WSI - Ćwiczenie 1

Imię i Nazwisko: Wiktor Kulesza

Numer albumu: 304050

Podział plików:

gradient_descent.py – implementacja algorytmu.

functions.py – zawiera wzór optymalizowanych funkcji oraz funkcje służące do gradientu w punkcie.

main.py – główny plik programu tworzący grafy podane niżej w celu przetestowania działania algorytmu

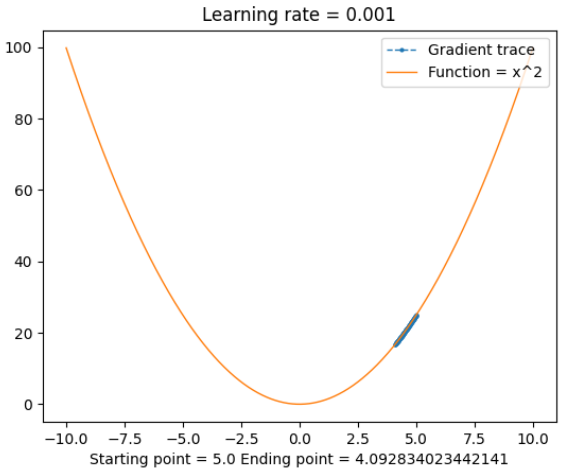
Opis zaimplementowanego algorytmu:

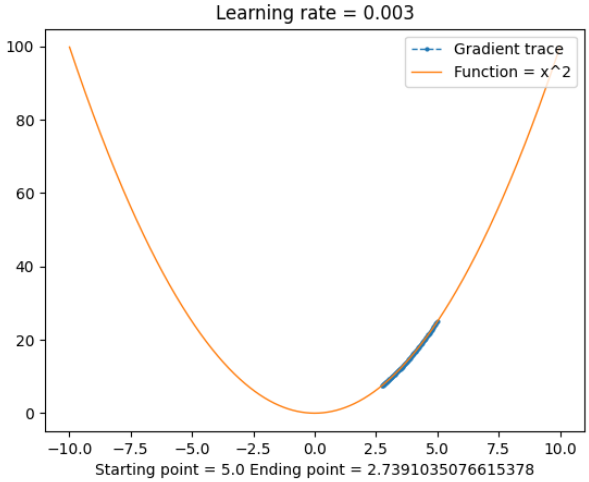
Wybieram punkt startowy. Obliczam d = antygradient (wektor pochodnych cząstkowych) przemnożony przez –1.
Obieram nowy punkt, będący poprzednim punktem przesuniętym o wektor beta*d
Powtarzam algorytm do momentu osiągnięcia kryterium stopu. Kryterium stopu jest osiągnięcie podanej liczby iteracji bądź sytuacja, w której każdy element gradientu jest mniejszy od epsilonu.
Algorytm ten zwraca macierz trace, która zawiera wszystkie punkty które wyznaczył algorytm

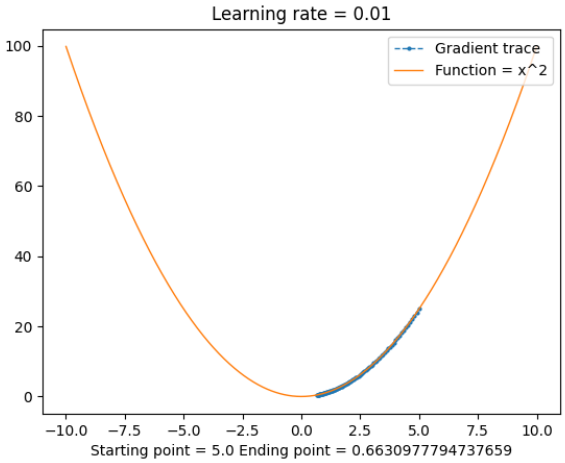
Przeprowadzone Badania:

Najpierw zająłem się sprawdzaniem wpływu wielkości rozmiaru kroku (learning rate) na działanie algorytmu. W tym celu przeprowadziłem parę eksperymentów dla obu problemów,
zarówno funkcji 2D jak i 3D.
Poniżej zamieszczam wyniki badań (wyniki tabelaryczne oraz wykresy funkcji wraz z naniesionym gradientem).
W każdym z tych przypadków postanowiłem ustawić ilość iteracji na 100,
tzn. algorytm gradientu prostego obliczał kroki 100 razy bądź do momentu, gdy krok każdej zmiennej był mniejsza od epsilonu.

