WSI ćwiczenie 1 - Implementacja algorytmu gradientu prostego

Natasza Jarecka

2025

1 Opis algorytmu

Algorytm gradientu prostego to algorytm numeryczny mający na celu znalezienie minimum funkcji celu. Aby zastosować powyższą metodę, funkcja musi być ciągła i różniczkowalna, a także wypukła w badanej dziedzinie.

Kolejne kroki algorytmu:

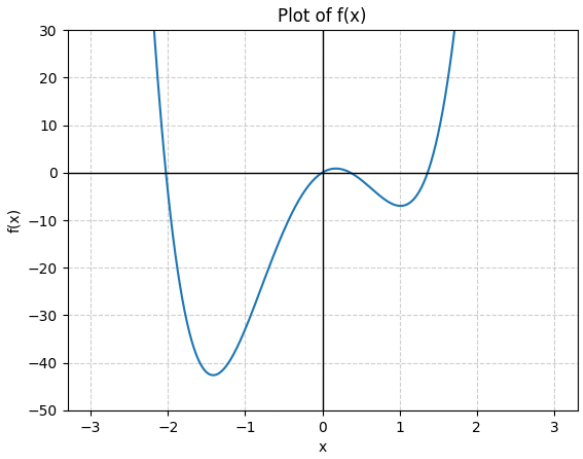
1. Wybieramy punkt startowy x_0.
2. Obliczamy kolejny punkt wzorem $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$.
3. Sprawdzamy czy obliczony punkt spełnia kryterium stopu $\|\nabla f(x_k)\| \leq \epsilon$.

Ze względów praktycznych dodatkowym kryterium zatrzymania algorytmu jest osiągnięcie zadanej liczby iteracji, aby uniknąć zbyt długiego czasu oczekiwania.

2 Badane funkcje

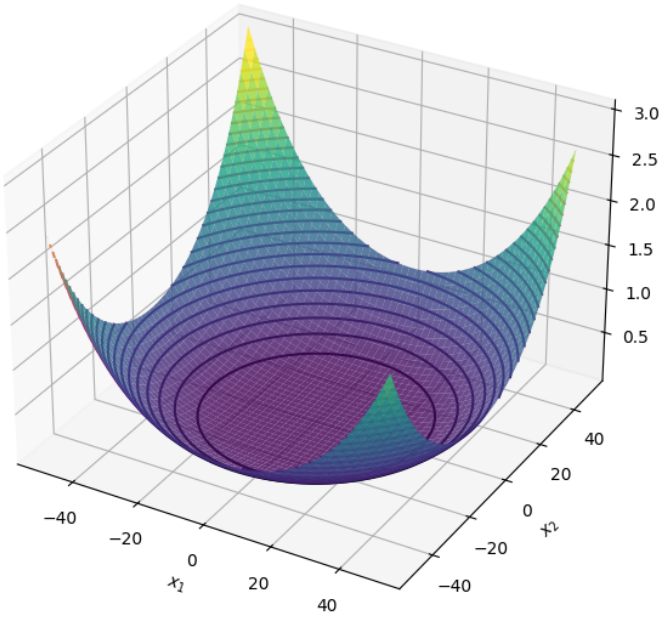
Pierwszą z badanych funkcji jest funkcja jednej zmiennej:

$$f(x) = 10x^4 + 3x^3 - 30x^2 + 10x$$



Drugą funkcja jest funkcja dwóch zmiennych:

$$g(x_1, x_2) = (x_1 - 2)^4 + (x_2 + 3)^4 + 2(x_1 - 2)^2(x_2 + 3)^2$$



3 Implementacja

Implementacja algorytmu wykorzystuje dwie biblioteki: NumPy - do korzystania z tablic NumPy i norm wektorów, SymPy - do operacji symbolicznych.

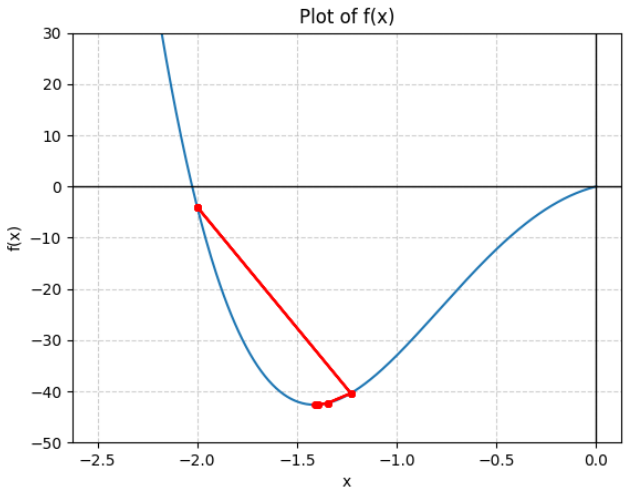
Zamiast obliczać gradient funkcji numerycznie przy pomocy NumPy, zdecydowałam się obliczyć go symbolicznie, ze względu na większą dokładność i przejrzystość.

