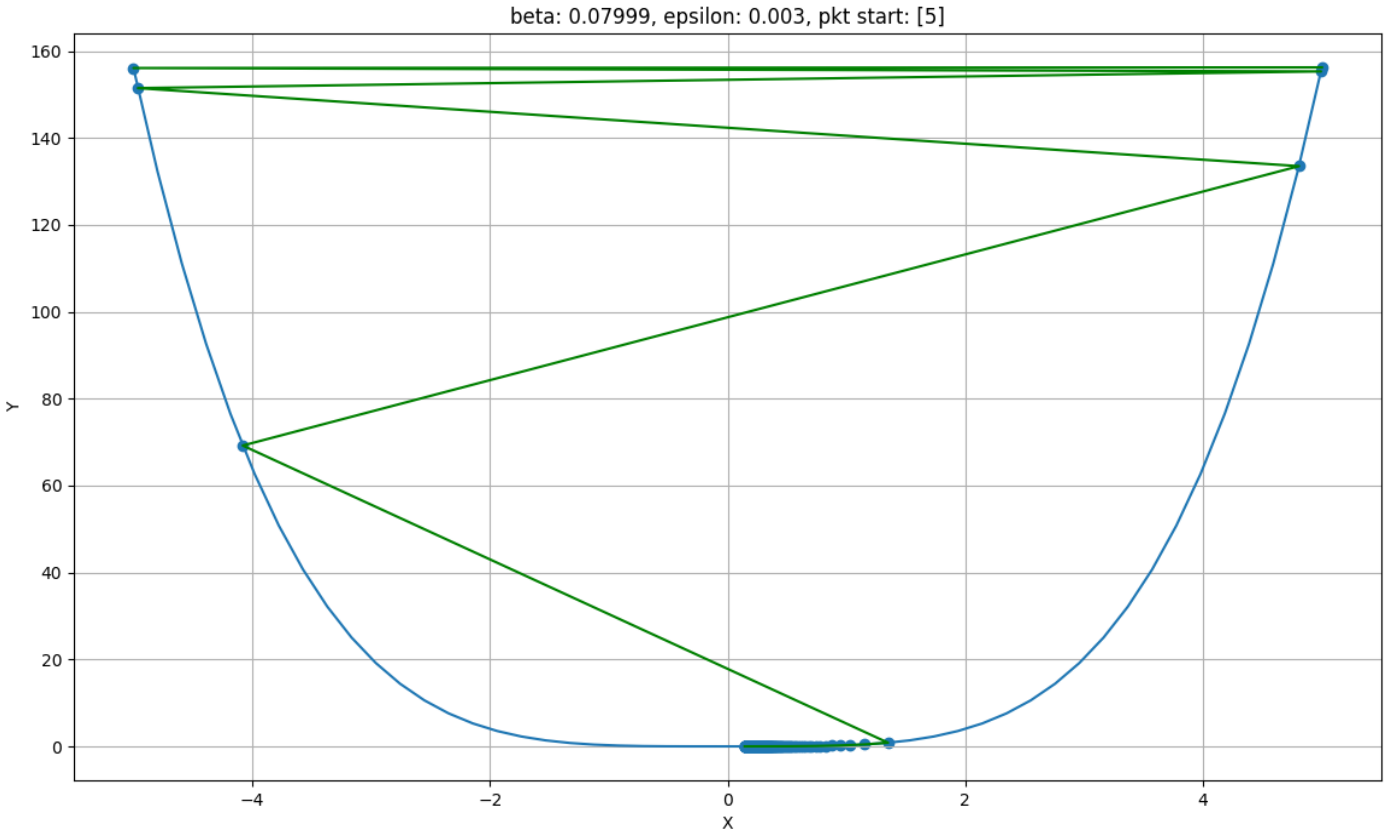
Celem projektu jest stworzenie solvera, który znajduje minimum dowolnej funkcji, jeśli ma wzory jej i jej gradientu i zbadanie, jak rozmiar kroku wpływa na wynik.
Zdefiniowałem "problem" do rozwiązania jako klasę zawierającą funkcję i jej gradient dla łatwego dostępu i obsługi wyznaczania wartości.