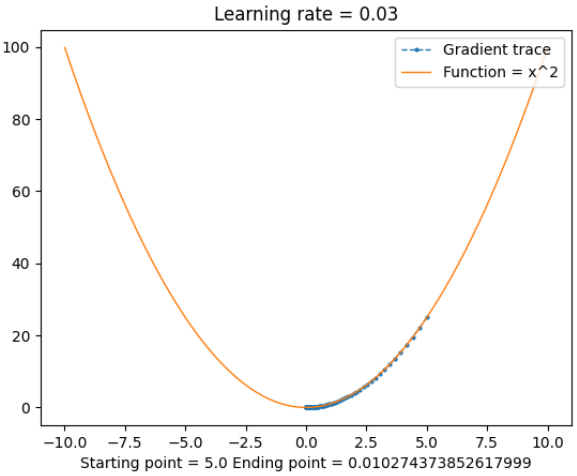
Beta	Ilość iteracji	Początkowy punkt	Końcowy punkt	Czy udało się dojść do minimum?
0.001	100	5.0	4.093	Nie
0.003	100	5.0	2.739	Nie
0.01	100	5.0	0.663	Nie
0.03	100	5.0	0	Tak
0.1	100	5.0	0	Tak
0.3	100	5.0	0	Tak
0.9	100	5.0	0	Tak
1.1	100	5.0	36.22	Nie
0.001	100	[1, -1]	[0.885, -0.885]	Tak
0.003	100	[1, -1]	[0.885, -0.885]	Tak
0.005	100	[1, -1]	[0.960, -0.960]	Tak
0.008	100	[1, -1]	[0.897, -0.897]	Tak
0.01	100	[1, -1]	[-0.433, 0.433]	Nie

