Algorytm gradientu prostego

Celem ćwiczenia było zaimplementowanie algorytmu gradientu protego w języku python. Algorytm ten wyznacza minimum lokalne różnowymiarowych funkcji. Minimum te może być globalne, ale ze względu na specyfikę algorytmu nie ma na to gwarancji. Dla analizy działania algorytmu wykorzystam funkcje:

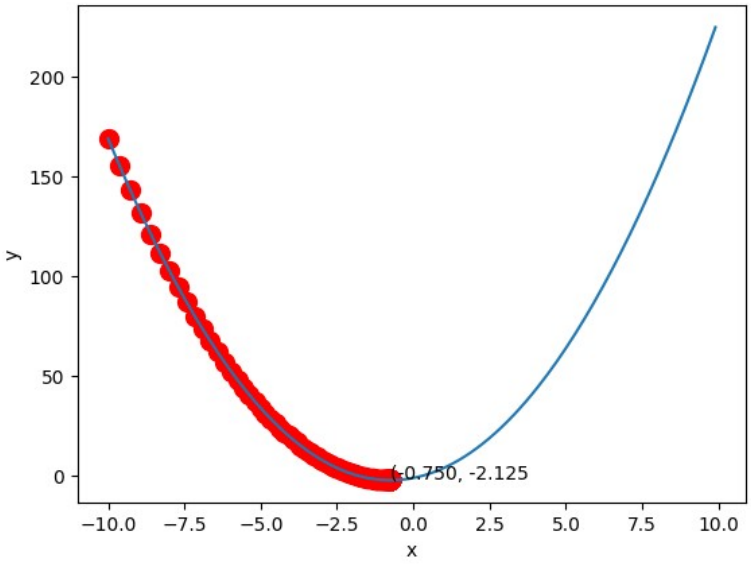
f(x) = 2x^2 + 3x − 1

g(x) = 1 − 0.6 exp {−x1^2 − x2^2} − 0.4 exp {−(x1 + 1.75)^2 − (x2 − 1)^2}

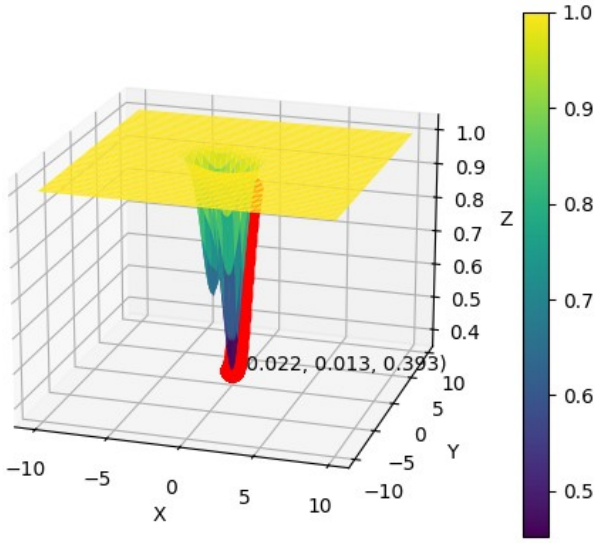
Sposób działania algorytmu

Podstawą działania algorytmu jest iteracyjne wyznaczanie nowych punktów zbliżających się do minimum lokalnego funkcji, wykorzystując do tego kierunek wektora gradientu. Pętla wykonuje się do momentu spełnienia warunku stopu, który w mojej implementacji przyjęłam jako odpowiednio małą długość wektora gradientu w znalezionym punkcie (długość musi być mniejsza niż jakaś dokładność epsilon). Dzięki temu możemy wyznaczyć minimum z zadaną dokładnością.

Sposób działania algorytmu obrazują poniższe wykresy, na których czerwonym kolorem zaznaczono kolejne punkty wyznaczane przez algorytm.



Ten wykres to wizualizacja działania algorytmu dla funkcji f(x) wspomnianej we wstępie. Jak widać, algorytm zaczął swoje działanie dla punktu początkowego x = -10 i przez kolejne iteracje wyznaczał punkty zbliżające się do minimum funkcji. Podobna sytuację można zobaczyć na wykresie dla funkcji g(x), który przedstawiam poniżej. Tutaj punkt początkowy wynosił (x,y) = (1,1).



