Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Wstęp do sztucznej inteligencji

Raport z laboratorium 1.

Kacper Bugała

Warszawa, 2022Z

1. Opis rozwiązania

Zaimplementowano algorytm gradientu prostego, który skutecznie rozwiązuje problemy znajdowania minimum funkcji jednowymiarowych i dwuwymiarowych. Eksperymenty prowadzono na jednowymiarowej funkcji (f(x)), oraz dwuwymiarowej funkcji g(x_1,x_2), o wzorach widocznych w rys.1.1:

$$f(x) = \frac{1}{4}x^4$$

$$g(x) = 2 - \exp\left\{-x_1^2 - x_2^2\right\} - 0.5\exp\left\{-(x_1+1.5)^2 - (x_2-2)^2\right\}$$

$$\nabla f(x) = x^3$$

$$\nabla g(x) = \begin{bmatrix} 2x_1\exp\left\{-x_1^2 - x_2^2\right\} + (x_1+1.5)\exp\left\{-(x_1+1.5)^2 - (x_2-2)^2\right\} \\ 2x_2\exp\left\{-x_1^2 - x_2^2\right\} + (x_2-2)\exp\left\{-(x_1+1.5)^2 - (x_2-2)^2\right\} \end{bmatrix}$$

Rys. 1.1. Wzory funkcji

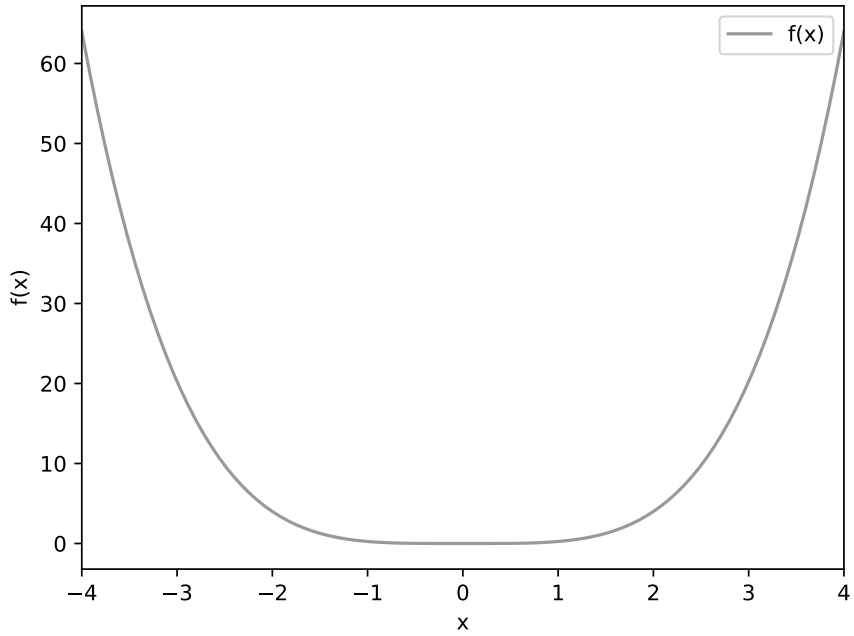
W wyniku eksperymentów, najlepsze jakościowo i wydajnościowo wyniki otrzymano dla ustalonych wartości hiperparametrów algorytmu:

> Maksymalna liczba iteracji – 10000
> Wartość epsilonu (warunek 'STOP') – 0.0001
> Wartość współczynnika zanikania kroku – 0.03

Przeprowadzono eksperymenty, w celu zbadania wpływu zmiany długości kroków dla różnych punktów początkowych na wyniki algorytmu.

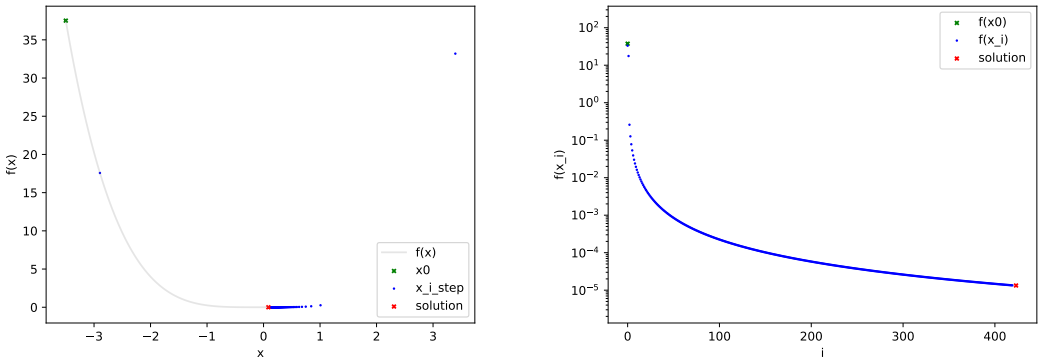
2. Funkcja jednowymiarowa *f(x)*

Funkcja f posiada jedno minimum lokalne, co sprawia, że algorytm z pewnością nie zatrzyma się w minimum lokalnym, które nie jest minimum globalnym. Wykres funkcji *f(x)* przedstawiono na rys. 2.1