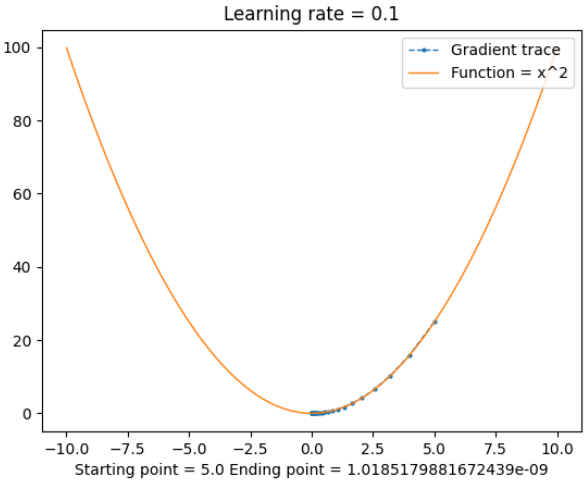


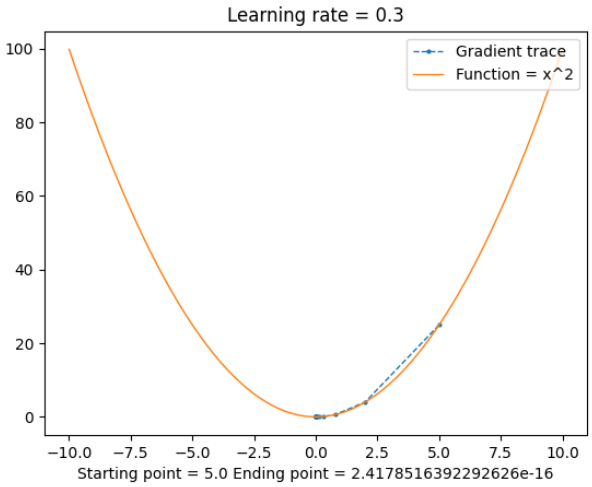


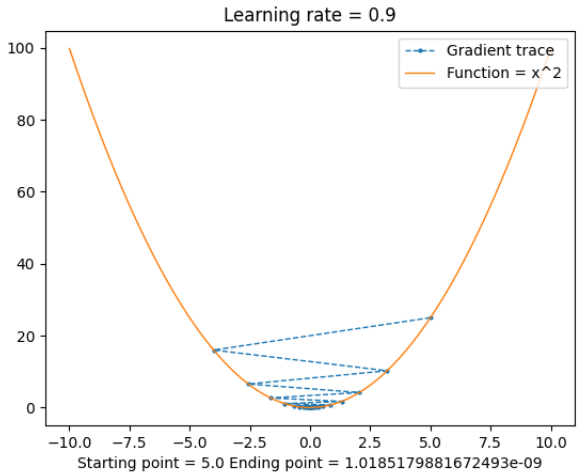


W pierwszych trzech przypadkach widzimy, że wielkość kroku jest ewidentnie za mała, przez co algorytm w 100 krokach nie jest w stanie dojść do minimum funkcji. Rozwiązaniem tego problemu mogłoby być zwiększenie wartości kroku bądź zwiększenie ilości iteracji funkcji gradientu prostego.