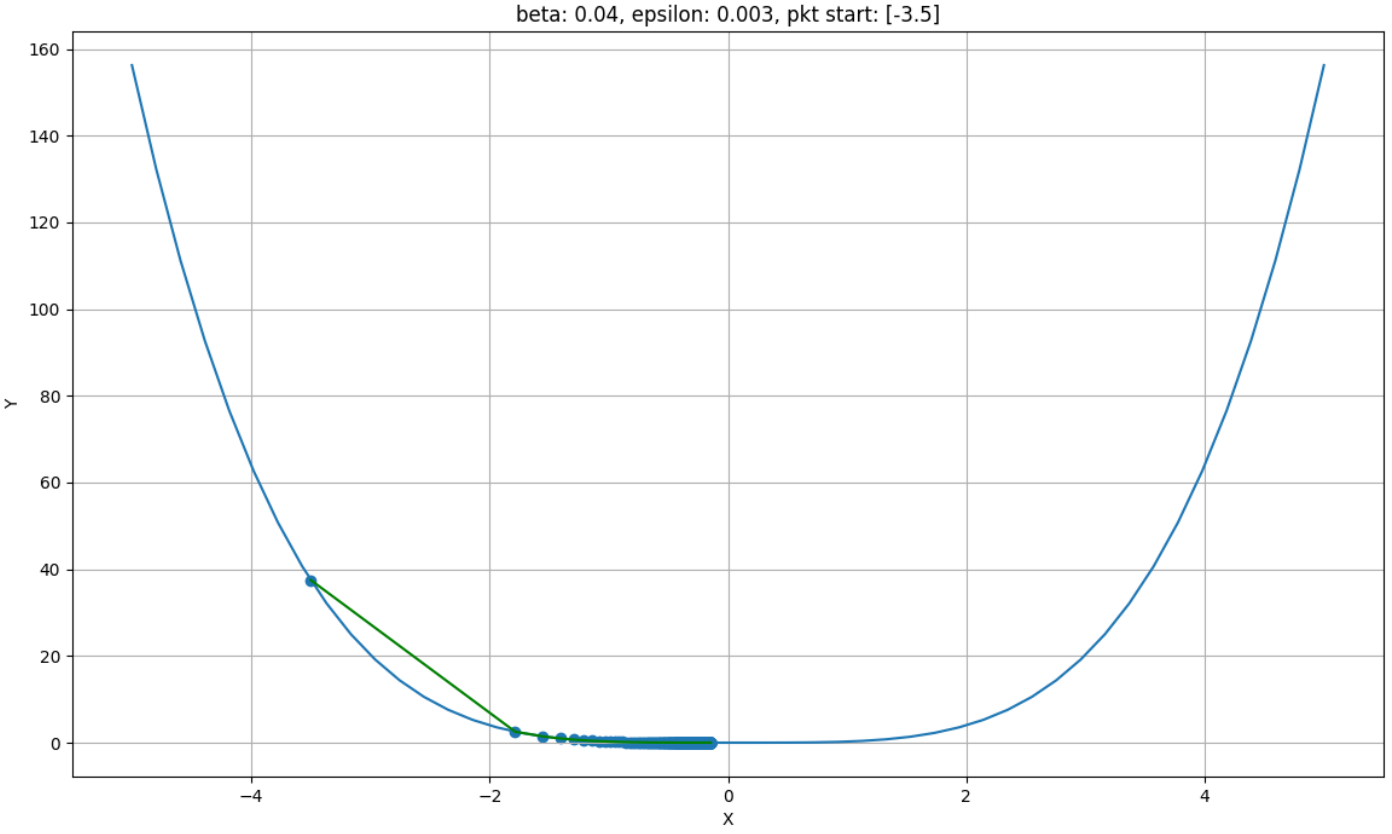
Wybieramy losowy punkt z jakiegoś przedziału, po czym sprawdzamy, jak dobór Bety (kontrolującej wielkość kroku) i epsilona (docelowa dokładność, poszukiwana “płaskość” wykresu w danym miejscu) wpływa na znajdowanie minimów lokalnych funkcji. Za mała beta może powodować długie wykonywanie się algorytmu, a zbyt duża oddalanie się od poprawnego wyniku zamiast zbliżanie się do niego.
Zbyt duży epsilon może powodować natomiast, że zatrzymamy się w poszukiwaniach na słabym rozwiązaniu.
Mały, że nigdy nie osiągniemy pożądanej dokładności przez niedoskonałość algorytmu co skutkuje długim czasem wykonania programu.
Przez "Krok" rozumiemy przesunięcie punktu z jednego miejsca w drugie podczas szukania minimum.
Maksymalna ilość iteracji algorytmu została ustawiona na 1000, aby zapobiedz zbyt długiemu wykonywaniu się algorytmu.