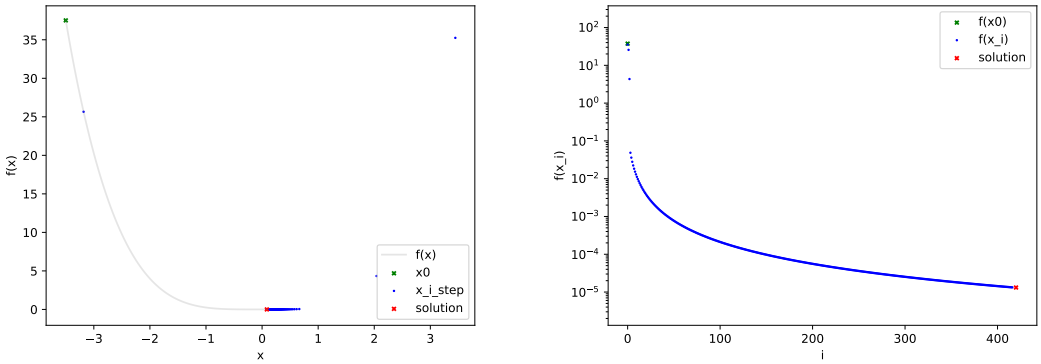


Rys. 2.1. *f(x)*

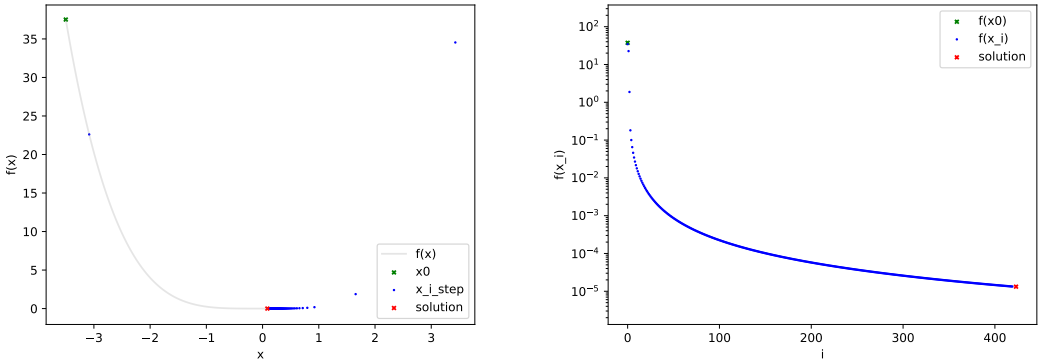
Na rys. 2.2-2.6 przedstawiono kolejne iteracje algorytmu dla różnych długości kroku, przy punkcie początkowym x=*-3.5*.



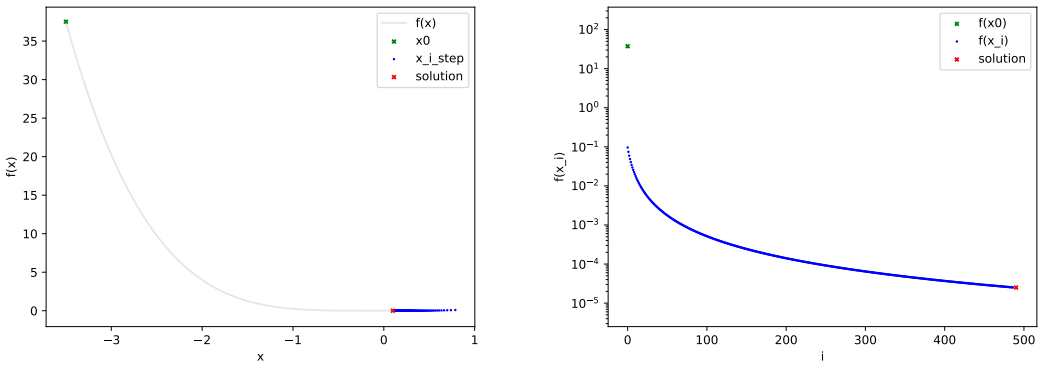
Rys. 2.2. *f(x)* dla punktu początkowego x= *-3.5*, oraz długości kroku 1.0; l. iteracji: 423



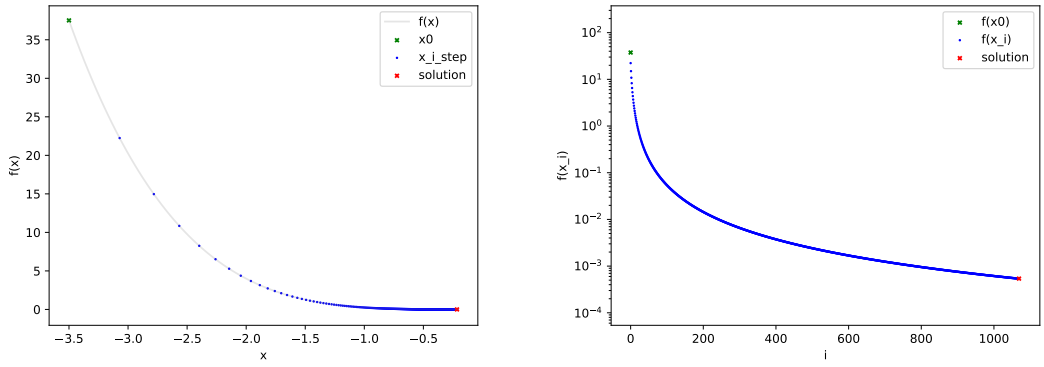
Rys. 2.3. *f(x)* dla punktu początkowego *x*= *-3.5*, oraz długości kroku 0.5; l. iteracji: 420



Rys. 2.4. *f(x)* dla punktu początkowego *x*= *-3.5*, oraz długości kroku 0.2; l. iteracji: 423

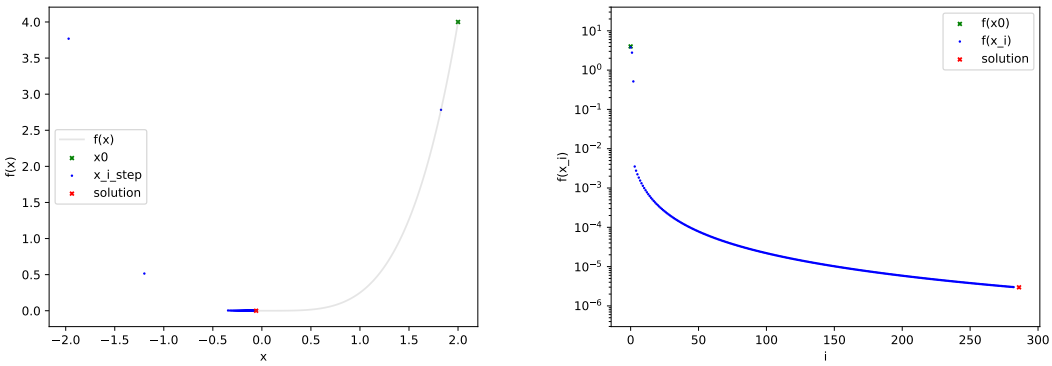


Rys. 2.5. *f(x)* dla punktu początkowego *x*= *-3.5*, oraz długości kroku 0.1; l. iteracji: 490