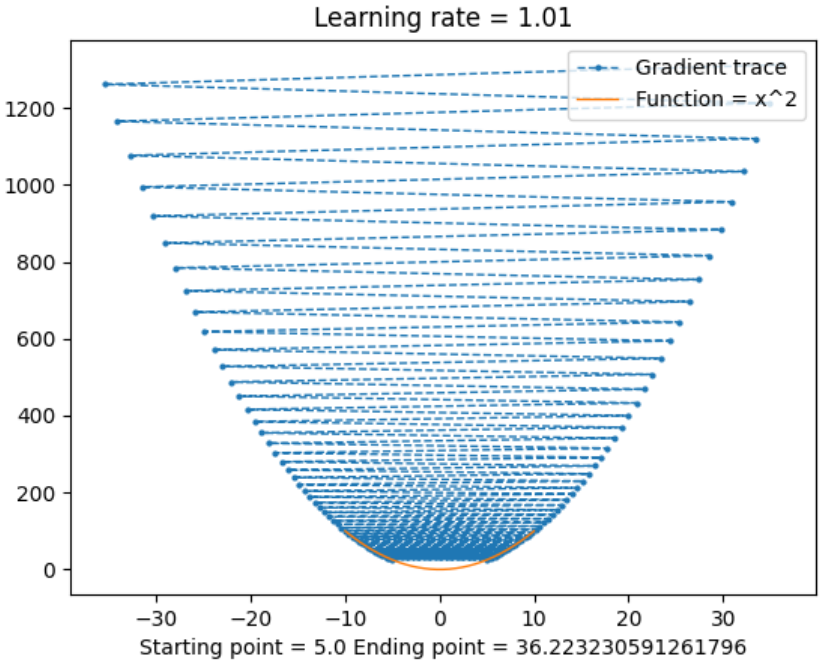




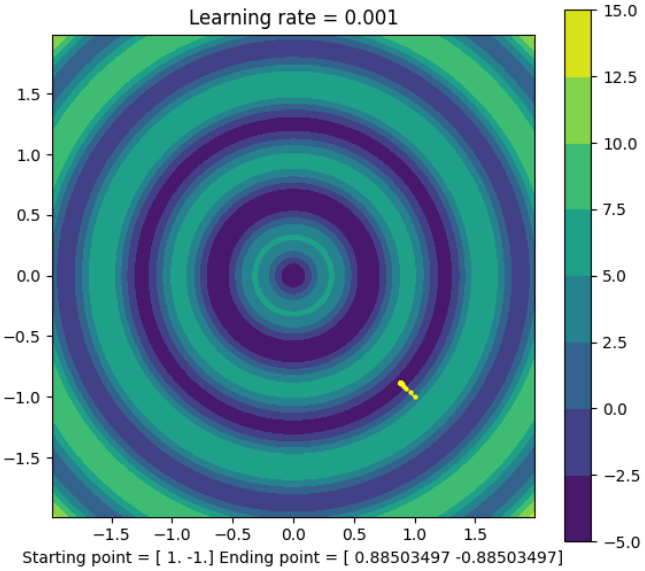


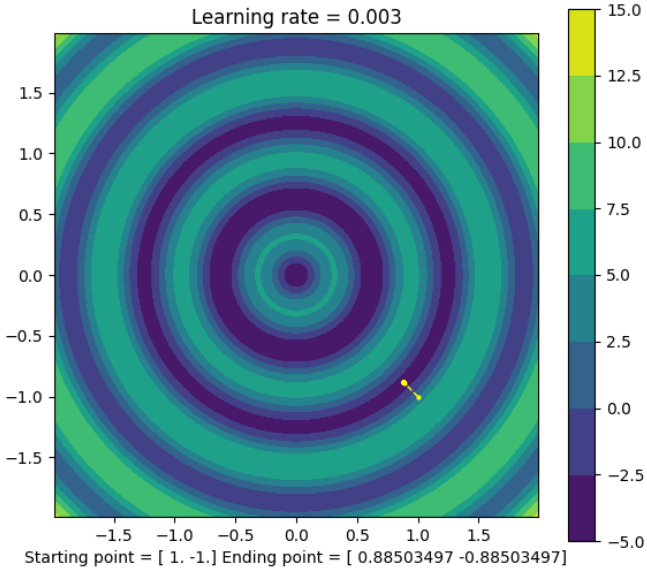


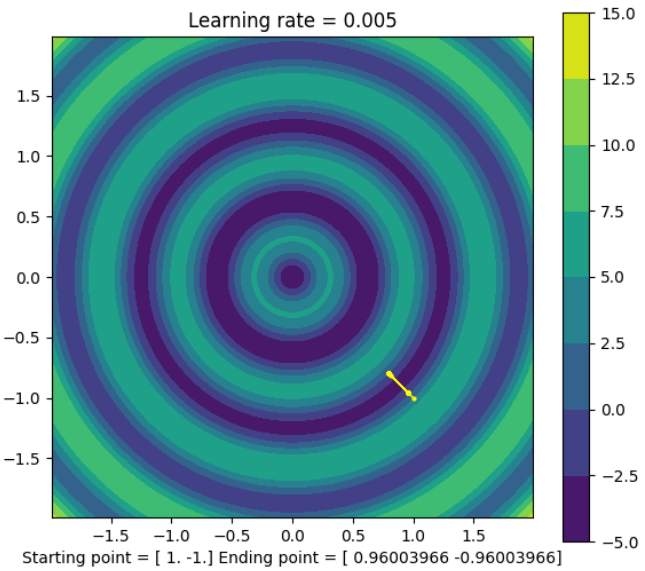
W następnych czterech przypadkach, dla kroku równego kolejno 0.03, 0.1, 0.3, 0.9 Widzimy, że funkcja gradientu odnajduje minimum lokalne funkcji, które jest również minimum globalnym. Warto zauważyć, że dla kroku równego 0.9 gradient robi niebezpiecznie duże kroki (jeżeli w pierwszym kroku oddaliłby się za punk –5.0 to funkcja gradientu dążyłaby do nieskończoności). Taka właśnie sytuacja wystąpiła po ustawieniu kroku na 1.01, co przedstawia wykres poniżej.