Funkcja gradient() przyjmuje jako argumenty funkcję anonimową lambda i wektor poziomy zmiennych funkcji. Wartością zwracaną jest gradient w formie funkcji anonimowej

Przechodząc do właściwego algorytmu gradientu prostego, funkcja gradient_descent() przyjmuje jako argumenty: funkcję w formie lambdy, wektor zmiennych, długość kroku, punkt startowy (w formie wektora), precyzję oraz maksymalną liczbę iteracji. W pętli while sprawdzany jest warunek stopu. Dopóki pętla się wykonuje obliczany jest gradient w obecnym punkcie oraz wyznaczany jest punkt kolejny. Zwiększany jest licznik iteracji, oraz każdy punkt dodawany jest do tablicy NumPy (która służy jedynie do śledzenia ścieżki algorytmu). Wartościami zwracanymi przez funkcję jest punkt, który algorytm wyznaczył jako minimum, liczba wykonanych iteracji oraz tablica ze wszystkimi punktami.

4 Wyniki

Dla funkcji $f(x)$ algorytm znalazł minimum w punkcie x ≈ -1.412. Poniżej znajduje się wizualizacja "szukania"punktu.



Ścieżka algorytmu

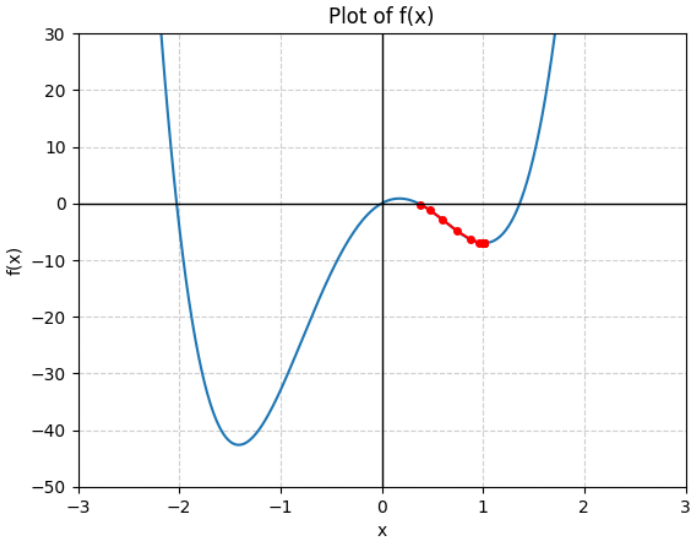
Użyty został poniższy zestaw parametrów:

- krok = 0.005
- punkt startowy = -2
- precyzja = 0.000001
- liczba iteracji = 10000

Minimum zostało osiągnięte już po 14 iteracjach.

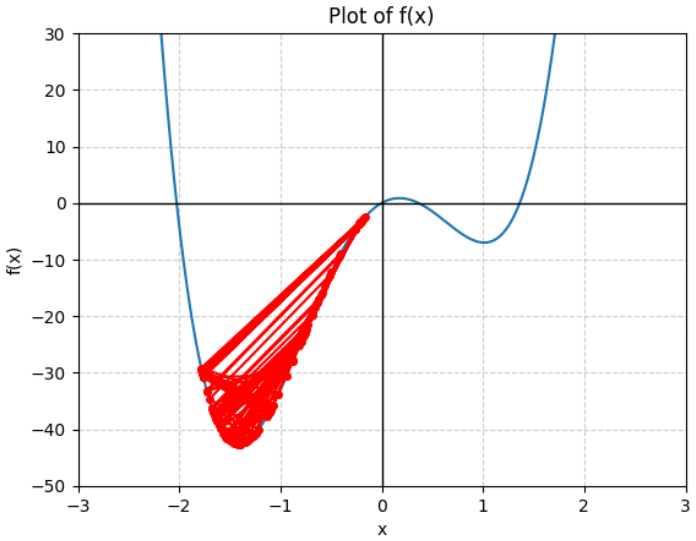
Obliczając minimum globalne metodą "kartka i długopis"rzeczywiście otrzymujemy taki wynik (jest on ułamkiem okresowym, więc ograniczę się do rozpatrywania trzech pierwszych liczb po przecinku).

Funkcja $f(x)$ ma jednak swoje utrudnienia. Po pierwsze, funkcja posiada drugie minimum, tyle że lokalne. Przy niektórych punktach startowych i długościach kroku algorytm "wpada"w minimum lokalne i uznaje je za optymalne rozwiązanie.