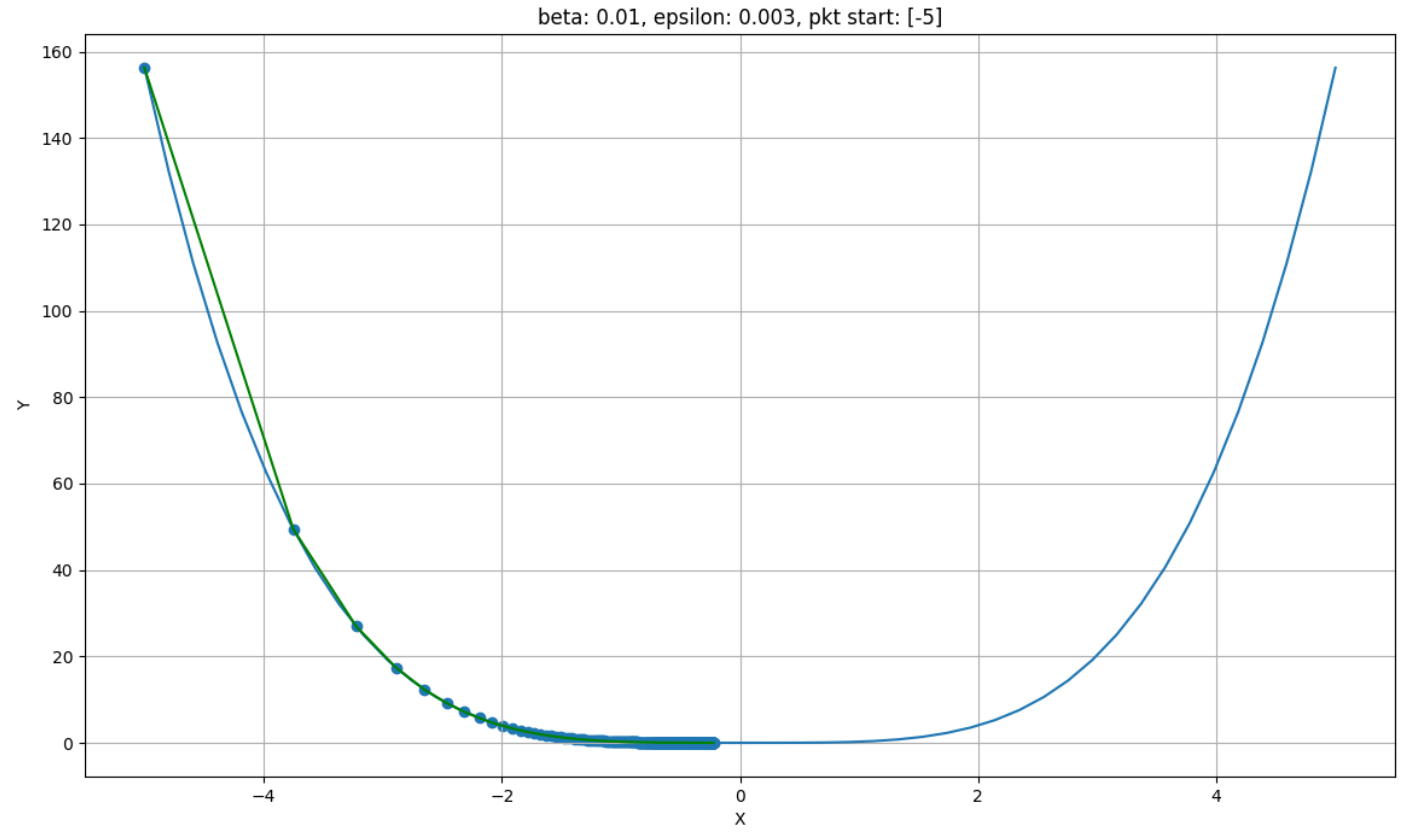
Testy funkcji 1 (x -> y) f(x) = ¼ * x^4
Wybrano 100 losowych punktów z X w przedziale -5 do 5
- przy beta > 0.08, jeśli punkt początkowy jest w -5 lub 5, algorytm będzie oddalał się od poprawnej odpowiedzi
Wynik = wartość funkcji w znalezionym punkcie

