WSI LAB 1
Przeszukiwanie przestrzeni

Będkowski Patryk, 310603

I. WSTĘP

Niniejszy raport przedstawia ćwiczenie z zagadnienia przeszukiwania polegające na implementacji dwóch algorytmów metody najszybszego spadku gradientu oraz metody Newton'a.

II. URUCHOMIENIE

Do uruchomienia skryptu potrzebne są następujące narzędzia:

- Python w wersji 3.8.8
- Moduły:
    - numpy 1.20.1
    - sympy 1.8
    - plotly 5.3.1 (opcjonalne)

Aby uruchomić skrypt należy użyć polecenia:

```
python main.py
```

bądź wprowadzić odpowiednie argumenty do programu poleceniem:

```
python main.py -wielkość_kroku -max_iteracji -współrzędna_x_punktu -współrzędna_y_punktu
```

Wprowadzenie dodatkowych argumentów spowoduje modyfikację domyślnych wartości ustawionych w programie.

III. PRZYGOTOWANE PLIKI

W załączniku raportu przesyłam plik .zip wraz z implementacją wymaganych algorytmów. Znajdują się one w pliku `main.py`. Zawiera on implementację algorytmów **Steepest Gradient Descent** oraz **Newton's method** w formie klas. Owe klasy zostały tak skonstruowane, że mogą przyjmować dowolną funkcję oraz budować automatyczne wykresy.
W archiwum znajduje się również folder przedstawiający wizualizacje algorytmów.

IV. ANALIZA METOD

Parametr *beta* będę nazywał step_size bądź też wielkością kroku.

Testując oba algorytmy będę badał ich czas wykonywania, ilość iteracji, w których uzyskano/próbowano uzyskać optimum lokalne dla danego punktu startowego oraz wielkości kroku (*step_size*).

a. METODA GRADIENTU PROSTEGO

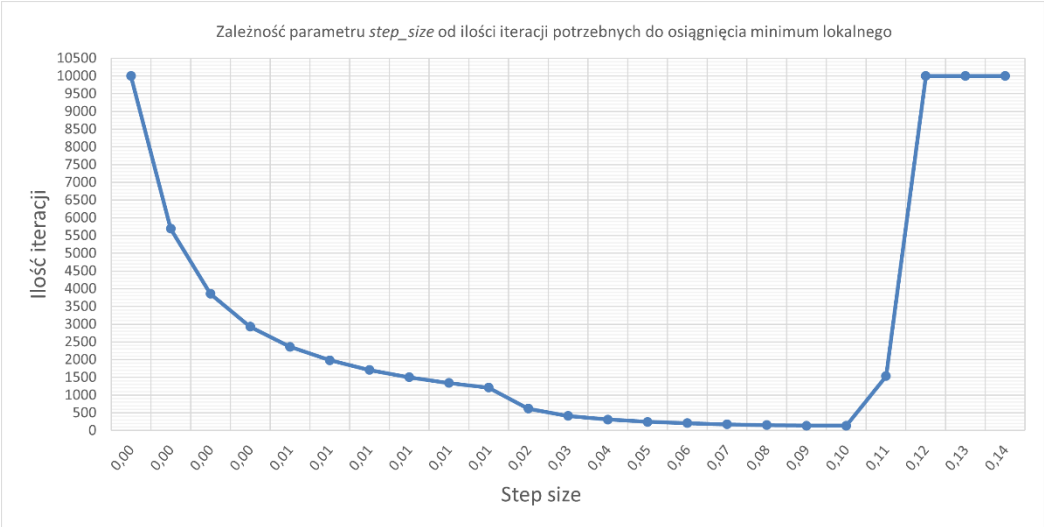
Metoda ta służy do znajdowania optimum lokalnego funkcji przy pomocy znajomości jej pierwszego rzędu. Określa ona kierunek podróży jako kierunek największego spadku wartości funkcji.

Przetestujmy zatem jak zmiana parametrów wpływa na czas wykonania algorytmu oraz ilość iteracji, w której zostanie znalezione optimum.