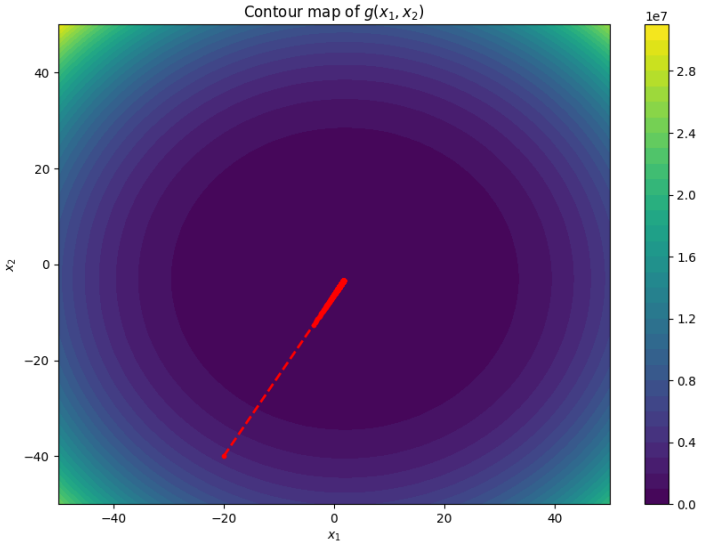
Osiągnięcie minimum lokalnego w punkcie x = 1.012, dla punktu początkowego = 0.383 i kroku = 0.01

Kolejną kwestią jest kształt funkcji. Minimum globalne znajduje się wewnątrz długiego, parabolicznego wgłębienia. Dla niektórych punktów startowych i długości kroków algorytm oscyluje wokół minimum i nie osiąga go w zamierzonej liczbie iteracji.



Oscylacje wokół minimum dla punktu początkowego = -0.5, kroku = 0.02 i 50 iteracji

Dla funkcji $g(x_1, x_2)$ algorytm nie znajduje jednoznacznego minimum, które powinno wynosić (2, -3). Wszystkie znalezione punkty są jedyne zbliżone do faktycznego minimum. Dla poniższego zestawu parametrów zostało one znalezione w punkcie $x_1 = 1.819$, $x_2 = -3.304$



Ścieżka algorytmu

- krok = 0.0001
- punkt startowy = (-20, -40)
- precyzja = 0.000001
- liczba iteracji = 10000

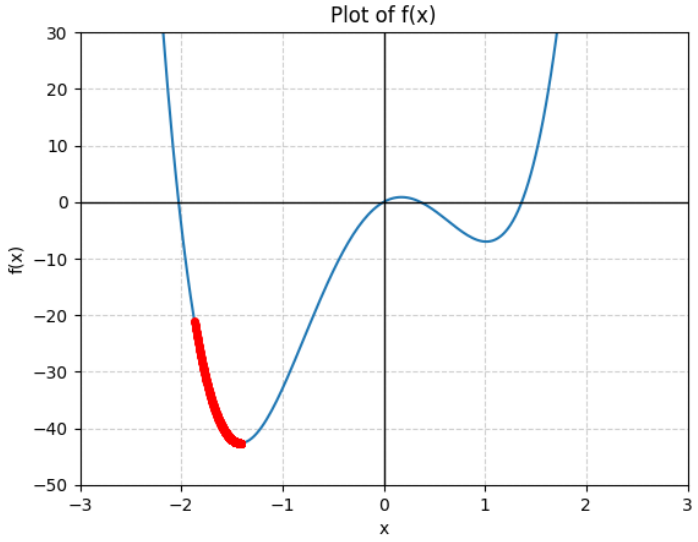
Algorytm wykorzystał maksymalną liczbę iteracji.

Kształt funkcji $g(x_1, x_2)$ jest dość problematyczny. W okolicach minimum znajduje się wiele punktów o tych samych wartościach (obszar wydaje się być "płaski"). Gdy algorytm dociera do takiego obszaru, gradient wynosi niemal zero, więc kroki są bardzo małe, przez co minimum może nie zostać osiągnięte.

5 Eksperymenty

Moim początkowym eksperymentem była obserwacja jak zachowa się algorytm, gdy początkowym punktem będzie punkt z przedziału [-100, 100]. Jednak punkty rzędu kilku dziesięciu powodują dążenie algorytmu do nieskończoności zamiast do minimum, nawet przy małych krokach. Spowodowane jest to tym, że w tych punktach funkcja przyjmuje bardzo duże wartości. Dalsze eksperymenty zdecydowałam się przeprowadzać zaczynając od punktu z okolic minimum.