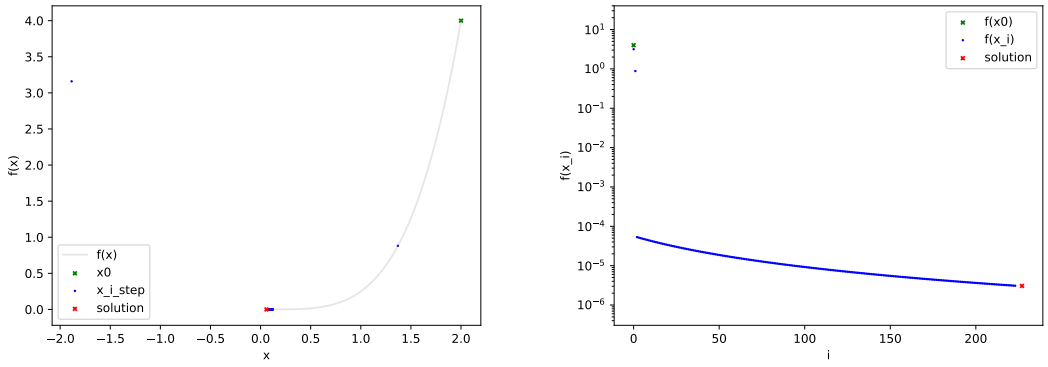


Rys. 2.6. *f(x)* dla punktu początkowego *x= -3.5*, oraz długości kroku 0.01; l. iteracji: 1069

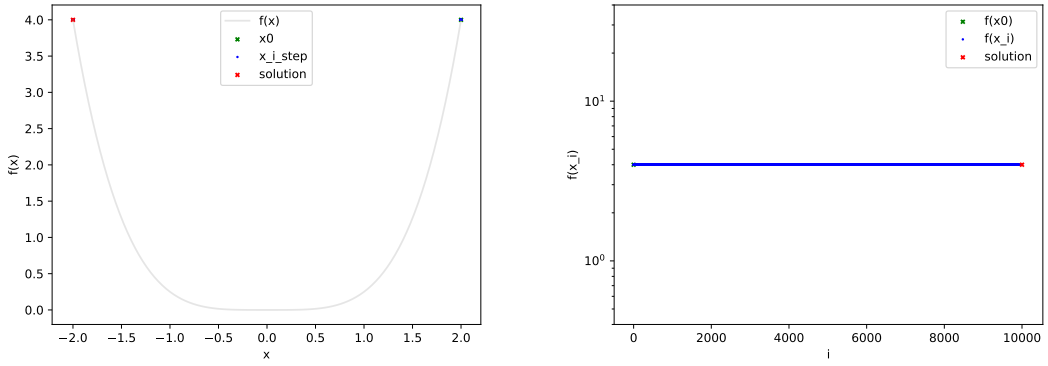
Jak widać, każde z podejść (rys. 2.2-2.6) zakończyło się pomyślnie, znajdując rozwiązanie rzędu e-5. Na wykresach można zauważyć, że na każdym z rys. 2.2-2.5 pierwszy krok miał na tyle duży gradient, że przy tak obranej długości kroku "przeszacował" minimum globalne, wobec czego kolejne iteracje znajdują się już na półosi dodatniej. Dopiero zmniejszenie długości kroku do *0.01* (rys. 2.6)sprawiło, że dla punktu początkowego *x=-3.5*, wszystkie kolejne iteracje wyznaczają punkty coraz bliższe zeru nie "przeskakując" przez nie. Analizując liczbę iteracji dla każdej z długości kroku można zauważyć, że najszybciej algorytm przeprowadzony został po wybraniu długości kroku *0.5*. Potwierdza to teorię, która mówi, że nie ma uniwersalnego podejścia (krok jak najmniejszy/jak największy) do optymalizacji obliczeń w algorytmie gradientu prostego. Dla wybranego punktu początkowego, algorytm spisał się najlepiej dla kroku równego *0.5*, co nie pokryło się z wynikami otrzymanymi w innych punktach początkowych. Jednym z ciekawszych wyników eksperymentu, są efekty obrania punktu początkowego *x=2.0*. Wykresy przedstawiono na rys.2.7-2.11



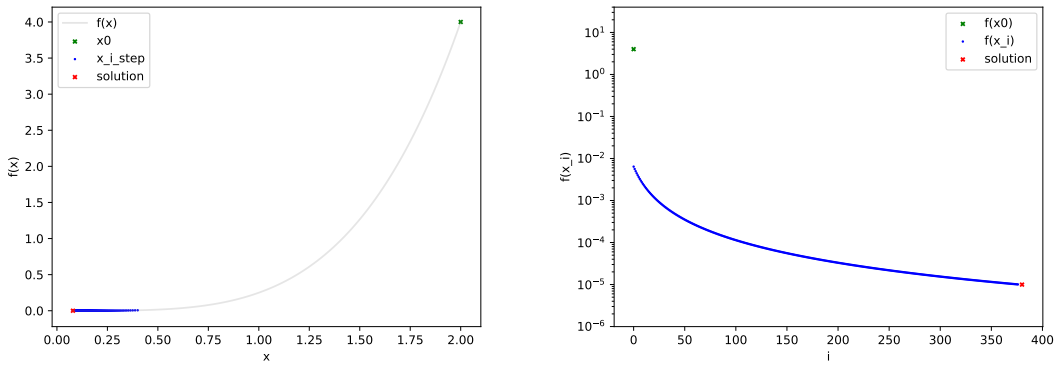
Rys. 2.7. *f(x)* dla punktu początkowego *x= 2.0*, oraz długości kroku 1.0; l. iteracji: 286



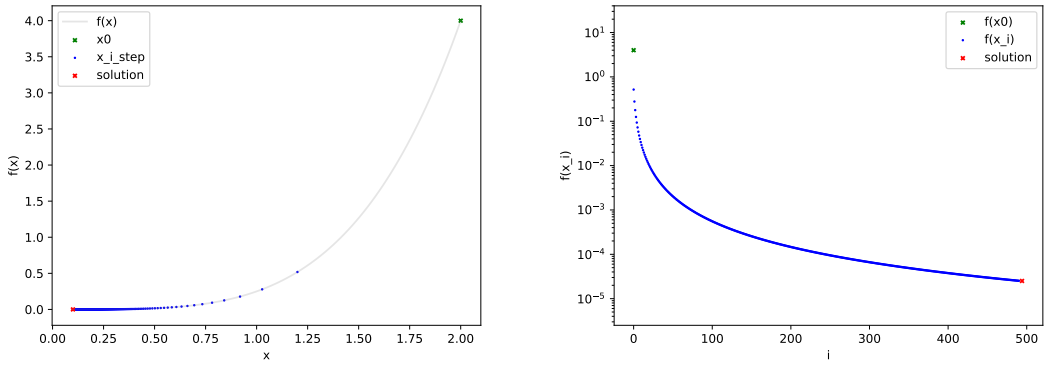
Rys. 2.8. *f(x)* dla punktu początkowego *x= 2.0*, oraz długości kroku 0.7; l. iteracji: 227