Punkt startowy (start_point)	Wielkość kroku (step_size)	Ilość iteracji	Maksymalna ilość iteracji	Czas wykonania [s]
(-5,-5)	0,002	3000	3000	7,192535
(-5,-5)	0,003	3000	3000	7,192535
(-5,-5)	0,004	2927	3000	7,147494
(-5,-5)	0,005	2362	3000	5,67916
(-5,-5)	0,01	1210	3000	2,898635
(-5,-5)	0,02	616	3000	1,470336
(-5,-5)	0,03	414	3000	0,998909
(-5,-5)	0,04	311	3000	0,756688
(-5,-5)	0,05	248	3000	0,574522
(-5,-5)	0,1	140	3000	0,361329
(-5,-5)	0,105	267	3000	0,703644
(-5,-5)	0,110	1534	3000	3,660842
(-5,-5)	0,115	3000	3000	6,104561

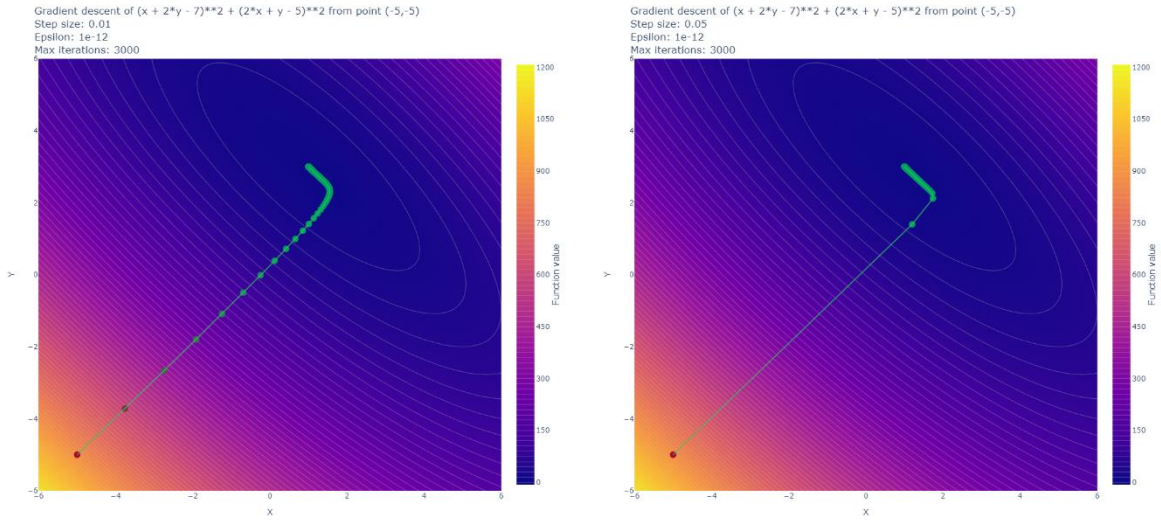
(tabela 1.1)

Z powyższej tabeli można wywnioskować, że parametr *step_size* wypływa na ilość kroków (a tym samym na czas wykonania) potrzebnych algorytmowi do uzyskania wartości zbliżonej do minimum lokalnego.
Algorytm najszybciej i z najmniejszą liczbą iteracji wyszukuje minimum lokalne dla parametru *step_size* = 0,1. Warto zaznaczyć, że gdy wartość tego parametru wzrośnie o 0,005, wyniki algorytmu ulegną pogorszeniu.



(wykres 1.1) Zależność *step_size* od ilości iteracji

Kolejne, warte zaznaczenia zjawisko wynikające z danych występuje, gdy algorytm w ustalonym limicie maksymalnych iteracji nie poradzi sobie ze znalezieniem minimum lokalnego (tj. warunek stopu związany z parametrem epsilon nie zostanie osiągnięty). Możemy to zauważyć gdy *step_size* równy jest jednej z wartości 0,002, 0,003 lub 0,115 (dla dwóch ostatnich z danych nie wynika, że algorytm nie wykonywałby więcej iteracji, mógłby wykonać dokładnie 3000 iteracji aby znaleźć min. lokalne). W tych przypadkach kolejne iteracje algorytmu zostały zatrzymane (ustalony epsilon) ze względu na osiągnięty limit.



