela 1: Punkty krytyczne funkcji

2.1.2 Zbadanie istnienia maksimów i minimów globalnych

Maksimum lub minimum globalne może wystąpić w jednym z punktów krytycznych, lub na granicy dziedziny. Funkcja `check_global_extrema` sprawdza wartości funkcji w punktach krytycznych i jej granicach. Następnie wybiera najmniejszą i największą z tych wartości. Jeśli występuje gdzieś nieskończoność, funkcja nie ma ekstremum globalnego.

```
def check_global_extrema(f):
    critical_points = find_critical_points(f)
    values_at_critical_points = [f.subs({x: point[0], y: point[1]})
    for point in critical_points]

    limits = [sp.limit(f, var, lim) for var, lim in [(x, sp.oo), (x, -sp.oo),
    (y, sp.oo), (y, -sp.oo)]]

    all_values = values_at_critical_points + limits

    numerical_values = [val for val in all_values if val.is_real
    and not val.has(sp.oo, -sp.oo)]

    if numerical_values:
        global_min = min(numerical_values)
        global_max = max(numerical_values)
    else:
        global_min, global_max = None, None

    if any(val == sp.oo or val == -sp.oo for val in limits):
        if all(val == sp.oo for val in limits):
            global_min = None  # No global minimum if all limits tend to +
        elif all(val == -sp.oo for val in limits):
            global_max = None  # No global maximum if all limits tend to -

    return global_min, global_max
```

Powyższą metodę wykonano dla każdej z badanych funkcji. Wyniki przedstawiono w poniższej tabeli.

Funkcja	**Globalne minimum**	**Globalne maksimum**
f_1	Nie istnieje	0
f_2	Nie istnieje	0
f_3	-1	1
f_4	Nie istnieje	1

Tabela 2: Wyniki ekstremów globalnych dla każdej funkcji

2.2 Zadanie 2

2.3 Wyprowadzenie gradientu funkcji celu F

Aby rozwiązać zadanie musimy wyznaczyć funkcję celu, którą będziemy minimalizować. W naszym przypadku będzie to:

$$F(x^{(0)}, x^{(1)}, \ldots, x^{(n)}) = \lambda_1 \sum_{j=1}^{k} \frac{1}{\epsilon + \|x^{(i)} - r^{(j)}\|^2} + \lambda_2 \sum_{i=0}^{n-1} \|x^{(i+1)} - x^{(i)}\|^2$$

gdzie:

- $x^{(i)}$ to pozycja robota w punkcie i,
- $r^{(j)}$ to pozycja przeszkody j,
- λ_1 i λ_2 to wagi poszczególnych składników funkcji celu,
- ϵ to mała stała zapobiegająca dzieleniu przez zero.

Pierwszy człon penalizuje zbliżanie się do przeszkód a drugi tworzenie długich ścieżek. Minimalizacja funkcji celu pozwoli na utworzenie możliwie krótkiej ścieżki unikając przeszkód.

Strategia jej minimalizacji wymaga wyznaczenia gradientu funkcji kosztu. Gradient ∇F funkcji celu F względem $x^{(i)}$ można wyrazić jako:

$$\nabla F = \left[\frac{\partial F}{\partial x^{(0)}}, \frac{\partial F}{\partial x^{(1)}}, \ldots, \frac{\partial F}{\partial x^{(n)}} \right]$$

1. **Pochodna pierwszego członu względem i-tej zmiennej**:

$$\frac{\partial}{\partial x^{(i)}} \left(\sum_{j=1}^{k} \frac{1}{\epsilon + \|x^{(i)} - r^{(j)}\|^2} \right) = \sum_{j=1}^{k} \frac{\partial}{\partial x^{(i)}} \left(\frac{1}{\epsilon + \|x^{(i)} - r^{(j)}\|^2} \right)$$

Używając reguły łańcuchowej:

$$\frac{\partial}{\partial x^{(i)}} \left(\frac{1}{\epsilon + \|x^{(i)} - r^{(j)}\|^2} \right) = \frac{-1}{(\epsilon + \|x^{(i)} - r^{(j)}\|^2)^2} \cdot \frac{\partial}{\partial x^{(i)}} \left(\|x^{(i)} - r^{(j)}\|^2 \right)$$

Wiadomo, że:

$$\frac{\partial}{\partial x} \|x\|^2 = 2x$$

Stąd:

$$\frac{\partial}{\partial x^{(i)}} \left(\|x^{(i)} - r^{(j)}\|^2 \right) = 2(x^{(i)} - r^{(j)})$$