Rys. 2.9. *f(x)* dla punktu początkowego *x= 2.0*, oraz długości kroku 0.5; l. iteracji: 9999

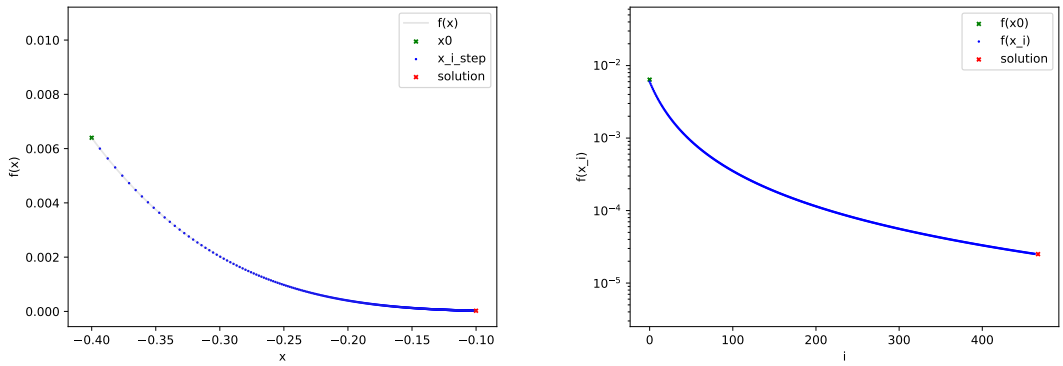


Rys. 2.10. *f(x)* dla punktu początkowego *x= 2.0*, oraz długości kroku 0.2; l. iteracji: 380

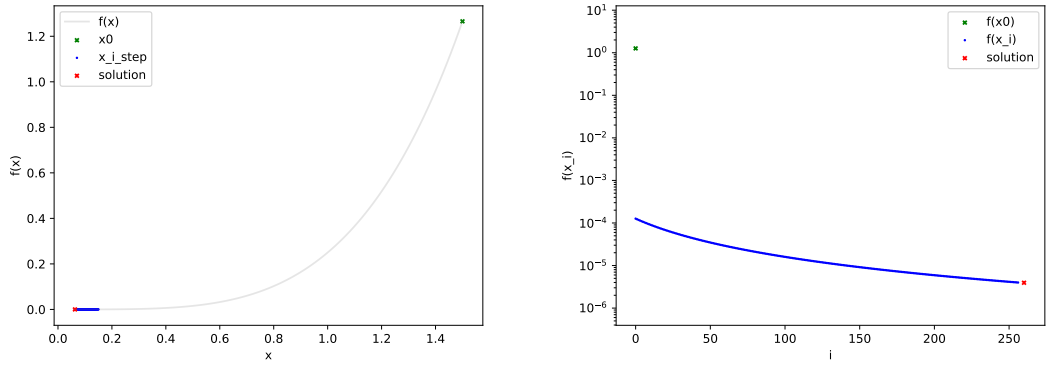


Rys. 2.11. *f(x)* dla punktu początkowego *x= 2.0*, oraz długości kroku 0.1; l. iteracji: 494

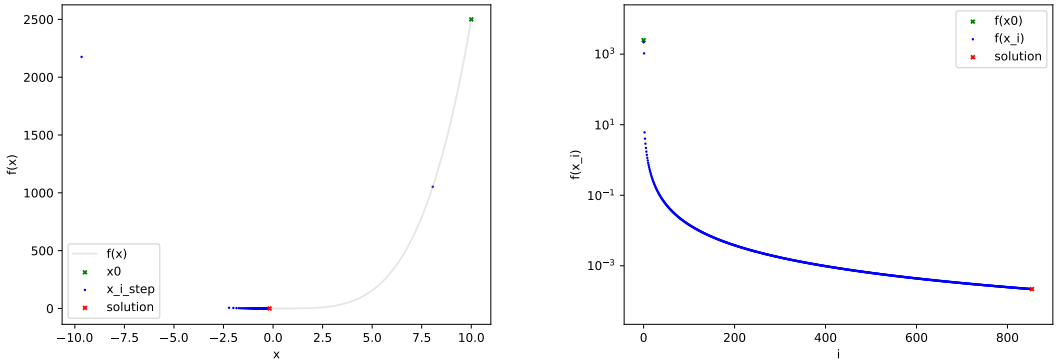
Po wybraniu punktu początkowego *x=2.0* pierwsze co można zauważyć to fakt, że dla długości kroku mniejszej niż 0.5 w algorytmie nie występuje już przeszacowanie. Liczba iteracji dla kroku równego 0.5 wynosi 9999, tj. maksymalną dopuszczalną. Algorytm nie zdołał znaleźć minimum lokalnego, ponieważ gradient i długość kroku w wybranym punkcie cały mają takie wartości, że punkt w kolejnych iteracjach zmienia się 2 -¿ -2, -2 -¿ 2. Widzimy, że optymalna długość dla punktu początkowego -3.5 okazuje się fatalna dla innego punktu początkowego. Na podstawie tego wyciągnąć można wnioski, że niemożliwe jest wybranie jednej, optymalnej długości kroku, niezależnie od obranego punktu początkowego.
Na rys.2.12-2.15 wybrane wyniki testów funkcji jednowymiarowej. Potwierdzają opisaną tezę.



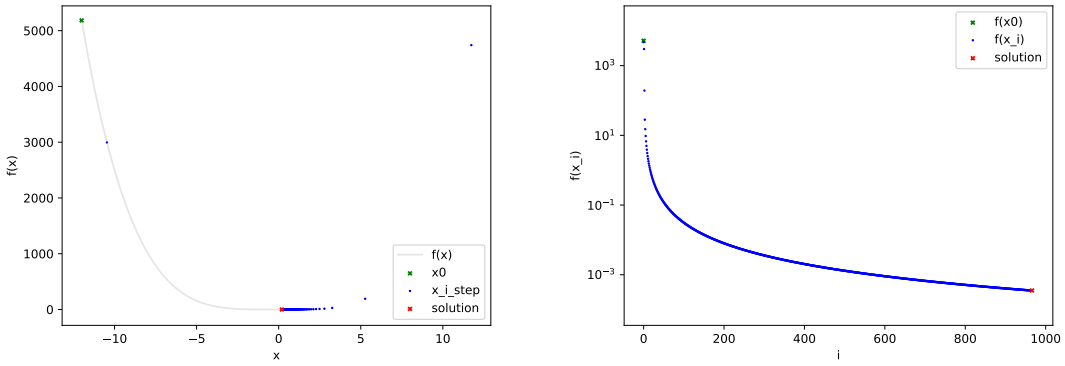
Rys. 2.12. *f(x)* dla punktu początkowego *x= -0.4*, oraz długości kroku 0.1; l. iteracji: 467



