KI LaTeX DOKUMENT
Materiały do przedmiotu "Rozwiązywanie zadań odwrotnych"

Metody gradientowe - opis zagadnień związanych z klasycznymi algorytmami optymalizacyjnymi - metoda gradientu prostego

dr inż. Konrad M. Gruszka,*

Abstract. Ten dokument prezentuje ogólny opis "krok po kroku" działanie algorytmu gradientu prostego zaprojektowanego do optymalizacji rozkładu temperatur w jednowymiarowym, stacjonarnym modelu transferu ciepła z ustalonymi warunkami brzegowymi.

1 Wprowadzenie

Metoda gradientu prostego (Gradient Descent, GD) to jedna z najprostszych i najczęściej stosowanych metod optymalizacji, używana do znajdowania minimów **funkcji kosztu**. Jest metodą iteracyjną, która aktualizuje parametry w kierunku przeciwnym do gradientu funkcji, co stopniowo prowadzi do znalezienia wartości minimalnej.

2 Idea metody

Metoda gradientu prostego aktualizuje rozwiązanie iteracyjnie według wzoru:

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \tag{1}$$

gdzie:

- x_k – aktualne wartości parametrów (np. warunki brzegowe T_{left}, T_{right}),
- α – współczynnik uczenia (learning rate), który kontroluje wielkość kroku optymalizacji,
- $\nabla f(x_k)$ – gradient funkcji kosztu w punkcie x_k, wskazujący kierunek najszybszego wzrostu funkcji.

Metoda polega na iteracyjnym przesuwaniu się w kierunku przeciwnym do gradientu, ponieważ gradient wskazuje lokalne nachylenie funkcji (czyli kierunek największego wzrostu funkcji kosztu), a my chcemy ją minimalizować.

* Katedra Informatyki, Wydział Informatyki i Sztucznej Inteligencji (kgruszka@icis.pcz.pl)

3 Jak wykorzystać Gradient Descent do rozwiązania problemu odwrotnego w jednowymiarowym transferze ciepła?

W naszym przypadku chcemy znaleźć warunki brzegowe (T_{left},T_{right}), które minimalizują funkcję kosztu:

$$J(T_{\text{left}}, T_{\text{right}}) = ||T_{\text{model}} - T_{\text{target}}||_2 \tag{2}$$

gdzie:
T_{model} – rozkład temperatury uzyskany z modelu numerycznego,
T_{target} – docelowy (rzeczywisty) rozkład temperatury.

Gradient funkcji kosztu opisuje, jak zmiana warunków brzegowych wpływa na różnicę między modelem a rzeczywistym rozkładem temperatury.

(1) $J(T_{left}, T_{right})$ – funkcja kosztu zależna od warunków brzegowych.
- Mierzy różnicę między przewidywanym (T_{model}) a rzeczywistym (T_{target}) rozkładem temperatury.
- Celem jest znalezienie takich warunków brzegowych, które minimalizują tę różnicę.

(2) T_{model} – rozkład temperatury uzyskany z rozwiązania równania przewodnictwa cieplnego dla określonych warunków brzegowych.

(3) T_{target} – docelowy rozkład temperatury (rzeczywisty, zmierzony lub założony profil temperatury, który chcemy uzyskać).

(4) $||T_{model} - T_{target}||_2$ – norma L_2 (euklidesowa) mierząca różnicę między dwoma wektorami temperatur.

Co oznacza norma L_2 (euklidesowa)? Norma L_2 oblicza **średniokwadratowy** błąd dopasowania:

$$||T_{\text{model}} - T_{\text{target}}||_2 = \sqrt{\sum_{i=1}^{N} (T_{\text{model},i} - T_{\text{target},i})^2} \tag{3}$$

gdzie:
N – liczba punktów w siatce,
$T_{model,i}$ – temperatura obliczona w punkcie i,
$T_{target,i}$ – docelowa temperatura w punkcie i.

Norma L_2 to pierwiastek sumy kwadratów różnic pomiędzy wartościami temperatury uzyskanej z modelu a temperaturą docelową. Oznacza to, że im większa różnica między T_{model} a T_{target}, tym większa wartość funkcji kosztu. W naszym przypadku mamy problem odwrotny, w którym próbujemy znaleźć warunki brzegowe $J(T_{left}$ i $, T_{right})$, które minimalizują błąd:

$$\arg \min_{T_{\text{left}}, T_{\text{right}}} J(T_{\text{left}}, T_{\text{right}}) \tag{4}$$

co oznacza, że chcemy znaleźć takie wartości $J(T_{left}$ i $, T_{right})$, dla których różnica między przewidywanym a rzeczywistym rozkładem temperatury jest najmniejsza.

Dlaczego minimalizujemy tę funkcję?
Jeśli $J(T_{\text{left}}, T_{\text{right}})$ jest małe, oznacza to, że T_{model} dobrze pasuje do T_{target}, a więc znalezione warunki brzegowe są poprawne. Minimalizacja normy L_2 jest standardową metodą w problemach dopasowania i optymalizacji.