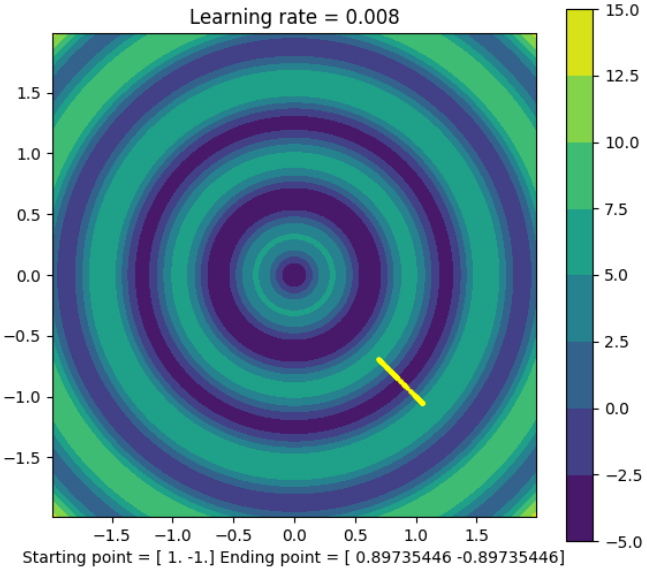


Następnie przeprowadziłem podobny test dla funkcji 3D z problemu numer 2.