Ostatecznie:

$$\frac{\partial}{\partial x^{(i)}} \left(\frac{1}{\epsilon + \|x^{(i)} - r^{(j)}\|^2} \right) = \frac{-2(x^{(i)} - r^{(j)})}{(\epsilon + \|x^{(i)} - r^{(j)}\|^2)^2}$$

2. **Pochodna drugiego członu względem i-tej zmiennej**:

$$\frac{\partial}{\partial x^{(i)}} \left(\sum_{i=0}^{n-1} \|x^{(i+1)} - x^{(i)}\|^2 \right) = \sum_{i=0}^{n-1} \frac{\partial}{\partial x^{(i)}} \left(\|x^{(i+1)} - x^{(i)}\|^2 \right) =$$

$$= \frac{\partial}{\partial x^{(i)}} \left(\|x^{(i)} - x^{(i-1)}\|^2 \right) + \frac{\partial}{\partial x^{(i)}} \left(\|x^{(i+1)} - x^{(i)}\|^2 \right)$$

Co daje:

$$2(x^{(i)} - x^{(i-1)}) + 2(x^{(i)} - x^{(i+1)})$$

Ostateczne wyrażenie na gradient:

$$\nabla F = \lambda_1 \sum_{j=1}^{k} \frac{-2(x^{(i)} - r^{(j)})}{(\epsilon + \|x^{(i)} - r^{(j)}\|^2)^2} + \lambda_2 \left(2(x^{(i)} - x^{(i-1)}) + 2(x^{(i)} - x^{(i+1)}) \right)$$

2.4 Strategia rozwiązania

Aby znaleźć optymalną ścieżkę, stosujemy algorytm największego spadku. Algorytm ten polega na iteracyjnej aktualizacji pozycji punktów $x^{(i)}$ w kierunku przeciwnym do gradientu, aż do osiągnięcia minimalnej wartości funkcji celu.

Przeszukiwanie liniowe (Line Search)

Do określenia optymalnej wartości kroku α używamy przeszukiwania liniowego metodą złotego podziału. Wartość α jest wybierana w taki sposób, aby po wykonaniu kroku, wartość funkcji celu była możliwie najmniejsza.

- Przypuśćmy, że funkcja f jest unimodalna na przedziale $[a, b]$, i niech x_1 oraz x_2 będą dwoma punktami wewnątrz $[a, b]$, gdzie $x_1 < x_2$.
- Porównując wartości $f(x_1)$ i $f(x_2)$, możemy odrzucić albo $[a, x_1]$, albo $[x_2, b]$, wiedząc, że minimum znajduje się w pozostałym podprzedziale.
- Aby powtórzyć proces, musimy obliczyć tylko jedną nową wartość funkcji.
- Aby zmniejszyć długość przedziału o stałą część przy każdej iteracji, nowa para punktów musi mieć ten sam związek z nowym przedziałem, jaki poprzednia para miała z poprzednim przedziałem.
- Aby to osiągnąć, wybieramy względne pozycje dwóch punktów jako τ i $1 - \tau$, gdzie $\tau^2 = 1 - \tau$, więc $\tau = (\sqrt{5} - 1)/2 \approx 0,618$ i $1 - \tau \approx 0,382$.
- Niezależnie od tego, który podprzedział zostanie zachowany, jego długość będzie wynosić τ w stosunku do poprzedniego przedziału, a wewnętrzny punkt zachowany będzie na pozycji albo τ, albo $1 - \tau$ w stosunku do nowego przedziału.
- Aby kontynuować iterację, musimy obliczyć tylko jedną nową wartość funkcji, w punkcie komplementarnym.
- Ten wybór punktów próbnych nazywa się metodą złotego podziału.
- Metoda złotego podziału jest bezpieczna, ale szybkość zbieżności jest tylko liniowa, z stałą $C \approx 0,618$.

2.5 Opis algorytmu

1. **Inicjalizacja**:
   - Losowe rozmieszczenie punktów $x^{(1)}, x^{(2)}, \ldots, x^{(n-1)}$.
   - Ustawienie punktów początkowego $x^{(0)}$ i końcowego $x^{(n)}$.
2. **Obliczanie funkcji celu**:
   - Obliczenie wartości funkcji celu F dla zadanej trasy.
3. **Obliczanie gradientu**:
   - Wyznaczenie gradientu funkcji celu ∇F.