Za mały krok



Minimum znalezione przy punkcie początkowym = -1.869, kroku = 0.00003

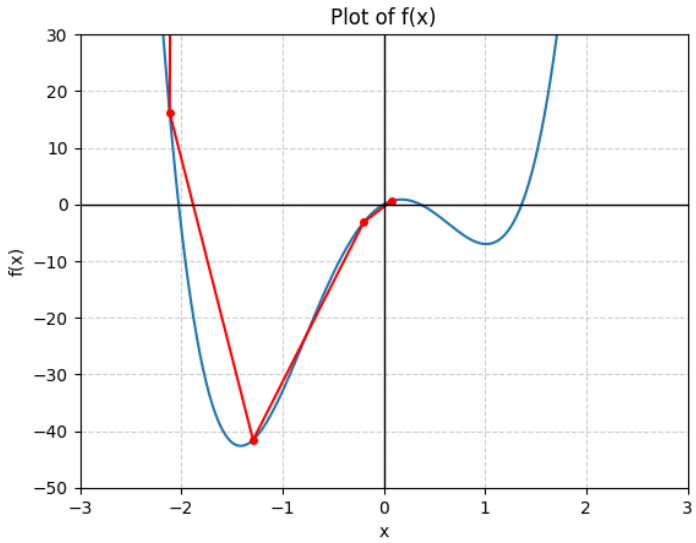
Dla kroku rzędu 0.00001 algorytm odnajduje minimum funkcji $f(x)$, jednak jak widać na wykresie, zbliża się do niego małymi krokami, co zajmuje więcej czasu. W powyższym przypadku trwało to aż 3816 iteracji.

Minimum znalezione przy punkcie początkowym = (-20, -40), kroku = 0.0000001

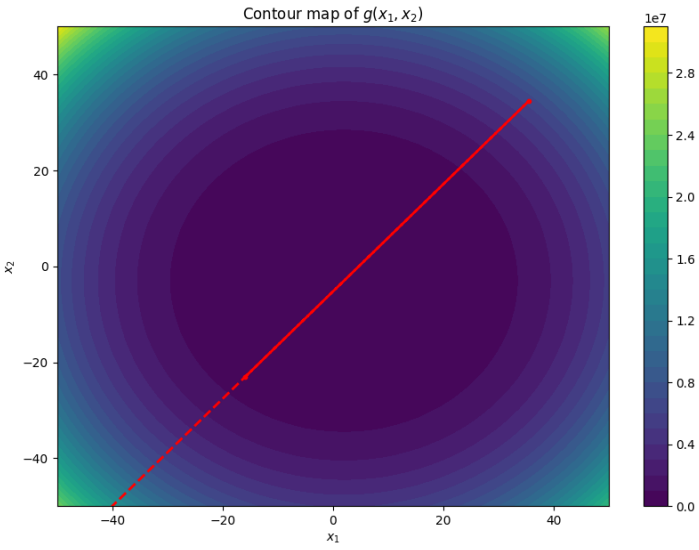
Dla kroku rzędu 0.0000001 algorytm nie znajduje minimum funkcji $g(x_1, x_2)$, dla 10000 iteracji zatrzymuje się daleko przed minimum (w powyższym przypadku w (-3.52, -12.3)). Algorytm "nie zdą-

żył"dotrzeć w okolice minimum w zadanej liczbie iteracji.

Za duży krok



Zachowanie algorytmu dla punktu początkowego = 0.076, kroku = 0.05



Zachowanie algorytmu dla punktu początkowego = (-15.866, -22.972), kroku = 0.001

Dla zbyt dużego kroku algorytm dąży ku nieskończoności, nie jest w stanie trafić do minimum.

6 Wnioski

Dla której funkcji algorytm działa lepiej?

Algorytm działa lepiej dla funkcji $f(x)$. Jest w stanie znaleźć dokładne minimum zanim osiągnie maksymalną liczbę iteracji. Dla funkcji $g(x_1, x_2)$ nie udało się uzyskać dokładnego wyniku. Przy każdym eksperymencie algorytm zatrzymuje się po osiągnięciu maksymalnej liczby iteracji. W najlepszym przypadku zwrócony punkt jest zbliżony do prawdziwego minimum, ale nigdy równy.

Powodem gorszego działania dla funkcji $g(x_1, x_2)$ jest to, że gradienty są bardzo małe w pobliżu minimum. Zmiany są minimalne, co powoduje powolne zbieganie.

Co można było zrobić lepiej?

Sposobem na lepsze działanie algorytmu byłoby zastosowanie zmiennej wartości kroku zamiast stałej. Pozwoliłoby to zmniejszyć krok, gdy algorytm próbuje zbiegać do nieskończoności lub zwiększyć go, gdy algorytm zbiega do minimum za wolno. Taki efekt dałby na przykład Adam Optimizer.