Rys. 2.13. *f(x)* dla punktu początkowego *x= 1.5*, oraz długości kroku 0.4; l. iteracji: 260



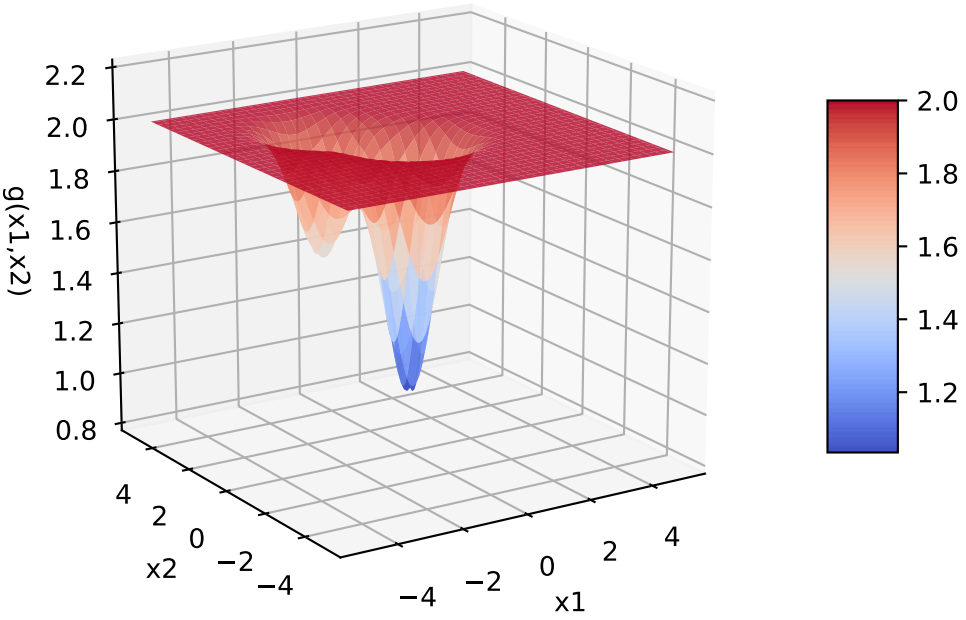
Rys. 2.14. *f(x)* dla punktu początkowego *x= 10.0*, oraz długości kroku 1.0; l. iteracji: 854



Rys. 2.15. *f(x)* dla punktu początkowego *x= -12.0*, oraz długości kroku 0.6; l. iteracji: 966

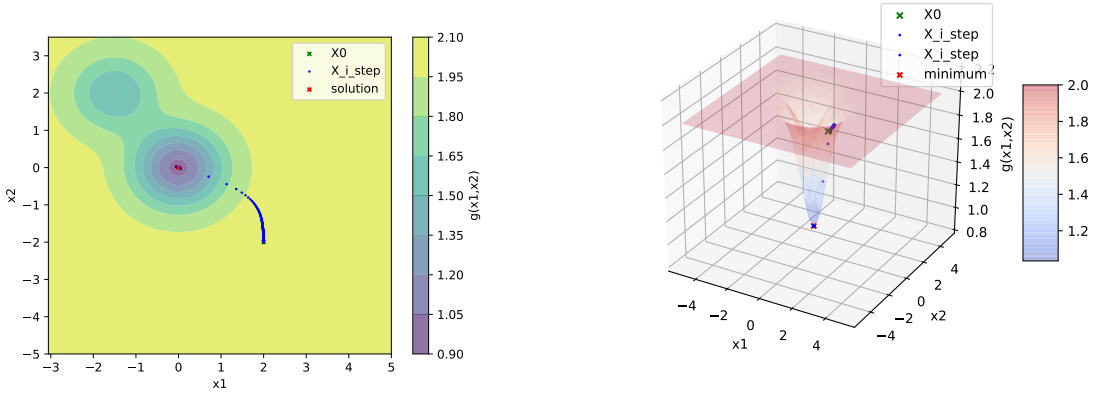
3. Funkcja wielowymiarowa $g(x_1, x_2)$

Funkcja g jest funkcją dwóch zmiennychm, która posiada dwa mininum lokalne, co oznacza, że przy nieodpowiednio dobranych parametrach algorytmu, warunek 'stop' może wystąpić w punkcie, który nie jest minimum globalnym. Wykres funkcji *g(x)* przedstawiono na rys. 3.1