4. **Przeszukiwanie liniowe**:
    - Znalezienie optymalnego kroku α metodą złotego podziału.
5. **Aktualizacja pozycji**:
    - Aktualizacja pozycji punktów $x^{(i)}$ zgodnie z regułą:

$$x_{\text{new}}^{(i)} = x_{\text{old}}^{(i)} - \alpha \frac{\partial F}{\partial x^{(i)}}$$

6. **Iteracja**:
    - Powtarzanie kroków 2-5 przez zadany liczby iteracji lub do osiągnięcia zbieżności.

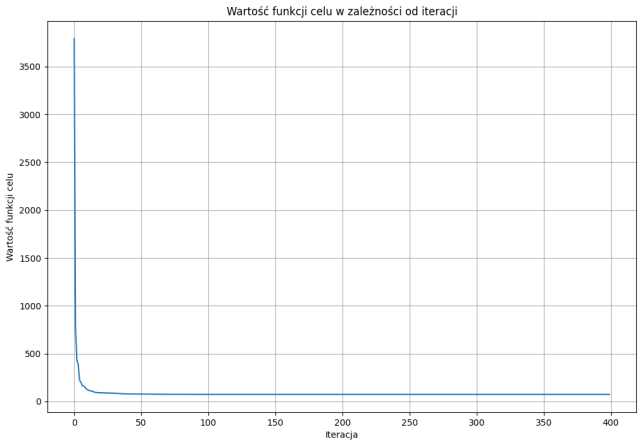
Implementacja

Poniżej znajduje się zaktualizowany kod w Pythonie implementujący powyższe kroki:

```
def objective_function(X): # Funkcja celu
    term1 = lambda_1 * np.sum([1 / (epsilon + np.linalg.norm(X[i] - r[j])**2)
    for i in range(1, n) for j in range(k)])
    term2 = lambda_2 * np.sum([np.linalg.norm(X[i+1] - X[i])**2 for i in range(n)])
    return term1 + term2

def gradient(X): # Wyznaczanie gradientu
    grad = np.zeros_like(X)
    for i in range(1, n):
        grad_term1 = np.sum([-2 * (X[i] - r[j]) /
        (epsilon + np.linalg.norm(X[i] - r[j])**2)**2 for j in range(k)], axis=0)
        grad_term2 = 2 * (X[i] - X[i-1]) + 2 * (X[i] - X[i+1])
        grad[i] = lambda_1 * grad_term1 + lambda_2 * grad_term2
    return grad

def golden_section_search(f, a, b, tol=1e-5): # Metoda zlotego podzialu
    invphi = (np.sqrt(5) - 1) / 2
    invphi2 = (3 - np.sqrt(5)) / 2
    h = b - a
    if h <= tol:
        return (a + b) / 2
    n = int(np.ceil(np.log(tol / h) / np.log(invphi)))
    c = a + invphi2 * h
    d = a + invphi * h
    yc = f(c)
    yd = f(d)
```

```
    for k in range(n - 1):
        if yc < yd:
            b = d
            d = c
            c = a + invphi2 * (b - a)
            yd = yc
            yc = f(c)
        else:
            a = c
            c = d
            d = a + invphi * (b - a)
            yc = yd
            yd = f(d)
    if yc < yd:
        return (a + d) / 2
    else:
        return (c + b) / 2

def line_search(X, grad, alpha=1, tol=1e-5):
    f = lambda a: objective_function(X - a * grad)
    return golden_section_search(f, 0, alpha, tol)
# Wykonanie algorytmu największego spadku
X = x_init.copy()
objective_values = []
for iter in range(iterations):
    grad = gradient(X)
    alpha = line_search(X, grad)
    X -= alpha * grad
    objective_values.append(objective_function(X))
```

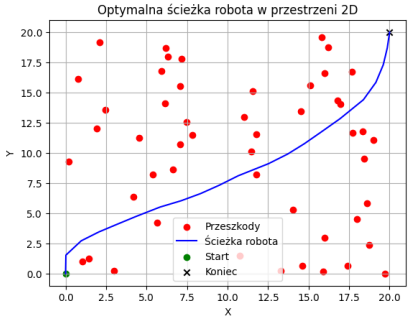
Algorytm wykonano kilkukrotnie dla różnych, inicjalizowanych losowo punktów wewnątrz ścieżki. Poniżej przedstawiono przykładowy wykres funkcji celu w zależności od liczby iteracji.