epsilon	0.04		0.01	
beta	ilość kroków	wynik	ilość kroków	wynik
0.079	36 +-20	0.002 +- 0.001	107 +- 42	0.0004 +-0.0001
0.04	87 +- 30	0.003 +-0.0007	227 +- 71	0.0005 +- 0.0001
0.01	373 +- 111	0.003 +- 0.0007	949 +- 192	0.0006 +- 0.00008

Jak widzimy zmniejszenie wielkości kroku powoduje, że kroków jest więcej, ale również lekko zmniejsza dokładność otrzymanego wyniku. Dzieje się tak dlatego, że algorytm zatrzymuje się na samej granicy akceptowalnych wyników, podczas gdy większe kroki powodują, że punkt “wskakuje” bliżej środka i tam się zatrzymuje. Przy bardzo małej becie i małym epsilonie (duża wymagana dokładność) widzimy, że algorytm bardzo często osiąga maksymalny limit iteracji.
Z racji na symetryczność tej funkcji nasz algorytm bardzo dobrze sprawdza się w poszukiwaniu minimum (jedynego w funkcji, wynoszącego 0), jeżeli tylko wybierzemy betę pozwalającą na schodzenie “w dół”wykresu dla sprawdzanego przez nas przedziału


beta: 0.07999, epsilon: 0.003, pkt start: [5]
160
140
120
100
80
60
40
20
0
Y
−4
−2
0
2
4
X


beta: 0.04, epsilon: 0.003, pkt start: [-3.5]
160
140
120
100
80
60
40
20
0
Y
−4
−2
0
2
4
X



Testy funkcji 2 (x, y -> z). Trochę mniej niż 0.5 w najniższym, 1 w min lokalnym, ok. 1.5 na "płaskim"

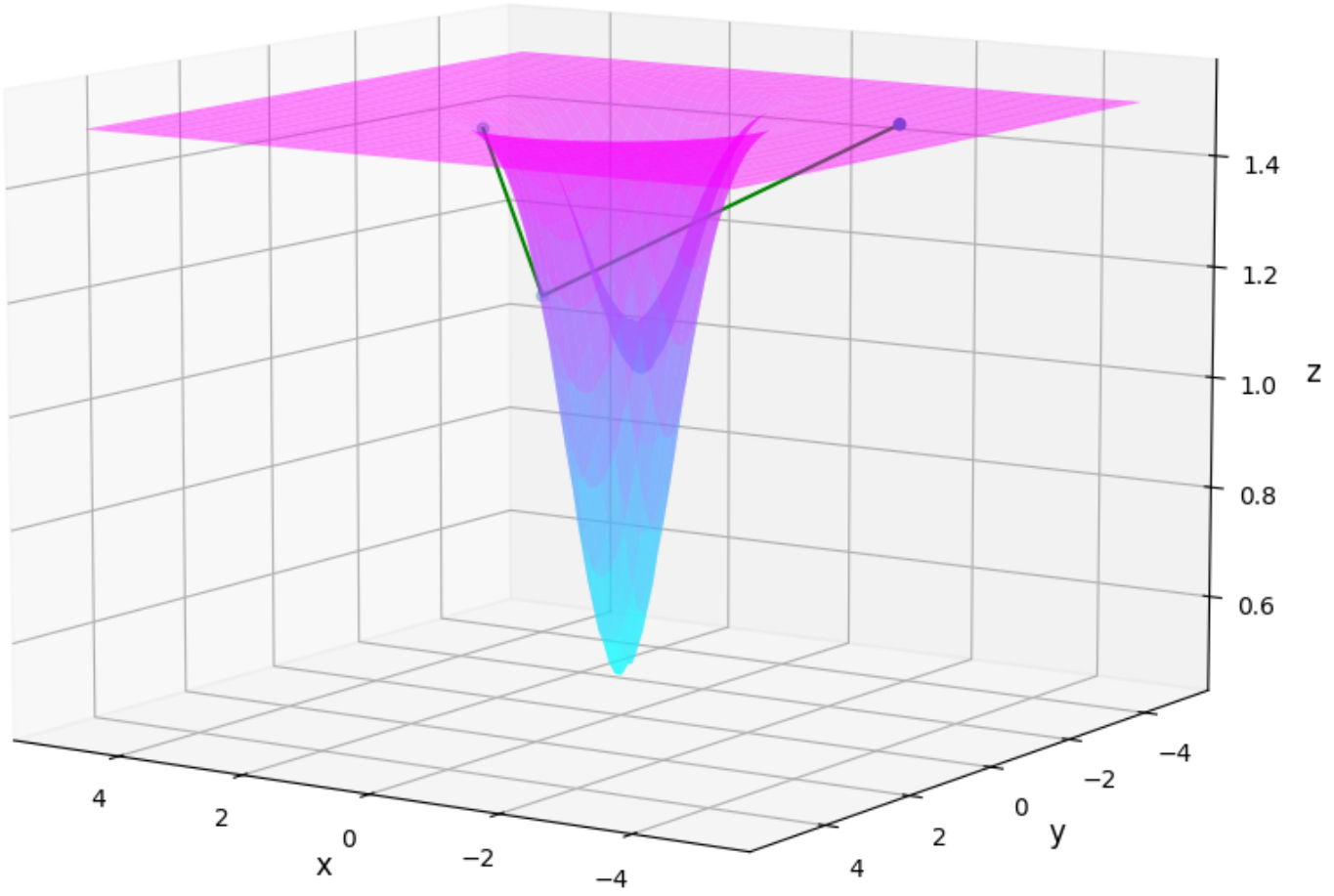
Dla wartości od (-4, -4) do (4, 4):
epsilon ustalony na 0.0001 aby uzyskać jak najdokładniejsze wyniki i jak najmniej “grzęznąć” na bardziej płaskich częściach funkcji.