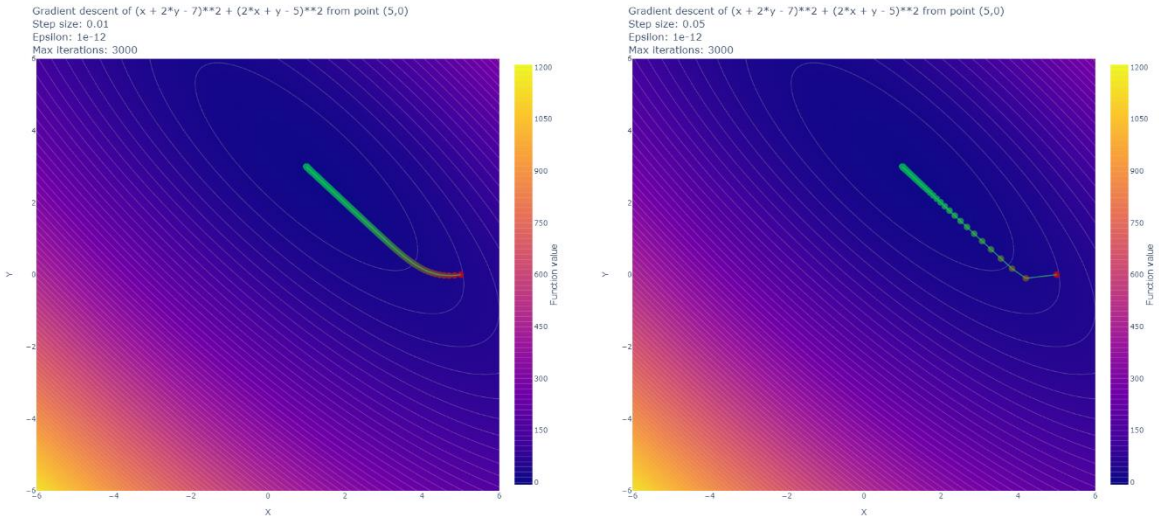
(wykres 1.2) Kolejne korki algorytmu dla step_size=0,01 (po lewej) oraz step_size=0,05 (po prawej) dla punktu startowego (-5, -5)

Do badania zależności pomiędzy parametrami *step_size* oraz *ilości iteracji* możemy również wykorzystać pomiary z innych punktów. Sprawdźmy zatem czy różne parametry *step_size* będą również wpływały na ilość iteracji dla punktu startowego położonego blisko minimum lokalnego w punkcie (1, 3).

Punkt startowy (start_point)	Wielkość kroku (step_size)	Ilość iteracji	Maksymalna ilość iteracji	Czas wykonania [s]
(5,0)	0,002	3000	3000	7,588912
(5,0)	0,003	3000	3000	7,370715
(5,0)	0,004	3000	3000	7,385728
(5,0)	0,005	2486	3000	6,141595
(5,0)	0,01	1272	3000	3,17189
(5,0)	0,02	647	3000	1,583442
(5,0)	0,03	434	3000	1,080985
(5,0)	0,04	326	3000	0,81374
(5,0)	0,05	260	3000	0,626571
(5,0)	0,1	128	3000	0,308281
(5,0)	0,105	244	3000	0,606552
(5,0)	0,110	1403	3000	3,22694
(5,0)	0,115	3000	3000	5,980448

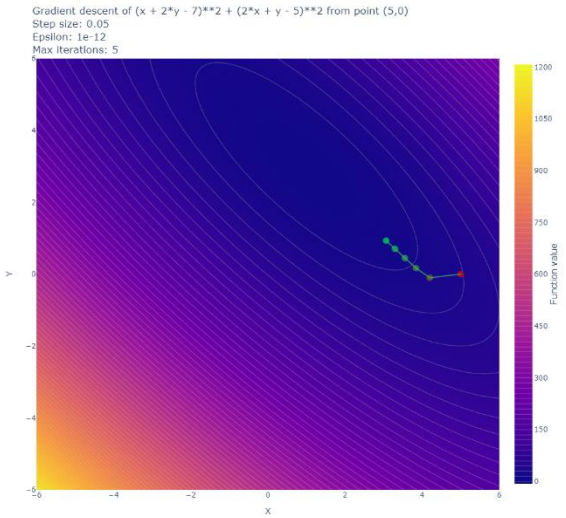
(tabela 1.2)