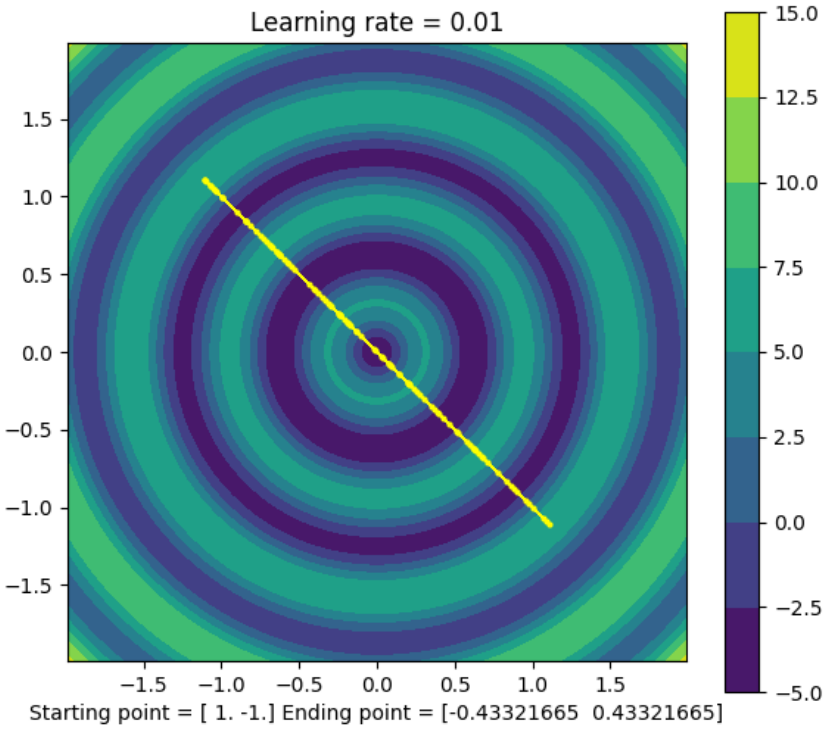






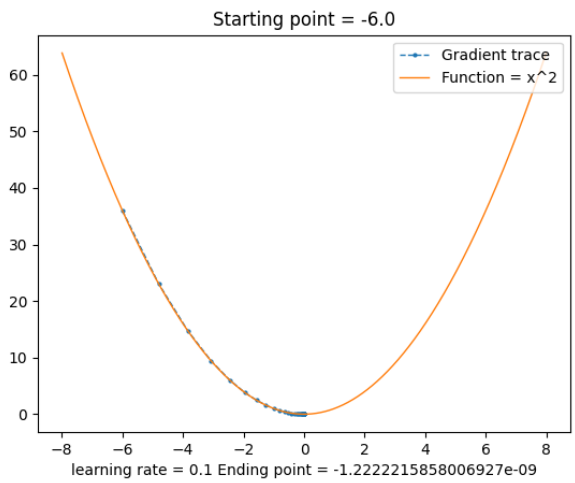


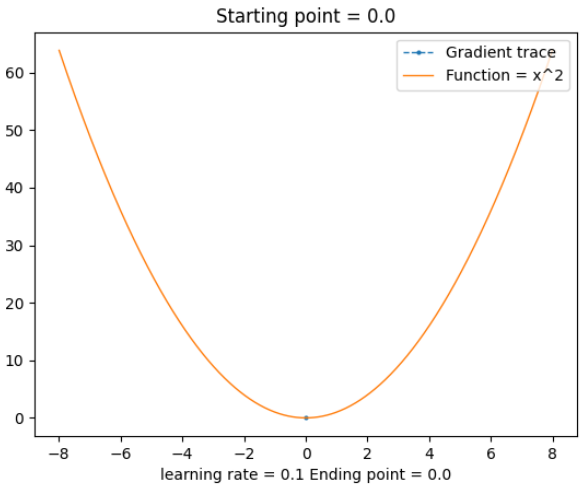
W powyższych 4 przypadkach możemy zauważyć, że funkcji gradientu udaje się dotrzeć do minimum lokalnego funkcji, (mimo, że w przypadku kroku równego 0.005 oraz 0.008 sytuacja wygląda podobnie, gdy w funkcji 2D krok wynosił 0.9, czyli funkcja przeskakuje na drugą stronę zbocza, ale trochę niżej dzięki czemu udaje jej się dojść do minimum.

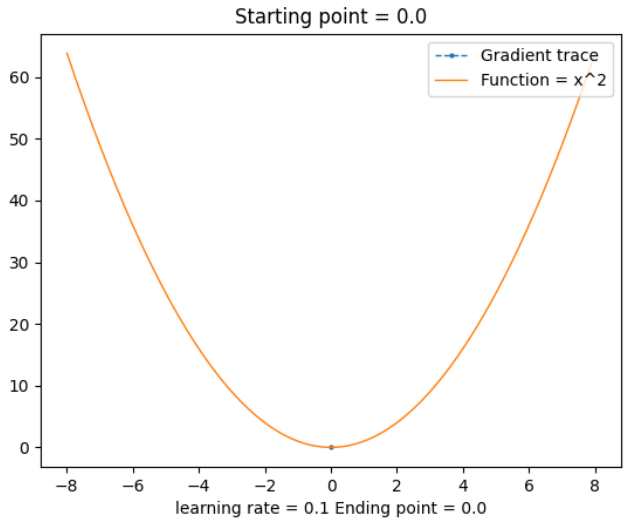


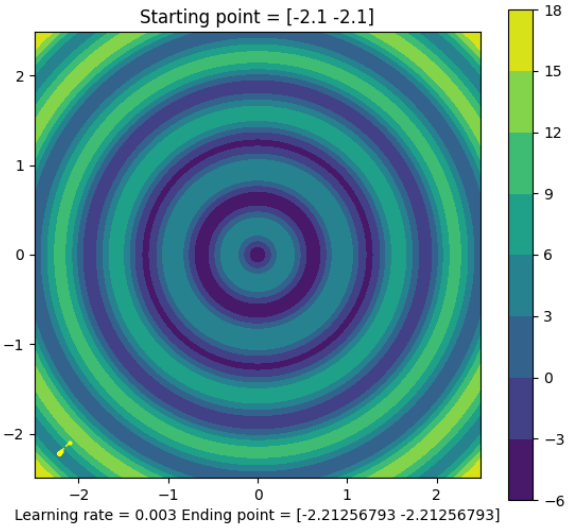
Natomiast w przypadku, gdy wartość kroku osiąga wartość 0.01 funkcja gradientu wariuje trafiając tak naprawdę w losowe miejsca funkcji.

