Sprawozdanie ćwiczenia nr 1 – Łukasz Szydlik

Cel i opis eksperymentów

Celem ćwiczenia jest zaimplementowanie metody gradientu prostego oraz wykorzystania go do obliczenia ekstremów funkcji opisanych w ćwiczeniu. Jakość algorytmu jest określona za pomocą liczby iteracji, która jest potrzebna do obliczenia ekstremum.

Instrukcja programu

Instalacja

1. Należy pobrać repozytorium

2. Wejść w folder lab1

3. Upewnić się, że posiadamy wymagane biblioteki:

‘pip install -r requirements.txt’

Uruchomienie programu

Pamiętamy o tym aby program uruchamiać bezpośrednio z folderu lab1.

1. W pliku compute_extremum.py podane są przykładowe wartości ‘B’ – długość kroku uczącego oraz ‘variables_start’. Można je ręcznie zmienić, a następnie uruchomić za pomocą komendy ‘python3 compute_extremum.py’.

W konsoli wyświetlą się ekstrema dla funkcji 1 oraz 2. Wykresy funkcji na której przedstawiono kolejne punkty schodzenia gradientu oraz obliczone ekstremum możemy zobaczyć w plikach Function1.png oraz Function2.png (folder Scores). Jeśli chcemy wyświetlić funkcje za pomocą matplotlib to w pliku compute_extremum.py wywołując funkcję plot_functionXD musimy usunąć parametr ścieżki zapisu do pliku.

2. Za pomocą komendy ‘python3 data_process.py’ możemy uruchomić plik data_process. Na podstawie danych z plików data1.json oraz data2.json (folder Data) możemy uzyskać pliki result1.txt i result2.txt (folder Scores), w których otrzymamy wiersze danych: długość kroku uczącego, zmienne startowe, ekstremum funkcji odpowiednio dla funkcji 1 i 2.

W plikach data1.json oraz data2.json możemy umieścić własne zbiory danych wartości ‘B’ oraz ‘variables_start’.

Aby zmienić precyzję, współczynnik redukcji B, limit redukcji B oraz maksymalną ilość iteracji należy je zmodyfikować w pliku gradinet_descent.py w konstruktorze GradientFunction lub utworzyć obiekt klasy pochodnej z odpowiednimi wartościami. Można stworzyć własną klasę pochodną od GradientFunction dla własnej funkcji. W tym celu należy stworzyć klasę GradientFunctionX i nadpisać metody function_value oraz next_variables.

Przy wywoływaniu metody grad_descent() można jako parametr wstawić True, co pozwoli nam obliczyć maksimum funkcji.

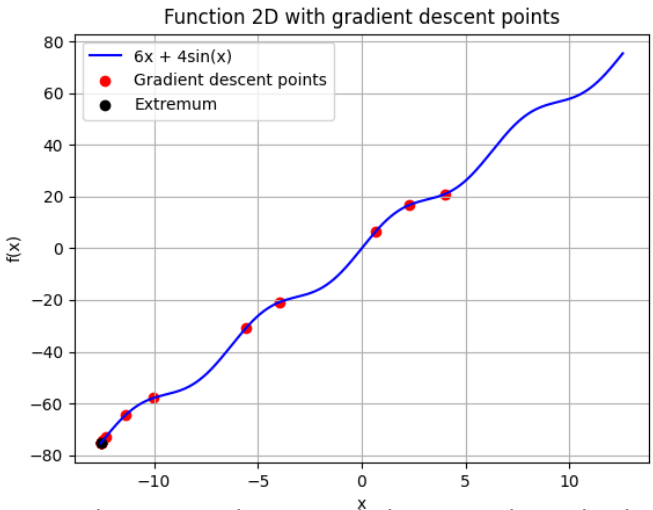
Wyniki ćwiczenia

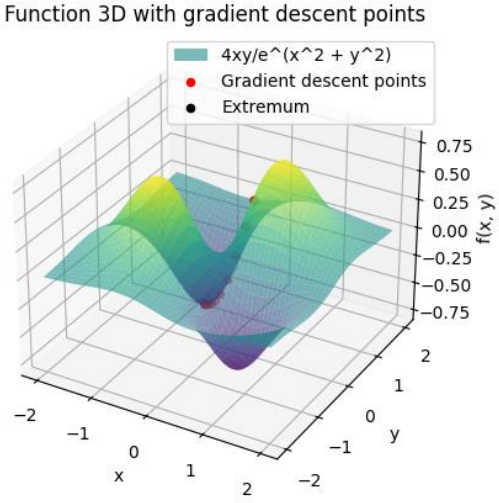
Wyniki zostały przeprowadzone dla kilku różnych długości korku uczącego ‘B’ oraz punktów startowych należących do dziedziny funkcji. Ich rezultat możemy zobaczyć w pliku results.xlsx (folder Scores).

Funkcja1		
Precyzja	Parametr	Średnia liczba iteracji
1,00E-06	0.1	58,67
1,00E-06	0.3	40,17
1,00E-06	0.5	37
1,00E-06	1	37
1,00E-06	3	38,5
1,00E-06	7	39,33
Precyzja	Parametr	Średnia liczba iteracji
1,00E-12	0.1	88,50
1,00E-12	0.3	71,83
1,00E-12	0.5	67,5
1,00E-12	1	67,5
1,00E-12	3	69,17
1,00E-12	7	68,83

Funkcja2		
Precyzja	Parametr	Średnia liczba iteracji
1,00E-06	0.1	42,17
1,00E-06	0.3	17,67
1,00E-06	0.5	22
1,00E-06	1	22,5
1,00E-06	3	20,33
1,00E-06	7	21,50
Precyzja	Parametr	Średnia liczba iteracji
1,00E-12	0.1	63,83
1,00E-12	0.3	36,67
1,00E-12	0.5	46
1,00E-12	1	43,5
1,00E-12	3	43
1,00E-12	7	44,80

1,00E-06	0.7	37,50
1,00E-12	0.7	66,50





Wykresy przedstawiają kolejne punkty schodzenia gradientu oraz ekstrema.