Wyniki mogą wydawać się podobne do tych dla punktu startowego (-5, -5). Widzimy jednakże drobny przyrost w ilościach iteracji dla poszczególnych parametrów *step_size.* Wynika to natury działania tego algorytmu. Możemy wywnioskować, że im bliżej punkt startowy znajduje się punktu optymalnego, tym kolejne kroki są coraz to mniejsze. Owy fakt możemy również zauważyć na wykresach przedstawiających kolejne kroki algorytmu.



(wykres 1.3) Kolejne korki algorytmu dla step_size=0,01 (po lewej) oraz step_size=0,05 (po prawej) dla punktu startowego (5, 0)

Jeżeli dodatkowo zmienimy parametr maksymalnej ilości iteracji (*max_iterations*) na wartość mniejszą, niż ilość kroków potrzebnej algorytmowi na znalezienie minimum lokalnego (dla danych *start_point* oraz *step_size*) to algorytm po prostu się zatrzyma. Owy przypadek został przedstawiony poniżej.

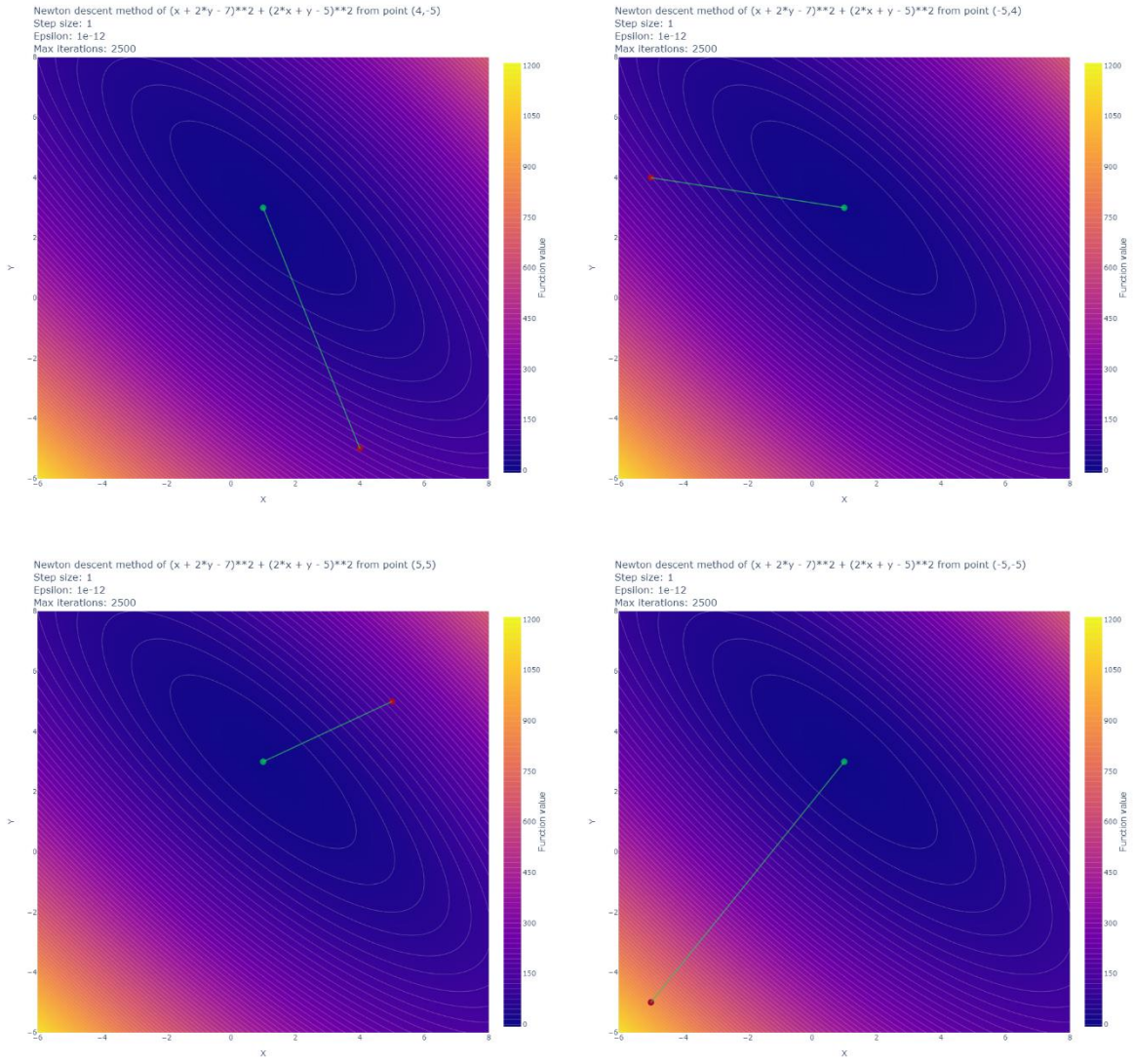


(wykres 1.4) Kolejne korki algorytmu dla step_size=0,05 dla punktu startowego (5, 0) oraz max_iterations=5

b. METODA NEWTONA

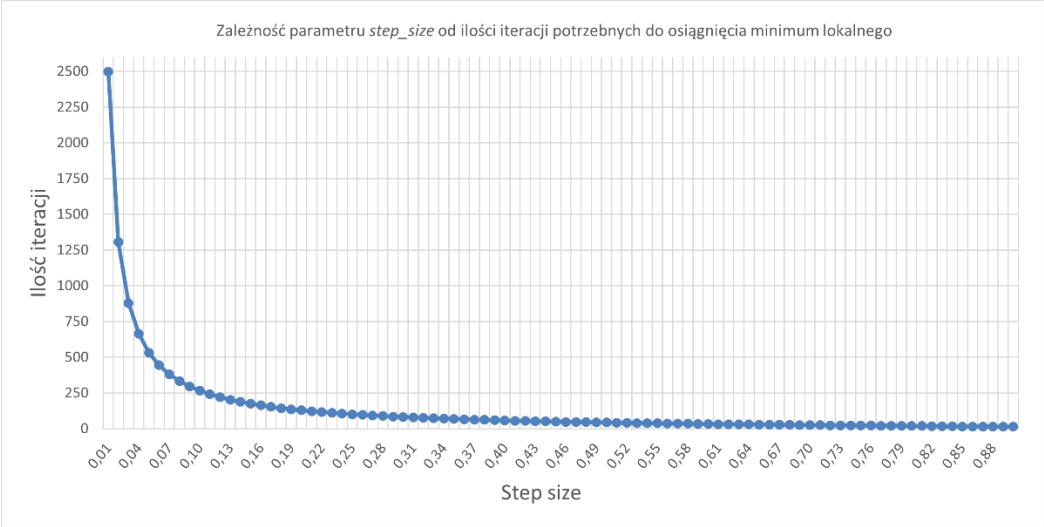
W optymalizacji znajomość informacji pierwszego rzędu może pomóc w określeniu kierunku podróży, ale nie pomaga w określeniu, jak daleko można przejść do lokalnego minimum. Metoda Newton'a wykorzystuje informację dotyczące drugiego rzędu funkcji, co czasami okazuje się lepszym rozwiązaniem.