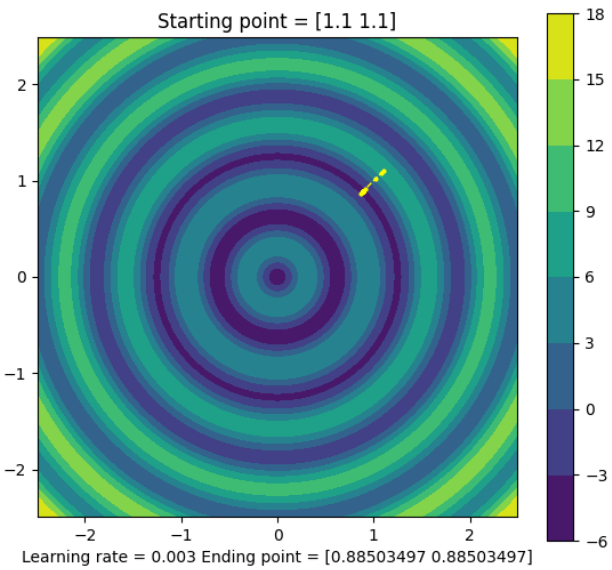
Na podstawie tych obserwacji uznałem, że najlepszym współczynnikiem alpha (wielkością kroku) dla funkcji 2D i 3D są kolejno: 0.1 i 0.003. To właśnie te współczynniki wykorzystałem przy sprawdzeniu jak zachowuje się funkcja gradientu prostego dla różnych punktów startowych. Poniżej zamieszczam wykresy pokazujące trasę gradientu dla obydwu problemów