Wykres1: B=0.5, zmienne startowe: x=4

Wykres2: B=0.1, zmienne startowe: x=0.2, y=0.7

Wnioski

Jakość metody gradientu prostego, która jest określana jako ilość liczby iteracji (im mniej tym lepiej) jest zależna przede wszystkim od precyzji. Im większą precyzję chcemy uzyskać tym będzie większa ilość iteracji.

Istotną rzeczą jest długość kroku uczącego. Najlepszą jakość uzyskamy w pobliżu wartości, która jest granicą kiedy może wystąpić rozbieżność metody gradientu (2/maksymalna wartość własna hesjanu funkcji). Dla funkcji 1 jest to punkt 0.5.

Limit maksymalnej liczby kroków algorytmu jest nam jedynie potrzebny gdy liczba iteracji jest bardzo duża przez co algorytm działa zbyt długo. Taka sytuacja może wystąpić, gdy długość kroku uczącego jest zbyt mała, a odległość punktu startowego do ekstremum bardzo duża. W takiej sytuacji najlepiej zwiększyć długość kroku uczącego.

Rozmieszczenie punktu startowego oraz współczynnik redukcji kroku uczącego również pełnią ważną rolę. Jeśli rozmieszczenie punktu startowego jest w znacznej odległości od ekstremum to lepiej sprawdzi się większa długość kroku uczącego, która następnie zostanie zredukowana przez współczynnik redukcji długości kroku uczącego. Bardzo dobrze widać to dla odległych punktów startowych od ekstremum w funkcji 1, gdzie dla dużego kroku uczącego liczba iteracji jest taka sama lub mniejsza co dla małych wartości długości kroku uczącego.

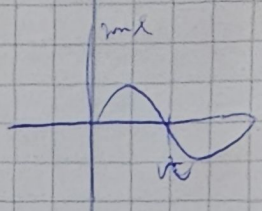
Problemy metody gradientu prostego

W przypadku występowania kilka ekstremum lokalnych funkcji metoda gradientu nie znajdzie ekstremum globalnego, gdyż będzie zmierzać do najbliższego ekstremum lokalnego. Bardzo dobrze to widać dla funkcji 2, gdzie dla wartości zmiennych startowych x=1.85, y=0.98 znalezione ekstremum wynosi w przybliżeniu 0.053, gdzie ekstremum globalne wynosi około -0.736. W celu zapobiegnięcia temu problemowi najlepiej byłoby wykorzystać metodę Newtona zamiast gradientu prostego.

Może również wystąpić sytuacja, że odległość do dwóch ekstremów będzie jednakowa. W takiej sytuacji funkcja będzie wyliczać kolejne coraz to niższe wartości dla kolejnych punktów, które będą równo odległe od ekstremów. Metoda gradientu znajdzie taki punkt o najniższej wartości i zatrzyma się w nim (Punkt siodłowy). W celu rozwiązania tego problemu można by dodać element pseudolosowy, który przesunąłby minimalnie punkt, aby mógł dotrzeć do jednego z ekstremów.

Notatki oraz obliczenia gradientów funkcji na kolejnej stronie

WSI Lab 1 337446 A=6 B=4 C=4



1) $f(x) = 6x + 4\sin(x)$

$x \in (-4\pi, 4\pi)$

$\nabla f(x) = 6 + 4\cos(x)$ $\quad \nabla^2 f(x) = -4\sin(x)$

$\nabla f(x) = 0$

$\sin$

max wartość własna

$\lambda = 4$

$f(x)$ ∩ dla $x \in \mathbb{R}$

β - parametr kroku

Jeżeli $\beta > \frac{1}{\lambda}$ ⇒ będą oscylacje

Jeżeli $\beta > \frac{2}{\lambda}$ ⇒ będzie rozbieżność

Maksimum po iteracji β optimal $\beta < 0$

$x_{t+1} = x_t - \beta \nabla f(x_t), \quad t = 1, 2, 3, \ldots$

$x_{t+1} = x_t - \beta(6 + 4\cos(x_t))$

$f(x_1) = 6x_1 + 4\sin(x_1)$

$f(x_2) = 6(x_1 - \beta(6 + 4\cos x_1)) + 4\sin(x_1 - \beta(6 + 4\cos x_1))$

2) $g(x,y) = \frac{4xy}{e^{x^2+y^2}}$

$x \in (-2, 2)$

$y \in (-2, 2)$

$$\nabla g(x,y) = \begin{bmatrix} \frac{4ye^{x^2+y^2} - 8x^2 y e^{x^2+y^2}}{e^{x^2+y^2}} \\ \frac{4xe^{x^2+y^2} - 8y^2 x e^{x^2+y^2}}{e^{x^2+y^2}} \end{bmatrix}$$

$x_{t+1} = x_t - \beta \nabla g(x,y)_1$

$y_{t+1} = y_t - \beta \nabla g(x,y)_2$