Stosując metodę Newton'a dla funkcji Booth'a możemy znaleźć minimum lokalne nawet w jednym kroku iteracji. Wynika to z faktu, iż owa funkcja jest kwadratowa. Z uwagi na to, że metoda Newton'a bazuje na aproksymacji Szeregiem Taylora metoda Newton'a w pewnym sensie automatycznie wykonuje adaptacyjny rozmiar korku (który zmienia kierunek) zgodnie z szybkością zmiany gradientu. Można to zrobić w jednym kroku, ponieważ szereg Taylora drugiego rzędu będący kwadratowym przybliżeniem funkcji sprowadza się do rozwiązania układu pierwszego rzędu funkcji kwadratowej (układ liniowych równań). Rozpatrywana funkcja Bootha jest kwadratowa, zatem znalezienie optimum lokalnego następuje w jednym kroku dla parametru *step_size=1*.[1] [2]



(wykres 2.1) Optimum lokalne w jednym kroku – Funkcja Booth'a, step_size=1

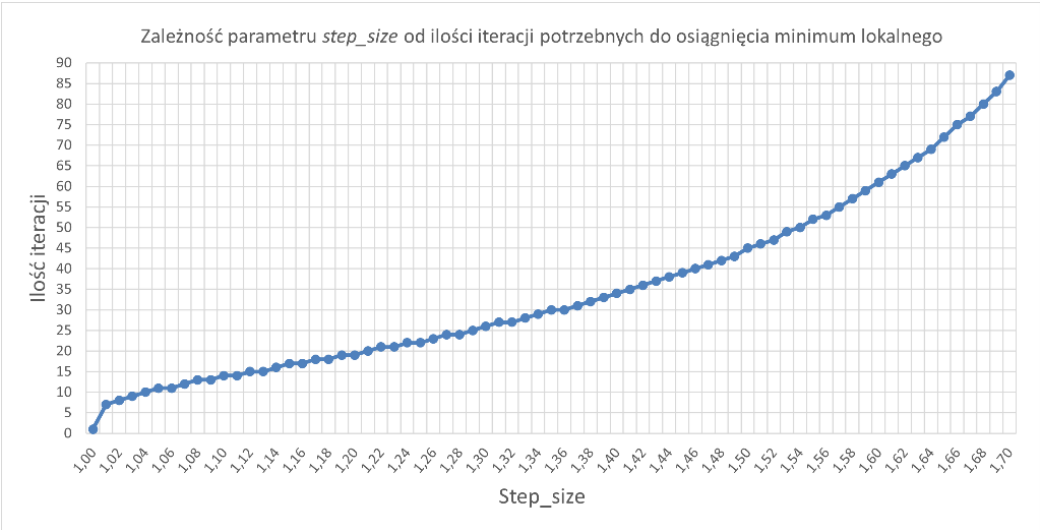
Wprowadzenie parametru *step_size* (ustawienie jego wartości innej niż 1) powoduje zwiększenie liczby iteracji potrzebnych do uzyskania minimum lokalnego. Poniższy wykres pokazuje tą zależność, dla $step\_size \in (0,1)$.



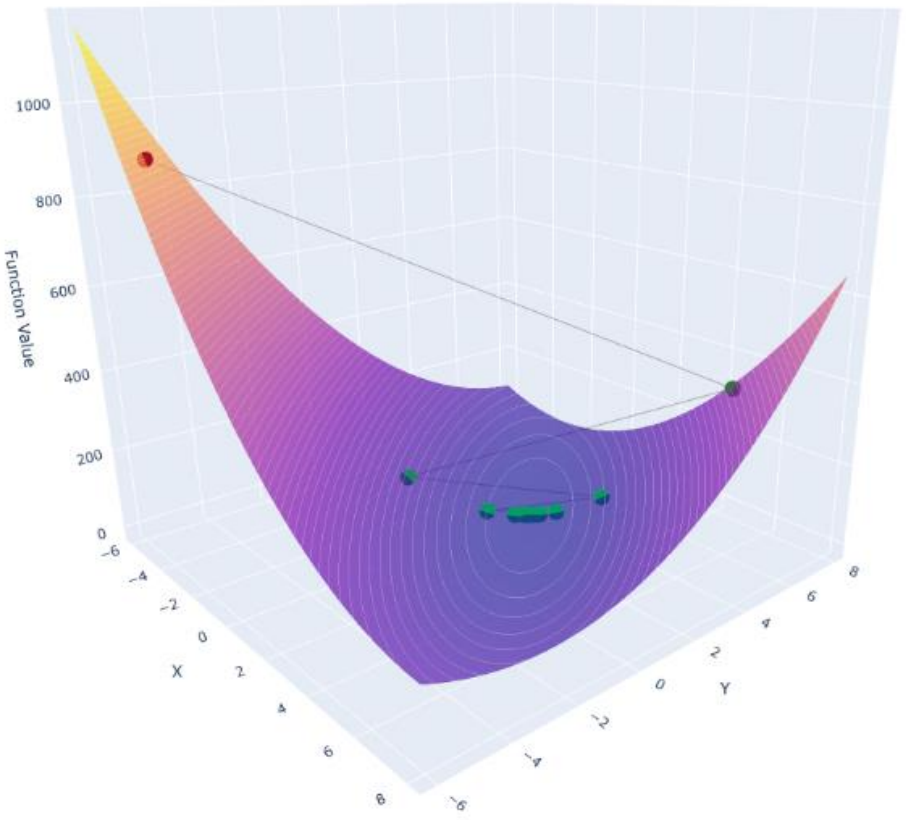
(wykres 2.2) Zależność *step_size* od ilości iteracji