4 Implementacja algorytmu

Aby zrealizować metodę należy postępować zgodnie z poniższym schematem działania

(1) Wygeneruj docelowy rozkład temperatury.
- Wykorzystaj wcześniej napisaną funkcję *simulate_heat_transfer(N, T0, TN, max iter, tolerance=None)* dla jednorodnego, jednowymiarowego przypadku

(2) Określ funkcję kosztu $J(T_{left}, T_{right})$ *def cost_function(boundary_conditions):* która zwróci normę różnicy T_{model} -T_{target} (patrz np.linalg.norm())

(3) Oblicz gradient funkcji kosztu *def numerical_gradient(f, x, epsilon=1e-5)*

(4) Zaimplementuj funkcję def gradient_descent(f, initial_guess, learning_rate=1.0, max_iters=100, tol=1e-3), która iteracyjnie aktualizuje wartości warunków brzegowych w kierunku przeciwnym do gradientu funkcji kosztu, dążąc do jego minimalizacji. Proces ten jest kontynuowany aż do osiągnięcia zbieżności lub przekroczenia maksymalnej liczby iteracji.

(5) Weryfikacja i wizualizacja wyników

Gradient funkcji kosztu to wektor zawierający pochodne cząstkowe funkcji względem jej parametrów. W kontekście Twojego problemu, gradient funkcji kosztu $J(T_{left}, T_{right})$ względem warunków brzegowych jest definiowany jako:

$$\nabla J = \left[\frac{\partial J}{\partial T_{\text{left}}}, \frac{\partial J}{\partial T_{\text{right}}}\right] \tag{5}$$

Gradient ten jest obliczany numerycznie za pomocą metody różnic skończonych w funkcji *numerical_gradient*. Podejście to polega na przybliżeniu pochodnych cząstkowych poprzez analizę zmian funkcji kosztu przy niewielkich perturbacjach poszczególnych parametrów:

$$\frac{\partial J}{\partial T_i} \approx \frac{J(T_i + \epsilon) - J(T_i - \epsilon)}{2\epsilon} \tag{6}$$

gdzie T_i reprezentuje T_{left} lub T_{right}, a ϵ to mała wartość (np. 10^{-5}) służąca do perturbacji zmiennej.
W kontekście naszego kodu, funkcja **numerical_gradient** iteruje przez każdy parametr (tj. T_{left} i T_{right}), obliczając wartość funkcji kosztu dla nieznacznie zwiększonego i zmniejszonego parametru, a następnie stosuje powyższy wzór do przybliżenia pochodnej cząstkowej względem tego parametru.

Dla każdego parametru (w tym przypadku T_{left} i T_{right}, procedura jest następująca:

(1) Obliczenie wartości funkcji kosztu dla nieznacznie **zwiększonego** parametru: $J(T_{left} + \epsilon T_{right})$

(2) Obliczenie wartości funkcji kosztu dla nieznacznie **zmniejszonego** parametru: $J(T_{\text{left}} - \epsilon, T_{\text{right}})$

(3) Obliczenie przybliżenia pochodnej cząstkowej względem danego parametru:

$$\frac{\partial J}{\partial T_{\text{left}}} \approx \frac{J(T_{\text{left}} + \epsilon, T_{\text{right}}) - J(T_{\text{left}} - \epsilon, T_{\text{right}})}{2\epsilon} \tag{7}$$

Analogicznie oblicza się pochodną cząstkową względem T_{right}. Wartość ϵ reprezentuje małą perturbację i jest kluczowa dla dokładności przybliżenia. Zbyt duża wartość ϵ może prowadzić do niedokładności, podczas gdy zbyt mała może powodować problemy numeryczne związane z precyzją obliczeń. W naszym przypadku dobra wartość to około $\epsilon = 10^5$

5 Podpowiedzi

Poniżej znajduje się kod źródłowy implementujący metodę gradientu prostego do optymalizacji warunków brzegowych w modelu temperatury.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# Funkcja rozwiązująca model temperatury dla zadanych warunków brzegowych
def solve_temperature(T_left, T_right, N=100, tolerance=0.1):
    """
    Rozwiązuje jednowymiarowy, stacjonarny problem przewodnictwa ciepła
    metodą różnic skończonych.

    Parametry:
    - T_left: temperatura na lewym brzegu pręta
    - T_right: temperatura na prawym brzegu pręta
    - N: liczba węzłów w siatce
    - tolerance: kryterium zbieżności iteracji

    Zwraca:
    - T: rozkład temperatury w pręcie
    """
    dx = 1.0 / (N - 1)  # Rozmiar kroku przestrzennego
    k_al = 237.0  # Współczynnik przewodnictwa cieplnego aluminium
    k_wood = 0.12  # Współczynnik przewodnictwa cieplnego drewna
    x_interface = N // 2  # Pozycja granicy między materiałami

    # Inicjalizacja temperatury
    T = np.zeros(N)
    T[0] = T_left
    T[-1] = T_right

    # Współczynniki przewodnictwa cieplnego dla każdego punktu
    k = np.array([k_al] * x_interface + [k_wood] * (N - x_interface))