Opis eksperymentów

Zbadam w jaki sposób czas oraz dokładność wyniku działania algorytmu zależą od wykorzystywanych przez niego wartości (hiperparametrów):

- punktu początkowego wykonywania się algorytmu

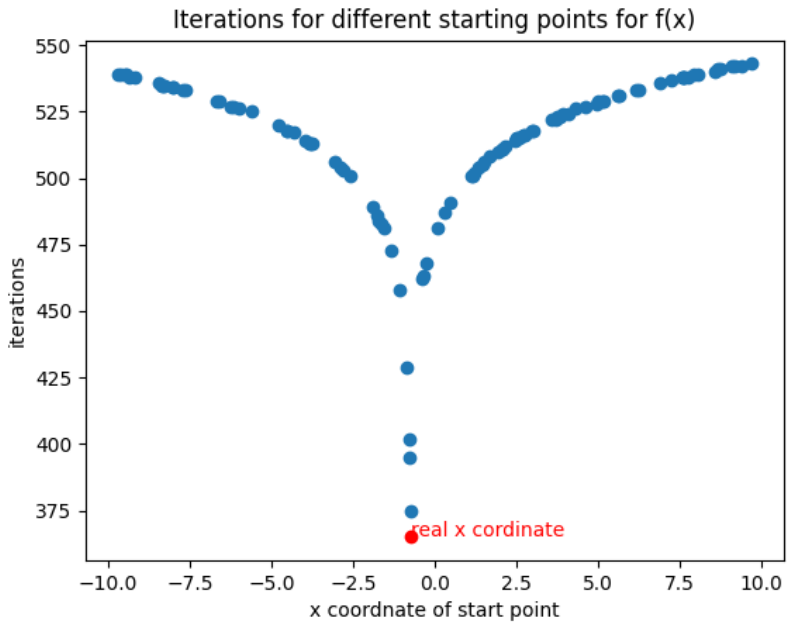
- kroku, czyli współczynnika, przez który mnożony jest gradient

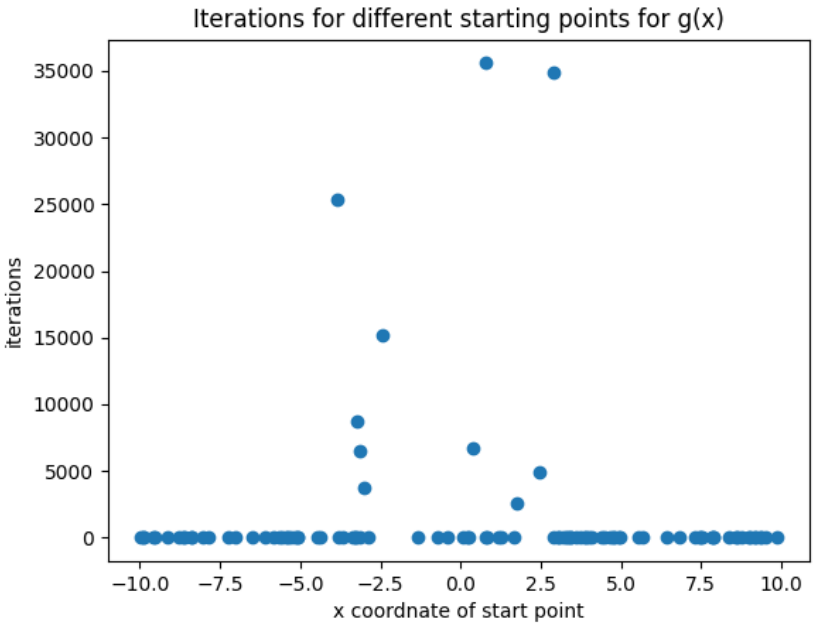
- epsilona, czyli dokładności warunku stopu