Z powyższego wykresu możemy stwierdzić, że zastosowanie parametru *step_size* mniejszego od zera powoduje, zredukowanie efektywności wyszukiwania minimum lokalnego dla danej funkcji.

Z drugiej stronny, użycie parametru *step_size* $\in (1, \infty)$, spowoduje „przeskakiwanie" minimum lokalnego w kolejnych iteracjach i również wydłużenie ilości iteracji potrzebnych na jego osiągnięcie.



(wykres 2.3) Zależność *step_size* od ilości iteracji



(wykres 2.4) Kolejne kroki algorytmu dla *step_size=1,60, start_point =(-5, -5)*

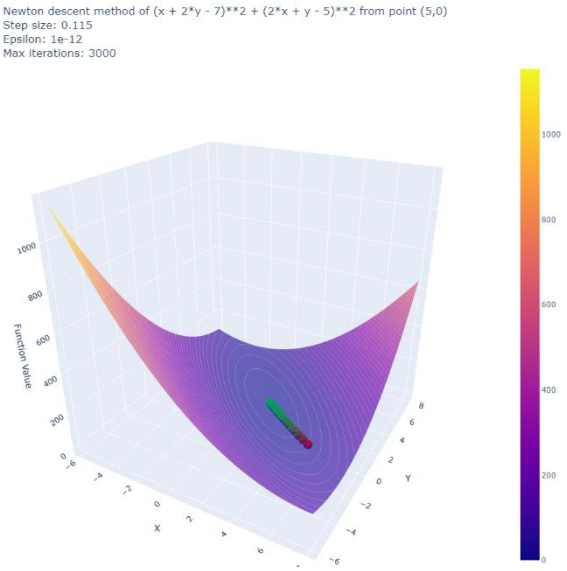
Wykonajmy również porównanie metody Newton'a i prostego gradientu dla tych samych, wcześniej badanych parametrów.

Punkt startowy (start_point)	Wielkość kroku (step_size)	Ilość iteracji	Maksymalna ilość iteracji	Czas wykonania [s]
(-5,-5)	0,002	3000	3000	10,52442
(-5,-5)	0,003	3000	3000	10,62303
(-5,-5)	0,004	3000	3000	10,70059
(-5,-5)	0,005	3000	3000	10,54143
(-5,-5)	0,01	2555	3000	9,05744
(-5,-5)	0,02	1306	3000	4,576998
(-5,-5)	0,03	880	3000	3,071303
(-5,-5)	0,04	664	3000	2,331124
(-5,-5)	0,05	533	3000	1,868702
(-5,-5)	0,1	267	3000	0,950208
(-5,-5)	0,105	254	3000	0,891813
(-5,-5)	0,110	242	3000	0,854779
(-5,-5)	0,115	232	3000	0,822749
(5, 0)	0,002	3000	3000	10,62303
(5, 0)	0,003	3000	3000	10,55946
(5, 0)	0,004	3000	3000	10,66255
(5, 0)	0,005	3000	3000	10,56245
(5, 0)	0,01	2486	3000	8,767014
(5, 0)	0,02	1272	3000	4,46707