    # Iteracyjnie rozwiązujemy równanie różnicowe metodą Gaussa-Seidela
    for _ in range(10000):  # Maksymalna liczba iteracji
```

```python
        T_old = T.copy()  # Zachowanie poprzednich wartości temperatury
        for i in range(1, N - 1):  # Pętla po węzłach wewnętrznych
            k_avg = 0.5 * (k[i - 1] + k[i])  # Średnia przewodność cieplna
            T[i] = (k_avg * (T_old[i - 1] + T_old[i + 1])) / (2.0 * k_avg)

        # Sprawdzanie kryterium zbieżności
        if np.linalg.norm(T - T_old, np.inf) < tolerance:
            break

    return T

# Generowanie docelowego rozkładu temperatury (z modelu bazowego)
T_target = solve_temperature(100, 0)

# Funkcja kosztu dla optymalizacji (różnica między rozwiązaniem a docelowym rozkładem temperatury)
def cost_function(boundary_conditions):
    """
    Funkcja oceniająca błąd między obliczonym a docelowym rozkładem temperatury.

    Parametry:
    - boundary_conditions: lista [T_left, T_right], czyli warunki brzegowe do optymalizacji

    Zwraca:
    - Norma różnicy L2 między obliczonym a docelowym rozkładem temperatury
    """
    T_left, T_right = boundary_conditions  # Pobranie warunków brzegowych
    T_model = solve_temperature(T_left, T_right)  # Obliczenie rozkładu temperatury
    return np.linalg.norm(T_model - T_target, 2)  # Norma L2 jako miara błędu

# Funkcja obliczająca numeryczny gradient funkcji kosztu
def numerical_gradient(f, x, epsilon=1e-5):
    """
    Oblicza numeryczny gradient funkcji f w punkcie x przy użyciu różnicy skończonej.

    Parametry:
    - f: funkcja, dla której obliczamy gradient
    - x: punkt, w którym obliczamy gradient (wektor 2D: [T_left, T_right])
    - epsilon: mała wartość do obliczania różnicy skończonej

    Zwraca:
    - grad: wektor gradientu [df/dT_left, df/dT_right]
    """
    grad = np.zeros_like(x)
    for i in range(len(x)):
        x_plus = x.copy()
        x_minus = x.copy()
        x_plus[i] += epsilon
        x_minus[i] -= epsilon
        grad[i] = (f(x_plus) - f(x_minus)) / (2 * epsilon)  # Różnica skończona

    return grad

```

```python
# Gradient Descent do optymalizacji warunków brzegowych
def gradient_descent(f, initial_guess, learning_rate=1.0, max_iters=100, tol=1e-3):
    """
    Optymalizuje warunki brzegowe metodą gradientu prostego.

    Parametry:
    - f: funkcja kosztu do minimalizacji
    - initial_guess: początkowe warunki brzegowe [T_left, T_right]
    - learning_rate: krok optymalizacji
    - max_iters: maksymalna liczba iteracji
    - tol: tolerancja błędu (warunek stopu)

    Zwraca:
    - optymalne warunki brzegowe (T_left_opt, T_right_opt)
    """
    x = np.array(initial_guess, dtype=float)

    for i in range(max_iters):
        grad = numerical_gradient(f, x)  # Obliczenie gradientu numerycznie
        x -= learning_rate * grad  # Aktualizacja parametrów w kierunku spadku gradientu

        # Warunek stopu (jeśli gradient jest bardzo mały)
        if np.linalg.norm(grad) < tol:
            print(f"Zbieżność osiągnięta po {i} iteracjach")
            break

    return x

# Inicjalne zgadywanie warunków brzegowych
initial_guess = [50, 50]

# Optymalizacja warunków brzegowych
T_left_opt_grad, T_right_opt_grad = gradient_descent(cost_function, initial_guess)

# Obliczenie odtworzonego rozkładu temperatury
T_reconstructed_grad = solve_temperature(T_left_opt_grad, T_right_opt_grad)

# Wizualizacja wyników
plt.plot(np.linspace(0, 1, 100), T_target, '-o', label="Oryginalny rozkład temperatury")
plt.plot(np.linspace(0, 1, 100), T_reconstructed_grad, '--', label="Odtworzony rozkład
↪ temperatury (gradient)")
plt.xlabel("Pozycja")
plt.ylabel("Temperatura")
plt.legend()
plt.show()

# Wyświetlenie odtworzonych warunków brzegowych
print(f"Odtworzona temperatura na lewym brzegu: {T_left_opt_grad:.2f}°C")
print(f"Odtworzona temperatura na prawym brzegu: {T_right_opt_grad:.2f}°C")
```

6 Podsumowanie

- $J(T_{left}, T_{right})$ mierzy błąd dopasowania między symulowanym a docelowym profilem temperatury.
- Norma L_2 to średniokwadratowy błąd dopasowania, czyli standardowa metryka stosowana w problemach regresji i optymalizacji.
- Minimalizacja tej funkcji $J(T_{left}, T_{right})$ pozwala znaleźć najbardziej prawdopodobne warunki brzegowe, które wyjaśniają obserwowane dane.