Wpływ punktu początkowego

Wyniki

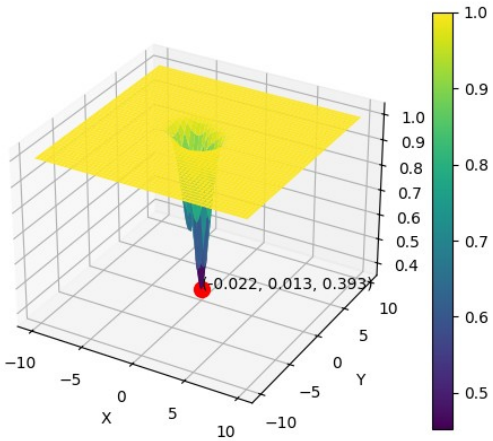
Wpływ punktu początkowego – liczba iteracji algorytmu:



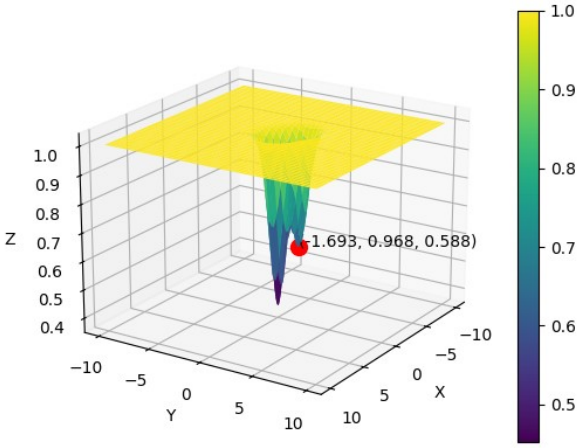


Wpływ punktu początkowego – wyznaczone minimum:

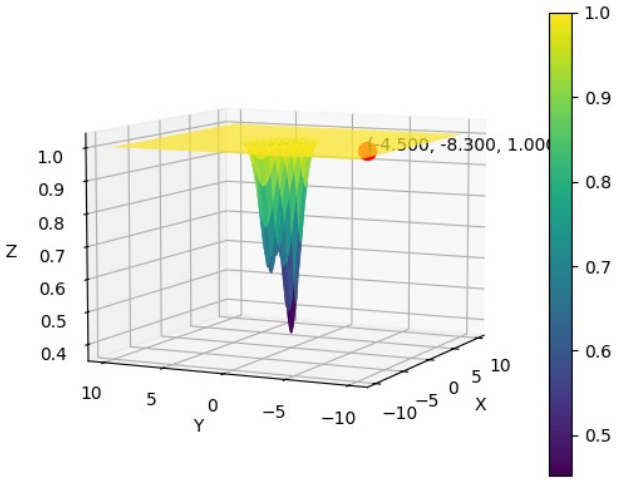
- dla (x0, y0) = (1, 1) → minimum globalne (x,y,z) = ~(-0.0219, 0.0125, 0.3927)



- dla (x0, y0) = (-2, 2) → minimum lokalne (x,y,z) = ~(-1.6943, 0.9685, 0.5883)



- dla (x0, y0) = (-4.5, -8.3) → dokładnie ten sam punkt (x,y,z) = (-4.5, -8.3, 1)



Wnioski

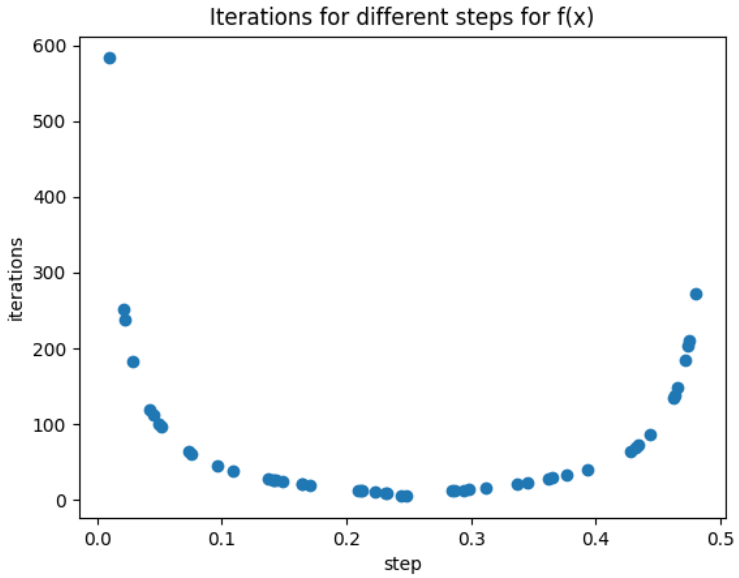
Po pierwsze, wybór punktu początkowego ma wpływ na szybkość wykonania algorytmu. Jeśli punkt początkowy znajduje się blisko minimum, algorytm zakończy swoje działanie bardzo szybko (będzie miało iteracji), jak widać to na wykresie pierwszym. Wykres 2 pokazuje, że w przypadku funkcji g(x) dla większości punktów startowych liczba iteracji wynosi 0. Jest tak, ponieważ wykres tej funkcji ma dużą ilość płaskiej przestrzeni, dla której gradient wynosi 0, przez co znaleziony przez algorytm punkt jest taki sam, jak punkt początkowy.