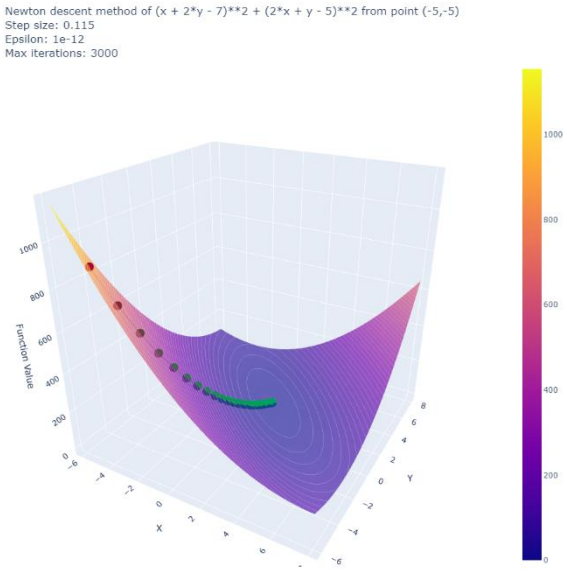
(5, 0)	0,03	857	3000	3,002073
(5, 0)	0,04	647	3000	2,26106
(5, 0)	0,05	520	3000	1,81516
(5, 0)	0,1	260	3000	0,927858
(5, 0)	0,105	248	3000	0,886808
(5, 0)	0,110	236	3000	0,810739
(5, 0)	0,115	226	3000	0,792723

(tabela 1.3)

Z powyższej tabeli wynika, że dla obu punktów ilość iteracji maleje wraz ze wzrostem parametru *step_size*. Jeżeli porównamy te wyniki wraz z tymi, dla metody gradientowej (tabela 1.2 i 1.3) to widzimy zaletę metody gradientu dla tych wartości parametru *step_size,* które okazały się zbyt duże dla metody gradientowej. Dodatkowo pomniejszającą się liczbę wymaganych iteracji wraz ze wzrostem step_size (do pewnego momentu) możemy zauważyć na *wykresie 1.5*.
Z drugiej jednak strony dla pewnego zakresu wartości step_size (od 0,005 do 0.1), metoda gradientowa szybciej i w mniejszej liczbie iteracji znajduje minimum lokalne tej funkcji dla tych właśnie parametrów.



(wykres 2.5) Kolejne kroki algorytmu dla step_size=0,115, start_point =(5, 0)



(wykres 2.6) Kolejne kroki algorytmu dla step_size=0,115, start_point =(-5, -5)

V. PODSUMOWANIE

Badając obie metody znajdywania optimum funkcji Booth'a, możemy stwierdzić, że metoda Newton'a, jako metoda drugiego rzędu zbiega znacznie szybciej do minimum lokalnego niż metoda gradientu. Musimy jednak wziąć pod uwagę, że metoda Newton'a, jest bardziej złożona obliczeniowo i pamięciowo. Obliczanie odwrotności macierzy Hessego może być bardzo kosztowne dla pewnych przypadków w wielowymiarowej przestrzeni. Istnieją jednak sposoby zmniejszenia kosztów poprzez np. zaktualizowanie hesjanu z poprzedniej iteracji.

Źródła

[1] https://suzyahyah.github.io/calculus/optimization/2018/04/06/Taylor-Series-Newtons-Method.html
[2] https://jermwatt.github.io/machine_learning_refined/notes/4_Second_order_methods/4_4_Newtons.html
[3] https://www.numerical-tours.com/matlab/optim_2_newton/
[4] https://dke.maastrichtuniversity.nl/westra/Education/ANO/nnfl%20deel%202.pdf
[5] https://calculus.subwiki.org/wiki/Newton%27s_method_for_optimization_of_a_function_of_multiple_variables
[6] https://suzyahyah.github.io/calculus/optimization/2018/04/06/Taylor-Series-Newtons-Method.html
[7] https://stats.stackexchange.com/questions/395653/do-there-exist-adaptive-step-size-methods-for-newton-raphson-optimization/395670#395670
[8] https://sites.stat.washington.edu/adobra/classes/536/Files/week1/newtonfull.pdf