KI LATEX DOKUMENT
Materiały do przedmiotu "Rozwiązywanie zadań odwrotnych"

Metody gradientowe - opis zagadnień związanych z klasycznymi algorytmami optymalizacyjnymi - metoda BFGS

dr inż. Konrad M. Gruszka,*

Abstract. Ten dokument prezentuje ogólny opis "krok po kroku" działanie algorytmu BFGS uywanego do optymalizacji rozkładu temperatur w jednowymiarowym, stacjonarnym modelu transferu ciepła w celu ustalenia warunków brzegowych.

1 Wprowadzenie

Metoda BFGS (Broyden-Fletcher-Goldfarb-Shanno) to jedna z najpopularniejszych metod optymalizacji quasi-Newtona, stosowana do znajdowania lokalnych minimów funkcji nieliniowych. Wykorzystuje przybliżenie macierzy Hesjana (macierzy drugich pochodnych) funkcji kosztu, co pozwala na efektywne i szybkie zbieżności w procesie optymalizacji.

2 Idea metody

Metody quasi-Newtona, takie jak BFGS, dążą do przybliżenia macierzy Hesjana bez konieczności jej bezpośredniego obliczania, co jest korzystne w przypadku dużych problemów optymalizacyjnych. BFGS aktualizuje przybliżenie macierzy Hesjana w każdej iteracji, wykorzystując informacje o gradientach funkcji kosztu z poprzednich kroków.

W kontekście jednowymiarowego przewodnictwa ciepła w materiale jednorodnym, problem odwrotny polega na wyznaczeniu nieznanych warunków brzegowych T_{left} i T_{right}, tak aby uzyskany rozkład temperatury T(x) w materiale odpowiadał zadanemu, rzeczywistemu profilowi $T_{target}(x)$. W tym celu potrzebna będzi tzw. **funkcja kosztu**. Funkcja kosztu (ang. cost function lub objective function, czasem loss function w kontekście uczenia maszynowego) to funkcja matematyczna określająca wartość błędu lub straty modelu. W metodach optymalizacyjnych, takich jak BFGS (Broyden-Fletcher-Goldfarb-Shanno) lub innych metod gradientowych, **funkcja kosztu to funkcja, którą chcemy zminimalizować** (lub zmaksymalizować w niektórych przypadkach).

Wyobraź sobie, że masz pewien model matematyczny, który ma parametry x. Funkcja kosztu mierzy, jak dobrze ten model działa dla danej wartości x. Chcemy znaleźć optymalne wartości parametrów, dla których ta funkcja osiąga minimum.

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

W ogólności metody gradientowe polegają na stopniowej poprawie wartości zmiennych x, aby zmniejszyć wartość funkcji kosztu. Główne kroki obejmują:

(1) Obliczenie gradientu funkcji kosztu $\nabla f(x)$ – określa kierunek najszybszego wzrostu funkcji.
(2) Przejście w kierunku przeciwnym do gradientu – aby znaleźć minimum funkcji.
(3) Powtarzanie kroku 1 i 2, aż gradient będzie bliski zeru (czyli znaleziono minimum).

Funkcja kosztu w metodzie BFGS. BFGS to jedna z metod quasi-Newtonowskich, używana do minimalizacji funkcji kosztu. Różni się od podstawowych metod gradientowych tym, że:

- Nie wymaga obliczania drugich pochodnych (Hessianu - czyli macierzy Hessego H), ale aproksymuje je.
- Używa macierzy przybliżonej odwrotnej Hessianu do szybszego i bardziej stabilnego wyboru kroku optymalizacyjnego.

Funkcja kosztu w tej metodzie spełnia tę samą rolę – wyznacza wartość, którą minimalizujemy, ale BFGS jest efektywniejsza niż np. zwykły gradient prosty (Gradient Descent).

Funkcję kosztu definiujemy jako normę różnicy między obliczonym a docelowym rozkładem temperatury:

$$J(T_{\text{left}}, T_{\text{right}}) = ||T_{\text{model}} - T_{\text{target}}||_2 \tag{1}$$

$$x_{k+1} = x_k - \alpha \nabla f(x_k) \tag{2}$$

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.

2.1 Schemat działania BFGS

2.1.1 Inicjalizacja

- Wybieramy początkowy punkt x_0.
- Przyjmujemy początkową macierz przybliżoną B_0 (zwykle macierz jednostkowa).

2.1.2 Iteracje

Dla każdej iteracji k:

(1) Obliczamy kierunek poszukiwań:

$$p_k = -B_k^{-1} \nabla f(x_k) \tag{3}$$

(2) Wykonujemy krok w kierunku p_k zgodnie z regułą:

$$x_{k+1} = x_k + \alpha_k p_k \tag{4}$$

gdzie α_k to długość kroku (zwykle dobierana metodą linii, np. warunki Wolfe'a).

(3) Obliczamy wektory różnicowe:

$$s_k = x_{k+1} - x_k \tag{5}$$

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k) \tag{6}$$

(4) Aktualizujemy macierz aproksymującą odwrotność Hessego:

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \tag{7}$$

Jest to tzw. **formuła BFGS**.

2.1.3 Warunek stopu

Iteracje kończą się, gdy norma gradientu $||\nabla f(x_k)||$ jest mniejsza od zadanej tolerancji.

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

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

$$J(T_{\text{left}}, T_{\text{right}}) = ||T_{\text{model}} - T_{\text{target}}||_2 \tag{8}$$

gdzie:

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

3.1 BFGS w Python

Metoda BFGS jest dostępna w pakiecie *scipy* i można jej użyć np tak:

```
from scipy.optimize import minimize

result = minimize(fun, x0, method='BFGS', jac=None, options=None)
```

gdzie:

fun – funkcja celu, którą chcemy zminimalizować (powinna zwracać wartość skalarną).
x0 – początkowe przybliżenie rozwiązania (lista, krotka lub numpy.ndarray).
method='BFGS' – określenie, że używamy metody BFGS.
jac (opcjonalne) – gradient funkcji celu (None oznacza, że gradient będzie obliczany numerycznie, ale można podać własną funkcję zwracającą gradient jako numpy.ndarray).
options – słownik z opcjami np. 'disp': True, 'maxiter': 1000.

3.1.1 Przykład użycia

Chcemy zminimalizować funkcję:

$$f(x) = (x_0 - 1)^2 + (x_1 - 2)^2$$

korzystając z *minimize*:

```
import numpy as np
from scipy.optimize import minimize

# Definicja funkcji celu
def func(x):
    return (x[0] - 1)**2 + (x[1] - 2)**2

# Gradient funkcji (pochodne cząstkowe)
def grad(x):
    return np.array([2 * (x[0] - 1), 2 * (x[1] - 2)])

# Punkt początkowy
x0 = np.array([0, 0])

# Minimalizacja BFGS
result = minimize(func, x0, method='BFGS', jac=grad,
 ↪  options={'disp': True})

# Wynik
print("Minimum znalezione w:", result.x)
```

Output:

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 2
         Function evaluations: 3
         Gradient evaluations: 3
Minimum znalezione w: [1. 2.]
```

Algorytm szybko znajduje minimum funkcji.
Użyliśmy jawnie zdefiniowanego gradientu (grad(x)). Można go pominąć, a minimize obliczy go numerycznie. Opcja disp=True pokazuje szczegóły optymalizacji.

4 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.