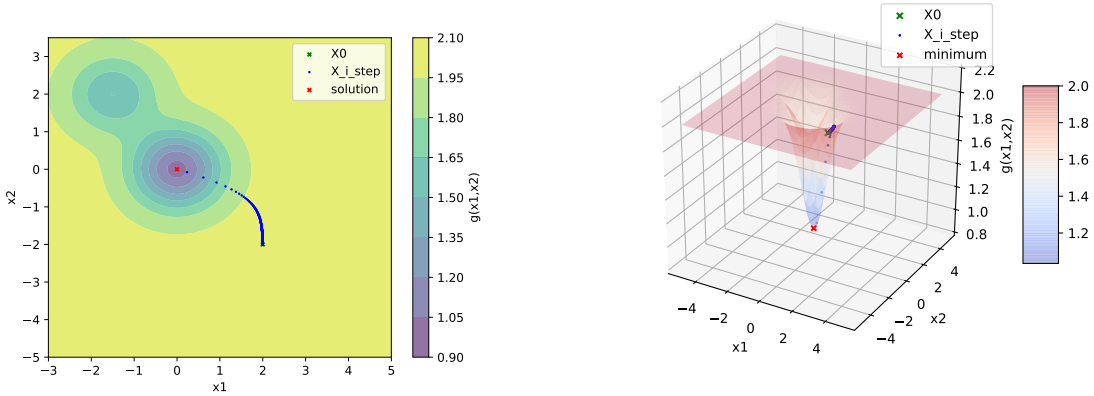


Rys. 3.1. Wykres funkcji *g(x1, x2)*

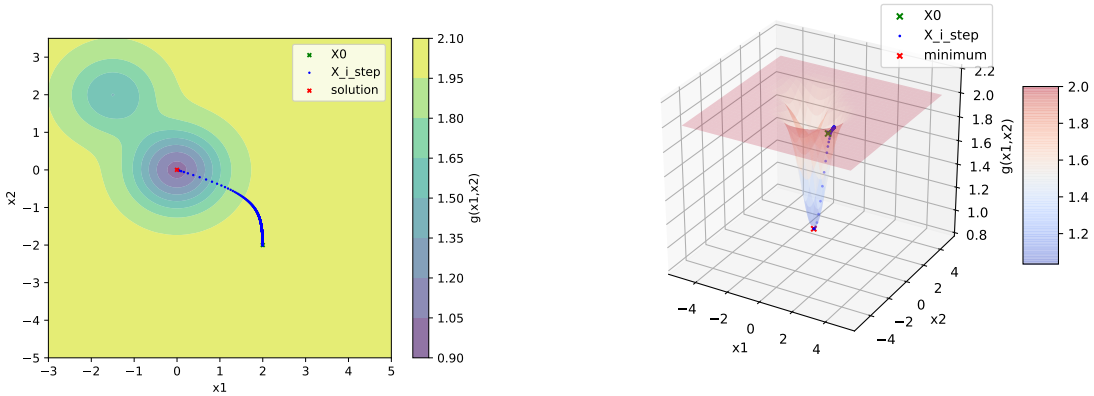
Rysunki 3.2-3.4 przedstawiają wyniki algorytmu dla różnej długości kroków, przy punkcie początkowym *X=[2, -2]*:



Rys. 3.2. *G(x_1,x_2) dla X = [2.0, -2.0] oraz długości kroku 1.0; l. iteracji: 440*

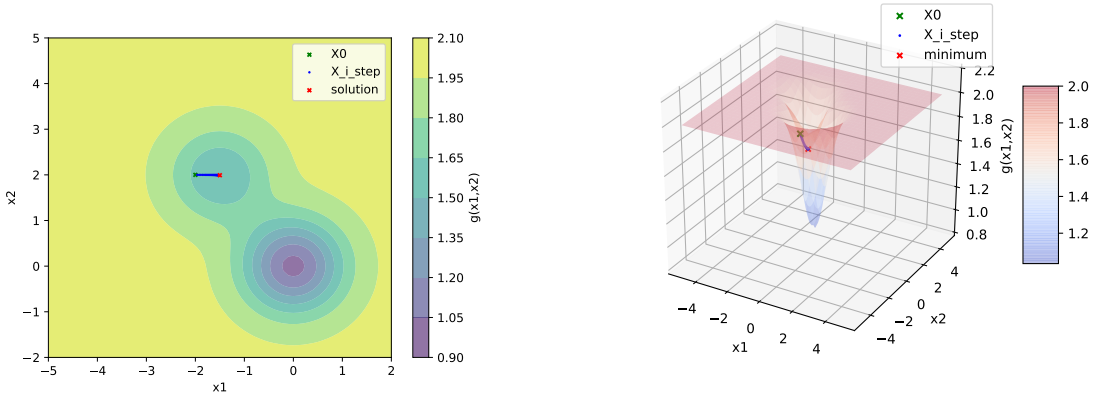


Rys. 3.3. *G(x_1,x_2) dla X = [2.0, -2.0] oraz długości kroku 0.5; l. iteracji: 469*

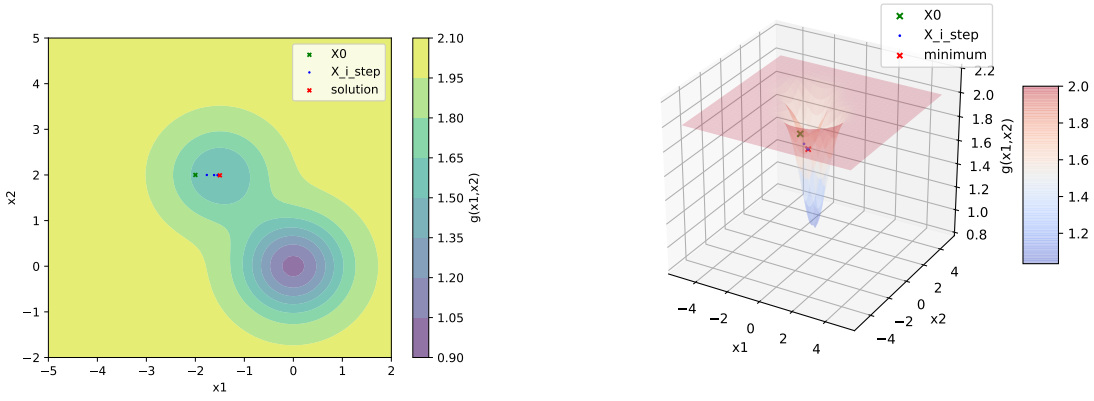


Rys. 3.4. *G(x_1,x_2) dla X = [2.0, -2.0] oraz długości kroku 0.2; l. iteracji: 1174*

Kolejne rysunki **??-??** to skutki wybrania punktu początkowego *X=[-2, 2]*



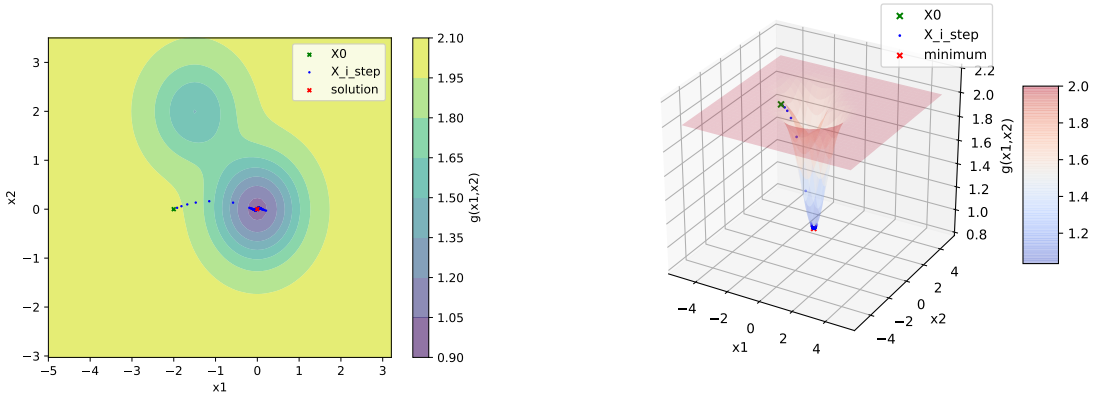
Rys. 3.5. *G(x_1,x_2) dla X = [-2.0, 2.0] oraz długości kroku 0.1; l. iteracji: 61*