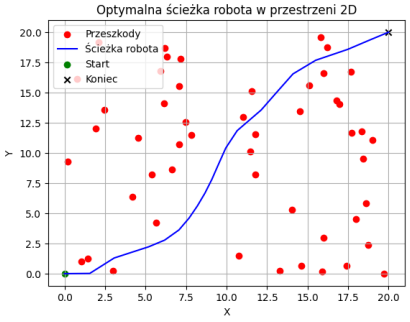
Wykres 1: Wartość funkcji celu w zależności od liczby iteracji

Jak widać jej wartość spada wraz z kolejnymi iteracjami co jest pożądane.

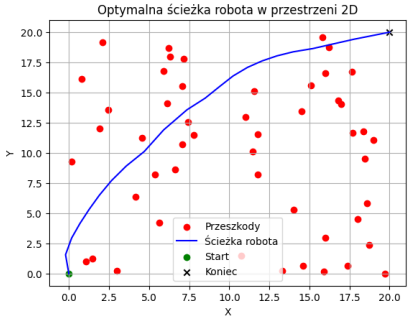
Przedstawiono przykładowe ścieżki wyznaczone przez algorytm dla różnych rozmieszczeń przeszkód.



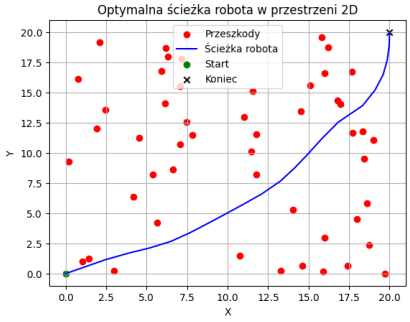
(a) Ścieżka 1



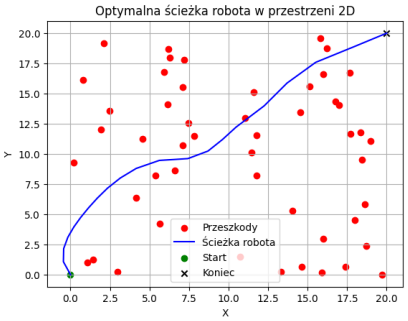
(b) Ścieżka 2



(c) Ścieżka 3



(d) Ścieżka 4



(e) Ścieżka 5

Wykres 2: Wyznaczone przez algorytm optymalne ścieżki dla robota

Porównano także wartości funkcji celu przy ostatniej iteracji dla 5 wywołań funkcji (za każdym razem przeszkody rozmieszczano niezależnie, od nowa).

Iteracja	**Końcowa wartość funkcji celu**
1	70.7
2	70.6
3	79.3
4	113.8
5	68.6

Tabela 3: Wartość funkcji celu w ostatniej iteracji dla kolejnych wywołań

3 Wnioski

1. **Badanie ekstremów**

   Znalezienie ekstremów i punktów siodłowych funkcji wielu zmiennych nie jest trudnym zadaniem dzięki analizie znaków wartości własnych Hesjanu. Ustalanie tych punktów jest pomocne, gdy nasz problem zależy od większej liczby zmiennych.

2. **Skuteczność optymalizacji**

   Jeśli możemy ustalić priorytety dla naszego problemu, np. zależy nam na ustaleniu ścieżki robota omijającej przeszkody, jednocześnie podążającego możliwie krótką trasą, możemy to osiągnąć poprzez określenie odpowiedniej funkcji kosztu. Funkcja ta musi uwzględniać parametry optymalizacyjne, które następnie będziemy minimalizować. Kluczowe jest odpowiednie ustalenie parametrów tej funkcji, ponieważ możemy przykładać różną wagę do różnych czynników. Dzięki wyznaczeniu gradientu możemy poruszać się iteracyjnie z niewielkim krokiem, znając jego wartości, co pozwala na znalezienie optymalnych parametrów.

3. **Wartości funkcji celu po zakończeniu optymalizacji**

   Stwierdzono, że różne rozmieszczenie przeszkód wpływa na końcowe wartości funkcji celu. W przypadku niekorzystnego rozkładu przeszkód funkcja może przyjąć wyższą wartość. Możemy jednak świadomie monitorować ten efekt i sprawdzić, czy jest on spowodowany długością ścieżki, czy bliskością przeszkód.

Bibliografia

[1] dr inż. Marcin Kuta, *Optimization*

[2] Prof. Michael T. Heath, *CS 450 – Numerical Analysis, Chapter 6: Optimization*