beta	ilość kroków	wynik
0,7	5 +-8	1.06 +- 0.499
1	590 +- 494	0.93 +- 0.49
7	23 +- 49	1.49 +- 0.000002

Pomiary 1 i 2 dają podobne wyniki, jednak w przypadku bety=1 widzimy bardzo dużą ilość kroków spowodowanych tym, że algorytm nigdy nie osiąga wymaganej dokładności. Natomiast w przypadku beta = 7, punkty nawet jeśli dążą w dobrym kierunku to są niejako “wystrzeliwane” na płaską część wykresu i tam zostają.

Przykład takiego zjawiska:

beta: 7, epsilon: 1e-05, pkt start: [1.7531726078563867, 0.6410759952713159]

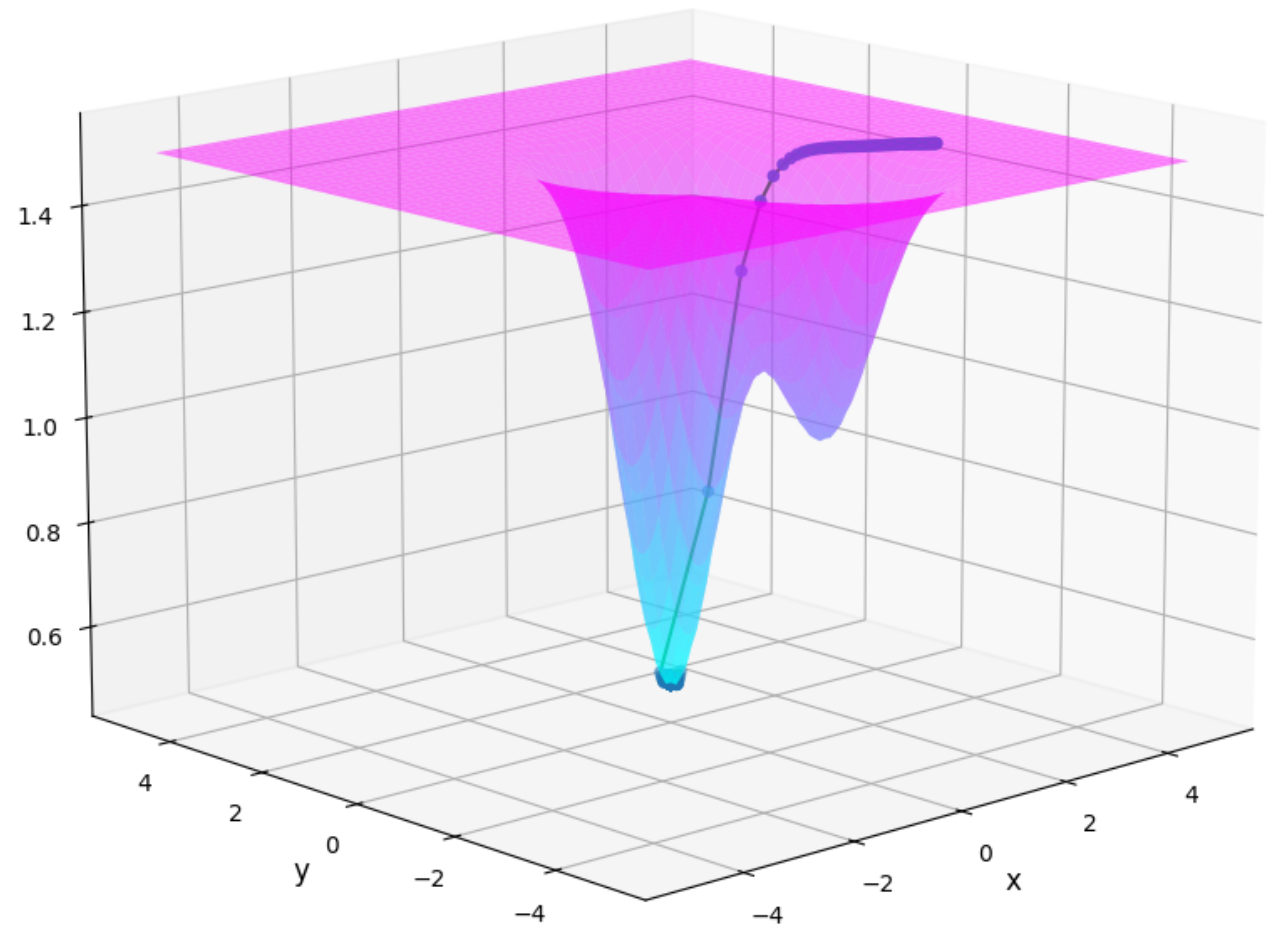


Sytuacja zmienia się, gdy początkowe punkty wybierać będziemy z obszaru bliższego minimów.
epsilon = 0.01

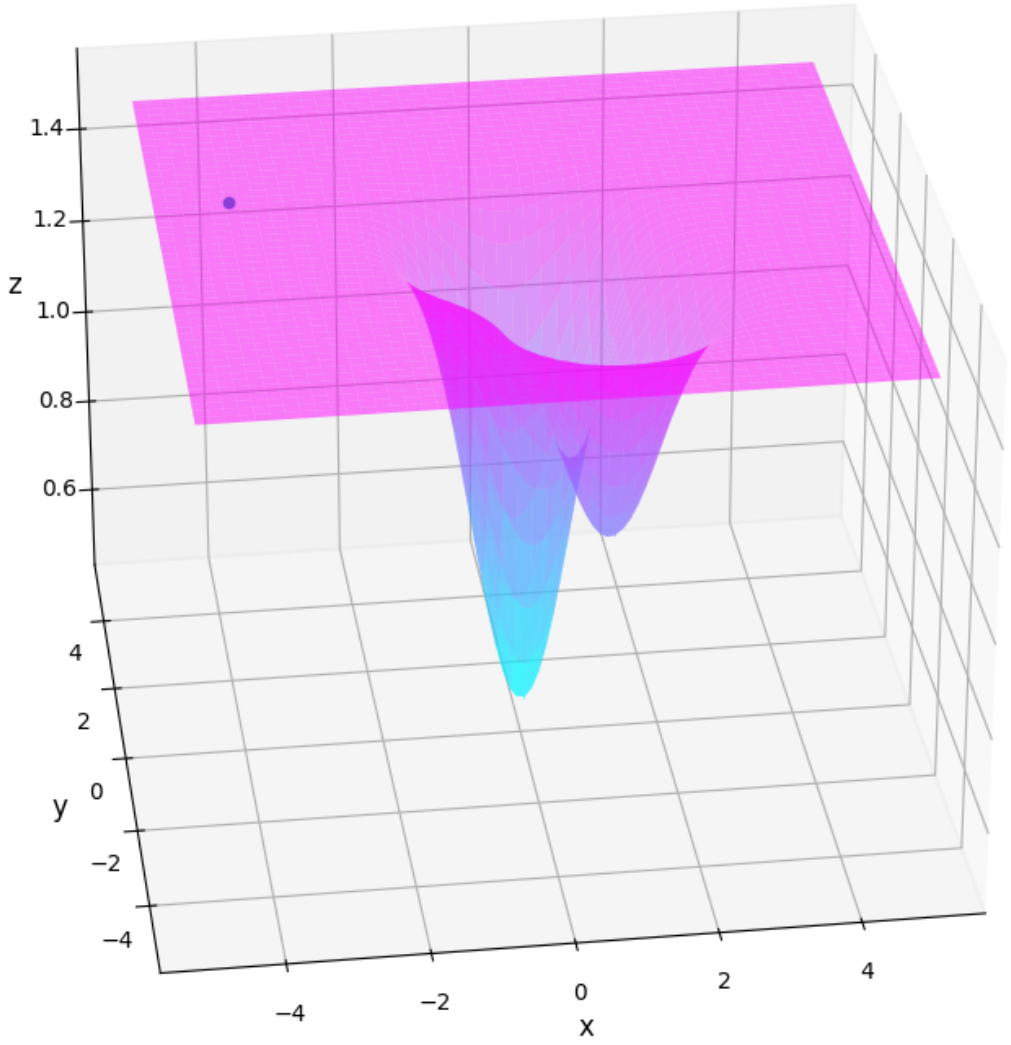
beta	ilość kroków	wynik
0,7	8.5 +-5	0.5 +- 0.1
1	980 +- 140	0.52 +-0.1
3	439 +- 418	1 +- 0.047