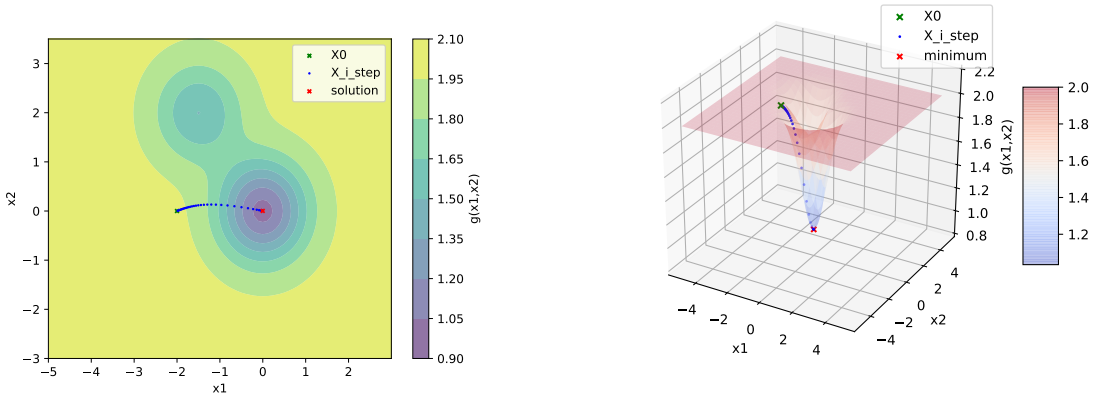
Rys. 3.6. *G(x_1,x_2) dla X = [-2.0, 2.0] oraz długości kroku 0.6; l. iteracji: 10*

Obierając nieprawidłowy punkt początkowy, istnieje skończone prawdopodobieństwo, że algorytm 'utknie' w minimum lokalnym. Taka sytuajca pojawiła się na wykresach 3.5 oraz 3.6. Widzimy zbieżność algorytmu do minimum lokalnego, w wyniku obrania punktu początkowego z problematycznym gradientem.

Kolejne wybrane wyniki przedstawione zostały poniżej, wraz z opisami doboru parametrów.

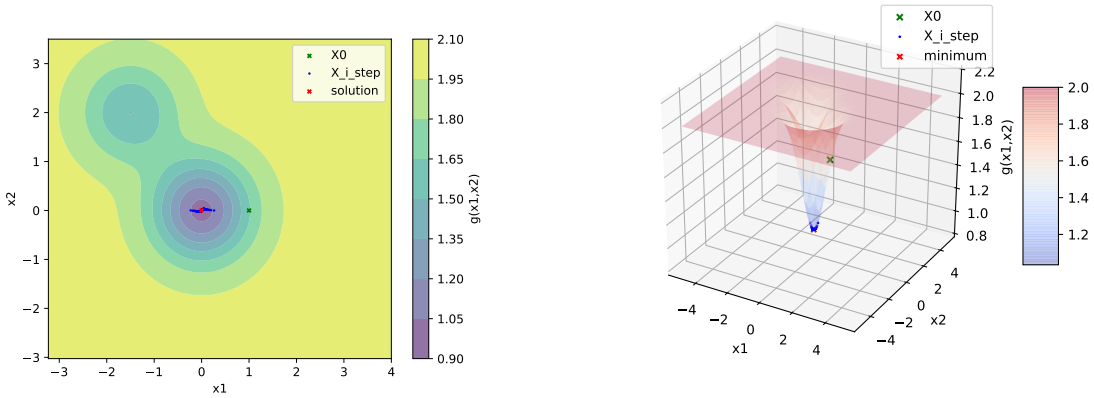


Rys. 3.7. *$G(x_1,x_2)$ dla X = [-2.0, 0.0] oraz długości kroku 1.0; l. iteracji: 992*



Rys. 3.8. *$G(x_1,x_2)$ dla X = [-2.0, 0.0] oraz długości kroku 0.2.; l. iteracji: 42*

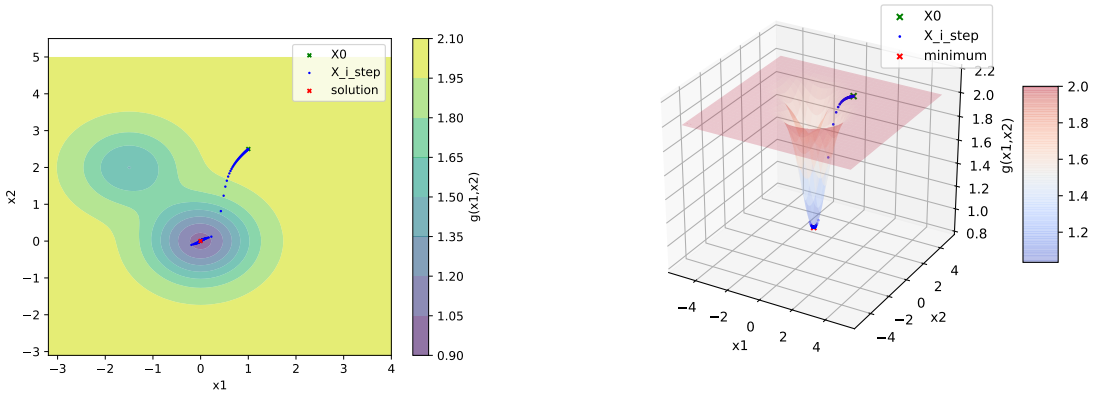
Obranie X_0= [-2, 0] mogłoby być problematyczne, na co wskazuje delikatny łuk na trajektorii kolejnych iteracji algorytmu na rys. 3.7 i 3.8. Mimo tego, dzięki odpowiednim wartościom hiperparametrów zbieżność do minimum globalnego zostaje zachowana.