Od wartości punktu startowego zależy także to, które minimum zostanie znalezione (będzie to najbliższe minimum po malejącej stronie punktu). Pokazane zostało to na 3 ostatnich wykresach, gdzie czerwonym punktem zaznaczono znalezione minimum.

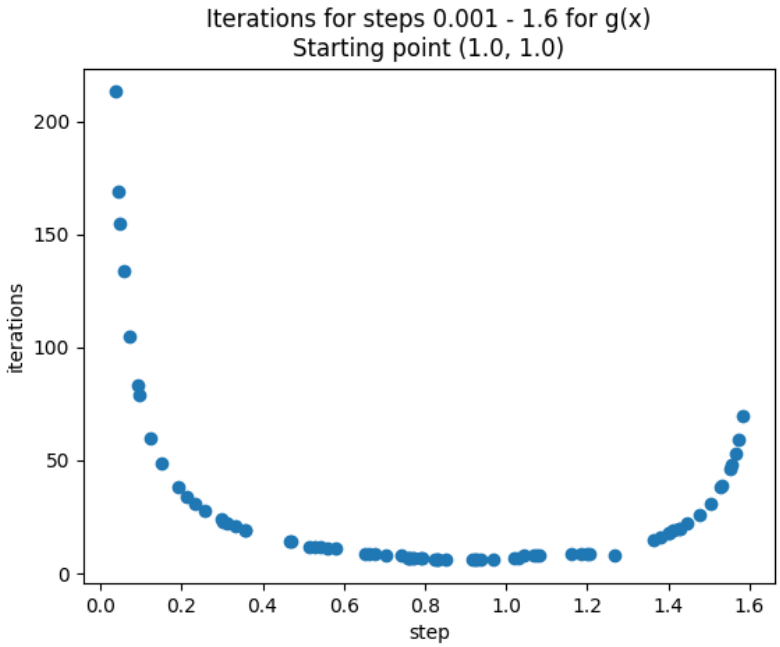
Wpływ kroku na wynik działania algorytmu

Wyniki

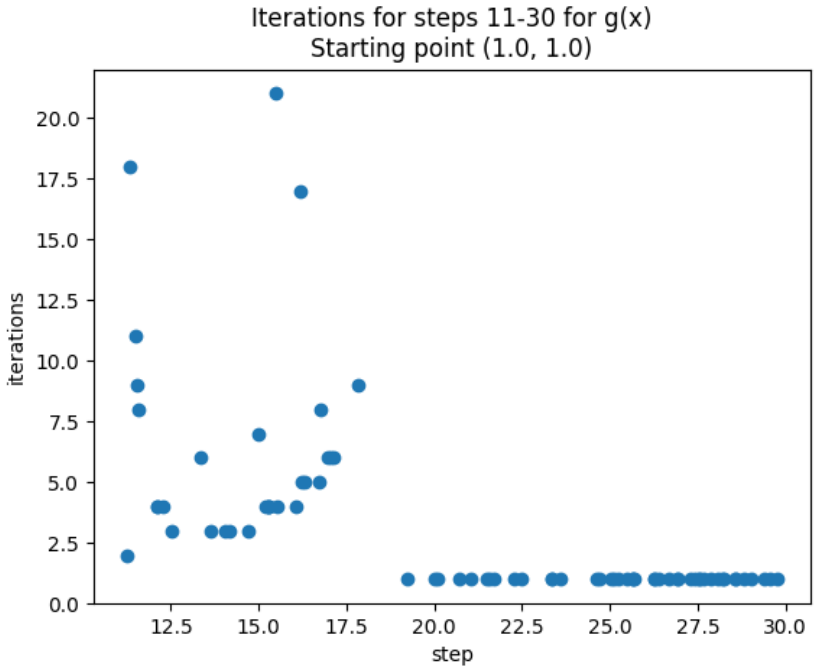
Wpływ kroku– liczba iteracji algorytmu:



Dla kroku >=0.5 dla f(x) program przestaje działać (zapętlna się w nieskończoność).

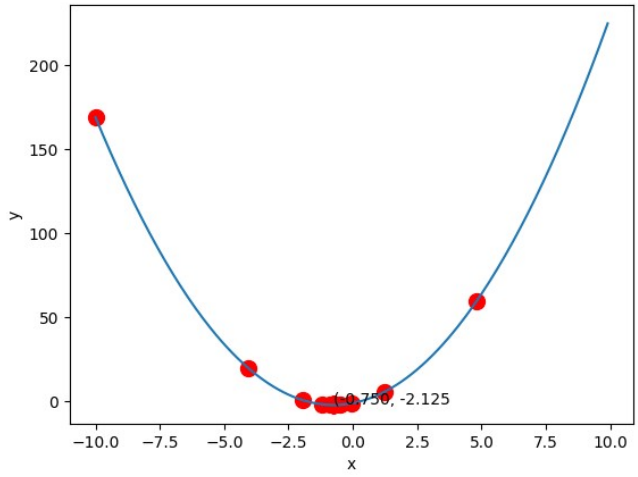


Dla kroków 1.7 – 10 dla g(x) przy punkcie początkowym (1.0, 1.0) algorytm zapętla się w nieskończoność

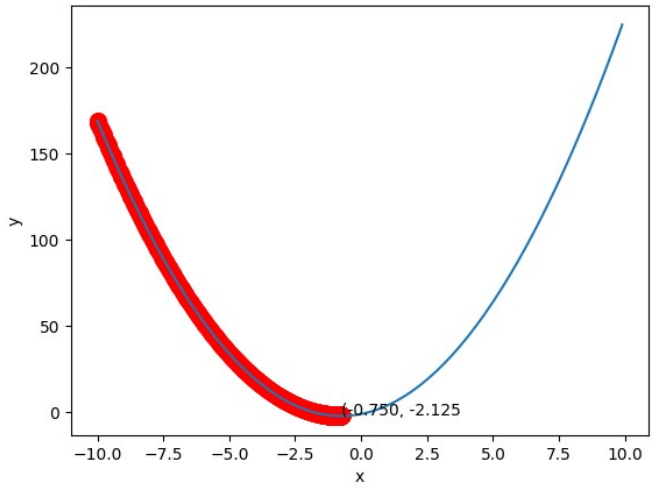


Wpływ kroku – sposób wyznaczania punktów:

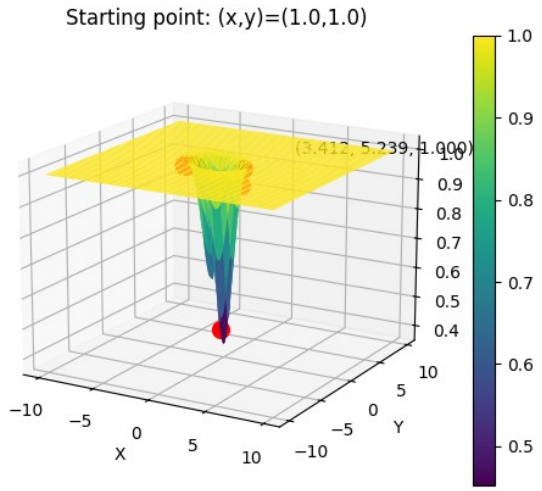
Oto jak zachowuje się algorytm wyznaczania minimum dla f(x) przy punkcie początkowym x=-10 i kroku 0.4. Widać, że kolejne punkty skaczą od lewej strony minimum do prawej, jednak wciąż udaje im się dotrzeć do minimum.