Tu widzimy, że dla bety trochę poniżej 1 otrzymywany wynik jest bardzo blisko w minimum globalnym, i jest osiągany szybko. Dla bety = 1 jest on osiągany, ale wykorzystując maksymalną liczbę iteracji przez niemożność osiągnięcia zadanej dokładności (prawdopodobnie odbija się od ścianek blisko minimum).
Beta = 3 jest już za duża

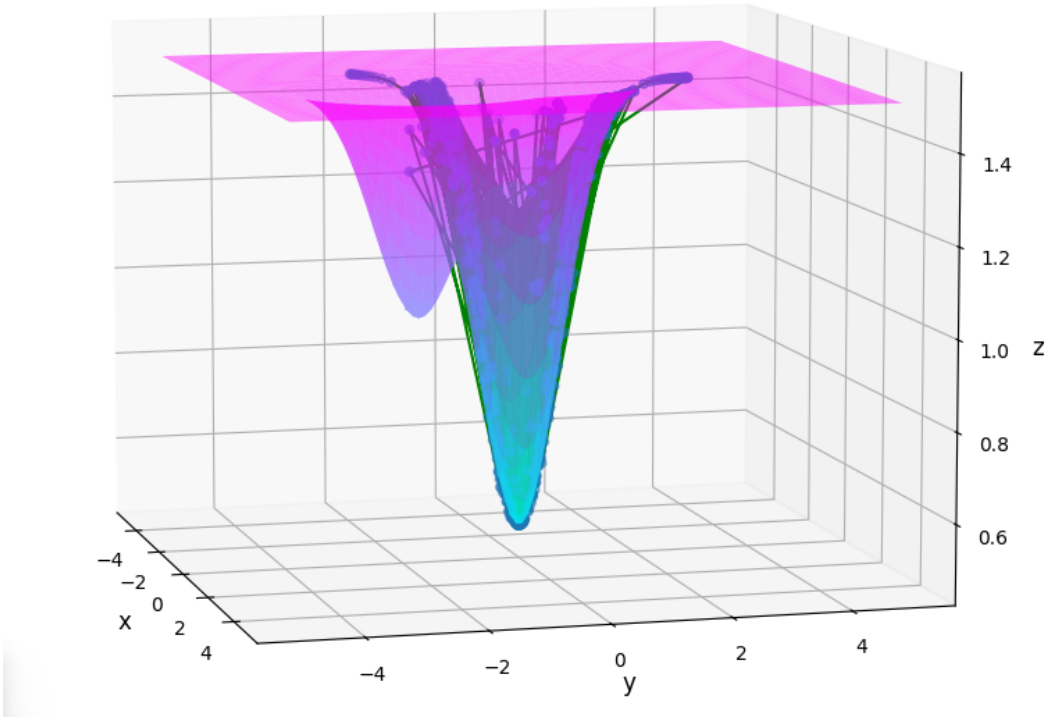
beta: 1, epsilon: 1e-05, pkt start: [3.368333965311309, -1.7627607428666279]