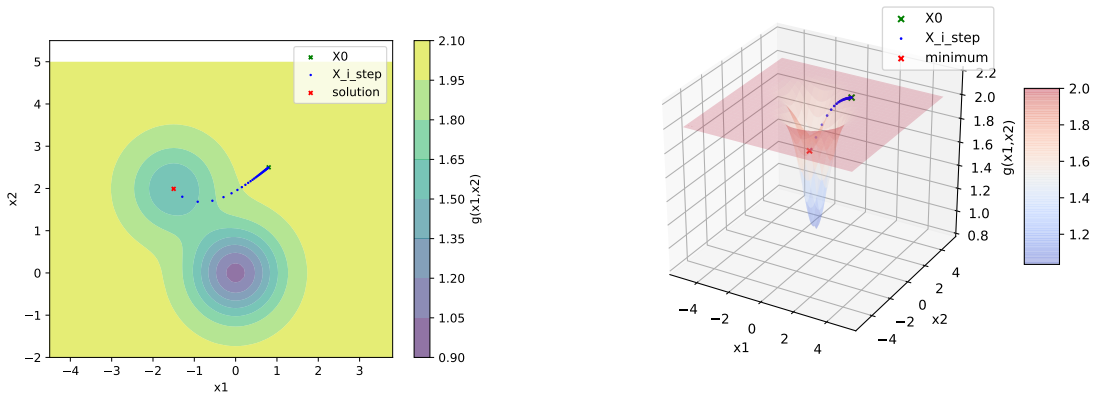
Rys. 3.9. *$G(x_1,x_2)$ dla X = [-1.0, 0.0] oraz długości kroku 1.0; l. iteracji: 994*

Na przykładzie 3.9 widać sytuację, gdzie kolejne pozycje punktów X bardzo szybko wykrywane są w okolicy minimum globalnego. Dzieje się tak dzięki wysokiej wartości długości kroku. To co przyniosło pozytywny efekt na początkowym etapie algorytmu, okazuje się niepożądane w kolejnych iteracjach, gdzie mimo znajdowania się "w okolicy" minimum, wymagane jest bardzo dużo (niemal 1000) kolejnych kroków. Wynika to z wysokiej wartości gradientu na tych "zboczach" g(X).

Ostatnim, bardzo ciekawym przypadkiem jest porównanie wyniku algorytmu dla dwóch zbliżonych do siebie punktów, z których jeden zwraca poprawny wynik, a drugi ląduje w niepożądanym minimum lokalnym. Mowa o punktach *[x_1, x_2] = [1, 2.5]* oraz *[x_1, x_2] = [0.8, 2.5]*, które przedstawione są kolejno na 3.10 i 3.11.



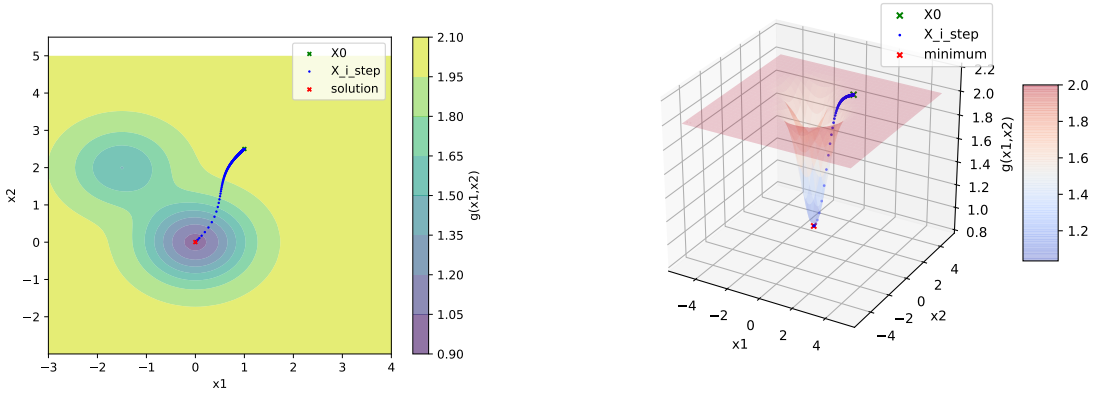
Rys. 3.10. *G(x_1,x_2) dla X = [1.0, 2.5] oraz długości kroku 1.0; l. iteracji: 1042*



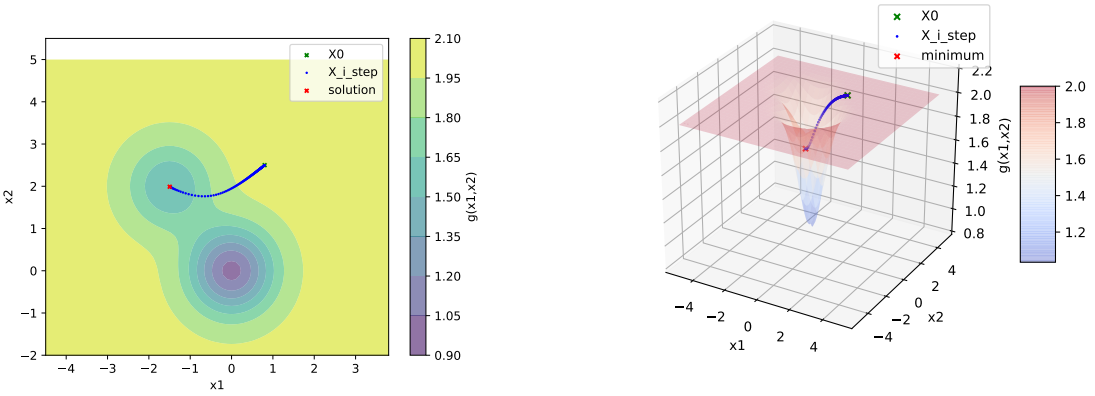
]

Rys. 3.11. *G(x_1,x_2) dla X = [0.8, 2.5] oraz długości kroku 1.0; l. iteracji: 34*

Zmiana długości kroku dla tych dwóch punktów nie zmienia tej sytuacji, co przedstawiono na 3.12 i 3.13.



Rys. 3.12. *G(x1,x2) dla X = [1.0, 2.5] oraz długości kroku 0.2; l. iteracji: 260*



]

Rys. 3.13. *G(x1,x2) dla X = [0.8, 2.5] oraz długości kroku 0.2; l. iteracji: 167*

4. Wnioski

Przeprowadzone badania wpływu zmiany długości kroku i punktów początkowych algorytmu gradientu prostego pokazują, że nie ma uniwersalnych wartości hiperparametrów, dzięki którym algorytm w sposób optymalny znajdzie minimum globalne dowolnej funkcji. Może nie być w stanie znaleźć żadnego minimum, może też się zdarzyć, że zwrócony wynik będzie niepoprawny. Wszystkie wyniki należy traktować z dozą tolerancji. Sposobem na wyeliminowanie błędów, może być wielokrotne uruchomienie algorytmu, dla różnych (randomizowanych) danych wejściowych (X_0, learning rate). Następnie w wyniku podsumowania wszystkich wyników można z *wysoką dozą prawdopodobieństwa* określić minimum globalne.