Tak natomiast wygląda przebieg algorytmu dla punktu początkowego x=-10 i kroku 0.001. Czas wykonania jest w tym przypadku dużo dłuższy, a na wykresie widać, jak gęsto rozmieszczone są kolejno znajdowane punkty.



Oto jak wygląda przebieg algorytmu dla (x0, y0) = (1.0, 1.0) i kroku 20 dla funkcji g(x):



Wnioski

Krok to współczynnik, przez który mnożony jest gradient i razem z długością wektora gradientu odpowiada za to, o ile przesunięty względem poprzedniego punktu będzie następny wyznaczony punkt. Dla różnych funkcji i różnych punktów początkowych lepsze są inne wartości kroku.

Mała wartość kroku sprawi, że algorytm będzie wykonywać się bardzo długo, ponieważ kolejne kroki będą wyznaczane bardzo blisko siebie. Czas rośnie także dla tych większych wartości kroku, ponieważ nowo wyznaczane punkty dla takich kroków bardzo długo “skaczą” dookoła minimum. Optymalne są wartości ze środka przedziału. Widać to na wykresach liczby iteracji.

Zbyt duży krok niesie ze sobą niebezpieczeństwo nieskończonego zapętlenia się algorytmu. Nowo wyznaczane punkty będą wtedy skakać po całej dziedzinie wykresu, przeskakując jakiekolwiek minima lub znajdzie się taki punkt, dla którego następny krok będzie przenosić nas naprzemiennie do dwóch, ciągle tych samych punktów po przeciwnych stronach minimum.

Dla funkcji f(x) zapętlenie ma miejsce przy kroku 0.5, a najbardziej optymalną wartością kroku wydaje się być około 0.25.

W przypadku funckji g(x), działanie kroku w dużej mierze zależy od punktu początkowego. Dla punktów z “płaskiej” częscie wykresu, nie ma on oczywiście żadnego znaczenia. Na wykresach wyżej widać zależności dla punktu (1,1). Dla niego, program działa dla kroków <= 1.6 (optymalny ktok to około 0.9). Dla przedziału 1.7-10 pojawia się nieskończone wykonywanie pętli, a dla kroków >=10, któryś z wyznaczanych przez algorytm następnych punktów jest “wyrzucany” w strefę, w której wykres funkcji jest płaski, zatem, ponieważ gradient wynosi 0, program zostaje zakończony i jego wynikiem jest punkt z tej poziomo płaskiej części wykresu. Widać to dobrze na ostatnim wykresie.

Wpływ dokładności epsilon na wynik działania programu

Wyniki

Zależność dokładności wyników od eps dla f(x)

eps	x	y
0.01	-0.7475552625339865	2.124988046517444
0.001	-0.7497514457013887	-2.1249998764415214
0.0001	-0.7499757405158598	-2.124999998822955
0.00001	-0.749997533559677	-2.1249999999878337
0.000001	-0.7499997592696235	- 2.124999999999884
0.0000001	-0.7499999755251554	-2.1249999999999987
0.00000001	-0.7499999975116642	-2.125
0.000000001	-0.7499999997571325	-2.125
0.0000000001	-0.7499999999753076	-2.125

Zależność dokładności wyników od eps dla f(x)

eps	x	y	z
0.01	-0.015198907319471926	0.017360206927848783	0.39280849429103143
0.001	-0.021224972056833637	0.012959199090407648	0.3927678810164442
0.00001	-0.02194006561904842	0.012544343937757626	0.3927674684527159
0.000001	-0.021947395267844498	0.012542006128560395	0.3927674684108275
0.00000001	-0.021948240310213914	0. 012541856238569684	0. 3927674684103984
0.0000000001	-0. 02194824914844751	0. 012541856686420487	0. 3927674684103984

Wnioski

Im mniejszy jest epsilon, tym bardziej dokładny jest wynik. Jednak wraz ze spadkiem wartości epsilona rośnie czas wykonywania się programu. Szczególnie zauważalne jest to dla bardziej skomplikowanej funkcji, jaką jest funkcja g(x). Przykładowo, dla eps = 0.00000000000001 czas działania programu nagle bardzo mocno się wydłuża.