beta: 1, epsilon: 1e-05, pkt start: [-3.9228123257466985, 1.5581949818491827]

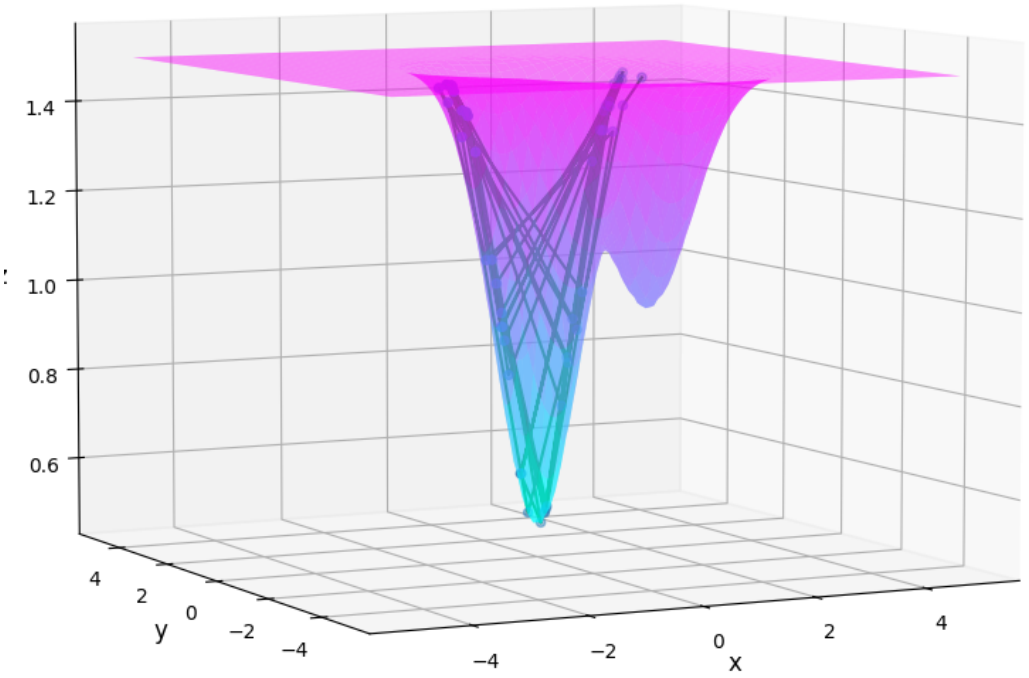


beta: 3, epsilon: 1e-05, pkt start: [-1.4784183177444596, -2.5235454231582954]



z większym epsilonem, który powoduje jednak częstsze “grzęźnięcie” na mniejszych wypłaszczeniach:

beta: 3, epsilon: 0.1, pkt start: [1.901700830150058, -0.0360801894410665]



Wniosek:
Ta metoda wyznaczania minimum funkcji jest bardzo nieefektywna w wypadku takiej funkcji.
Poza centralnymi minimami na środku osi współrzędnych funkcja jest bardzo "płaska" przez co gradient ma bardzo małą wartość i punkt nie rusza się z miejsca. Działa jedynie jeśli obszar z którego wybieramy losowy punkt jest bardzo blisko tych “dołków”.

Wnioski ogólne:
Efektywność algorytmu mocno zależy od rodzaju funkcji, z jaką mamy do czynienia. W przypadku funkcji “wypłaszczających się” w granicach musimy wybierać punkty z bliskiego otoczenia minimum żeby osiągnąć jakkolwiek satysfakcjonujący wynik. Jednak gdy funkcja ciągle rośnie lub maleje, algorytm znajduje minimum lokalne dosyć łatwo i dokładnie, o ile beta nie jest zbyt duża.