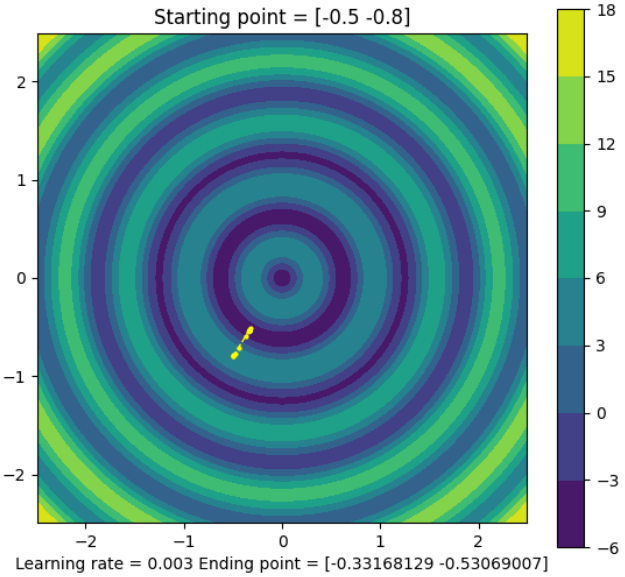


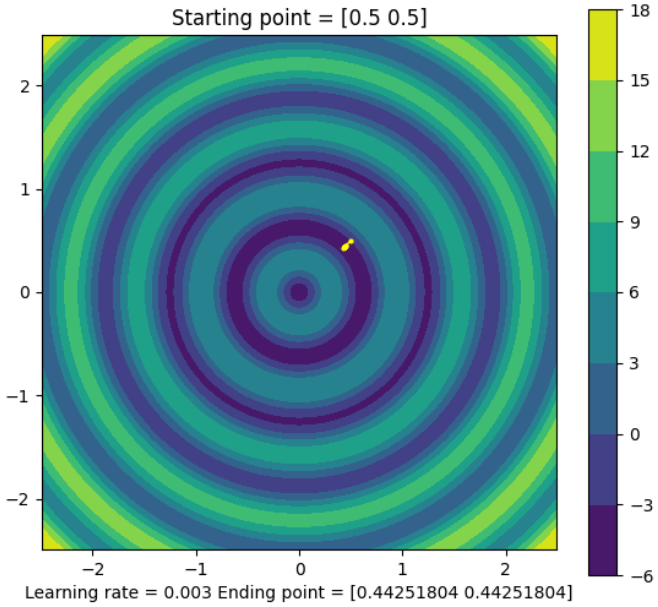



Starting point = 0.0
Gradient trace
Function = x^2
learning rate = 0.1 Ending point = 0.0


Starting point = [-2.1 -2.1]
Learning rate = 0.003 Ending point = [-2.21256793 -2.21256793]

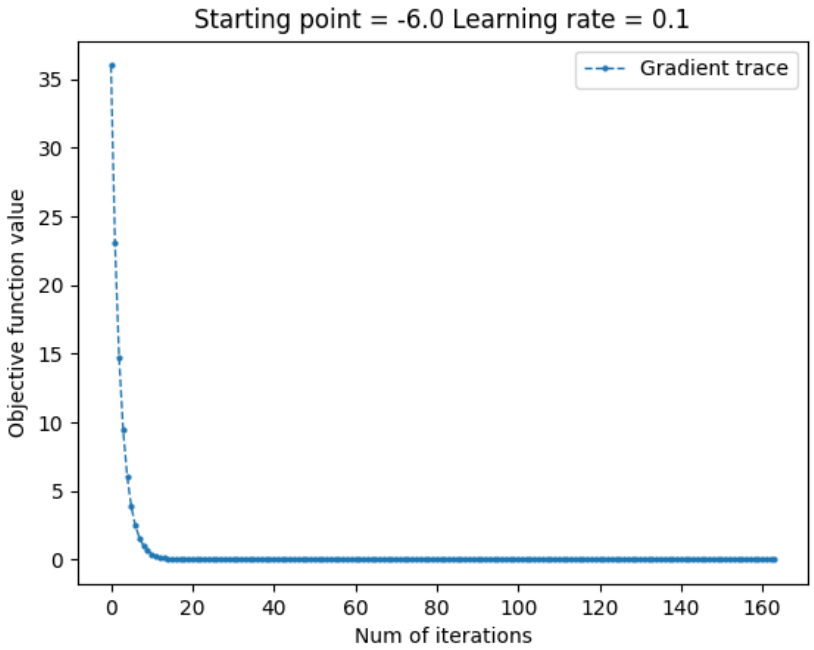

Starting point = [1.1 1.1]
Learning rate = 0.003 Ending point = [0.88503497 0.88503497]


Starting point = [-0.5 -0.8]
Learning rate = 0.003 Ending point = [-0.33168129 -0.53069007]



Jak widać na załączonych wykresach niezależnie od punktu startowego funkcja gradientu prostego znajdowała minimum lokalne funkcji, zarówno 2D jak i 3D.

Na końcu postanowiłem przyjrzeć się wartości funkcji celu w funkcji liczby kroków algorytmu.
W tym celu posłużyłem się funkcją celu z problemu pierwszego dla punktu startowego równego -6.0, wielkości kroku równej 0.1 oraz zaznaczyłem, że algorytm ma mieć 10000 iteracji (bądź zakończyć się jeżeli krok będzie mniejszy od epsilonu). W wyniku tej operacji otrzymałem taki oto wykres:



Widać na nim wyraźnie, że funkcja jest praktycznie płaska zaledwie po 15 iteracjach. To znaczy ze znajduje ona minimum funkcji bardzo szybko. Na wykresie widać jedynie 160 iteracji, ponieważ po około tylu iteracjach krok był na tyle mały, że algorytm przerwał swoje działanie.