Ireneusz Okniński

Nr albumu: 310228

Wstęp do sztucznej inteligencji – sprawozdanie laboratorium 1.

1. Cel.

Celem zadania laboratoryjnego była implementacja algorytmu gradientu prostego oraz zbadanie zbieżności tego algorytmu przy użyciu funkcji $q(x) = \sum_{i=1}^{n} \alpha^{\frac{i-1}{n-1}} x_i^2$ dla $\alpha \in \{1,\ 10,\ 100\}$, $x \in [-100,\ 100]^n \subset \mathbb{R}^n$, $n = 10$. Funkcja ta posiada jedno minimum w punkcie o wszystkich współrzędnych zerowych. Wartość funkcji w tym punkcie wynosi 0. Współczynnik α odpowiada za szybkość wzrostu funkcji $q(x)$.

2. Implementacja

Katalog zawiera plik main.py, który jest implementacją rozwiązania, plik requirements.txt z bibliotekami wymaganymi do działania skryptu oraz katalog plots z wykresami wygenerowanymi przez skrypt.

Struktura pliku main.py:

solver – funkcja, która implementuje algorytm gradientu prostego. Przyjmuje funkcję celu, punkt początkowy, liczbę iteracji oraz rozmiar kroku. Zwraca punkt końcowy, oraz listę wartości funkcji po każdej iteracji algorytmu.

q – funkcja używana do testowania algorytmu. Przyjmuje wektor liczb rzeczywistych i zwraca liczbę rzeczywistą obliczoną według wzoru $q(x) = \sum_{i=1}^{n} \alpha^{\frac{i-1}{n-1}} x_i^2$.

is_minimum – funkcja pomocnicza, która sprawdza, czy dany punkt jest minimum funkcji q z dokładnością do epsilon. Przyjmuje punkt, oraz epsilon, zwraca wartość logiczną.

W wywołaniu programu definiuję listy wartości, dla których chcę zbadać zbieżność algorytmu, wywołuję funkcję solver dla każdej kombinacji podanych argumentów, mierzę czas jej działania oraz tworzę wykresy z zebranych danych.

3. Obserwacje

Przyjąłem następujące parametry:

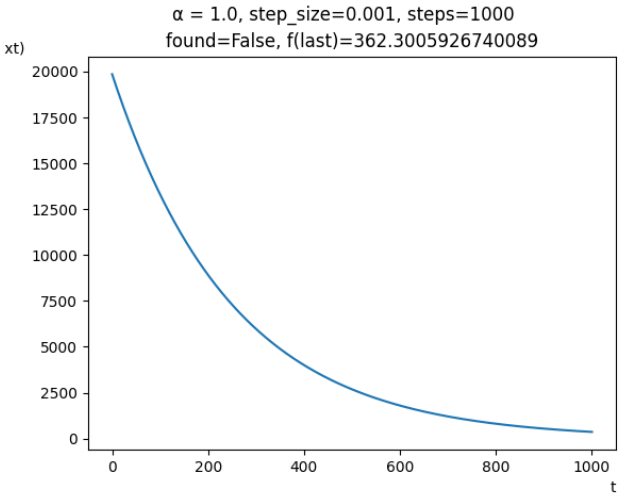
Liczba iteracji: 1000, 2000

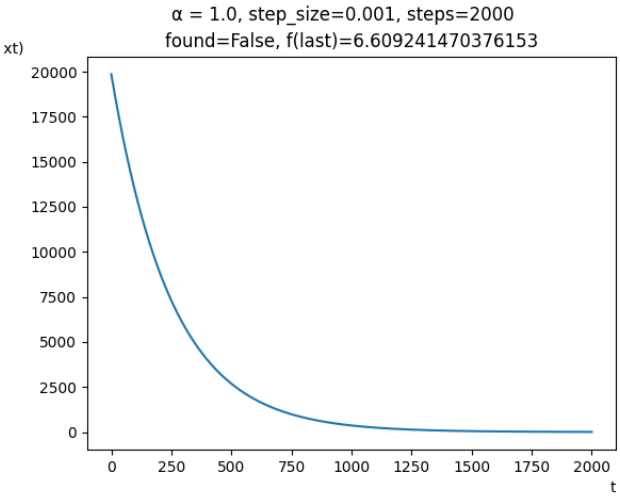
Rozmiar kroku: 0.001, 0.01, 0.1

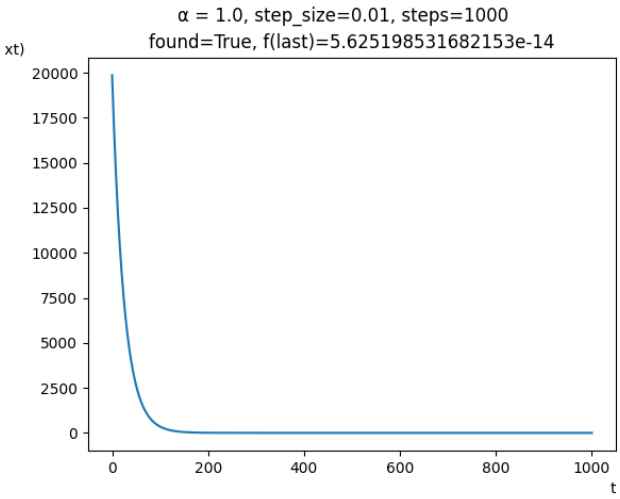
α: 1, 10, 100

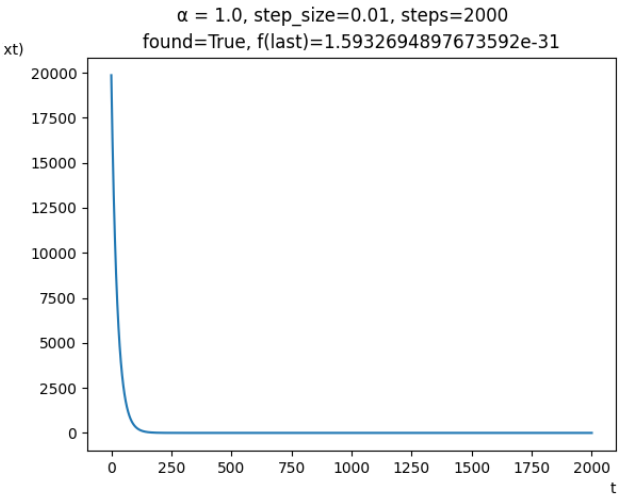
Aby ujednolicić wyniki za każdym razem funkcja byłą wywoływana dla takiego samego punktu początkowego.

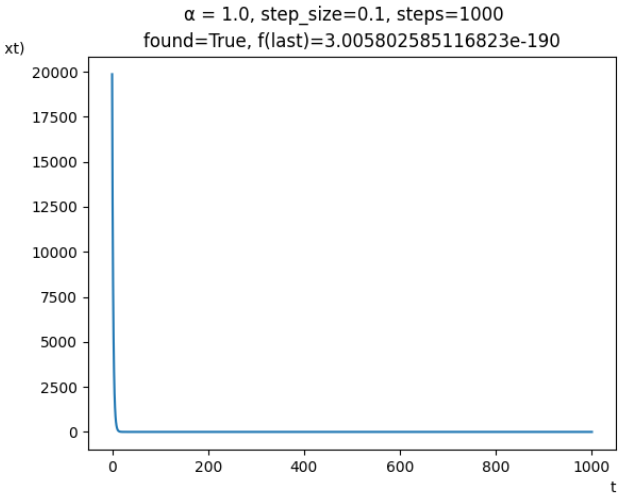
Otrzymałem następujące wyniki:

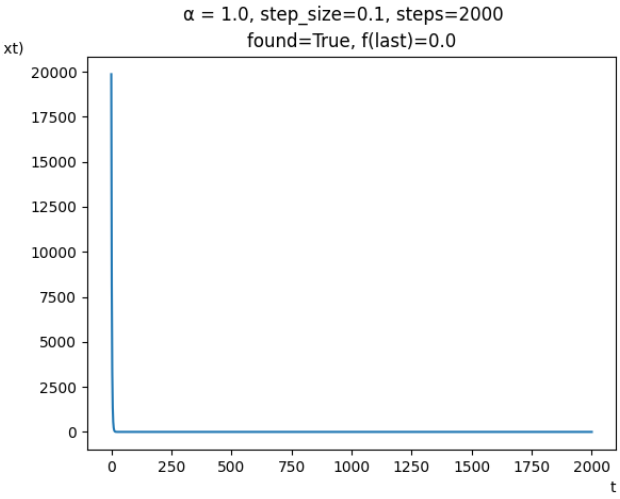

α = 1.0, step_size=0.001, steps=1000
found=False, f(last)=362.3005926740089
xt)
t

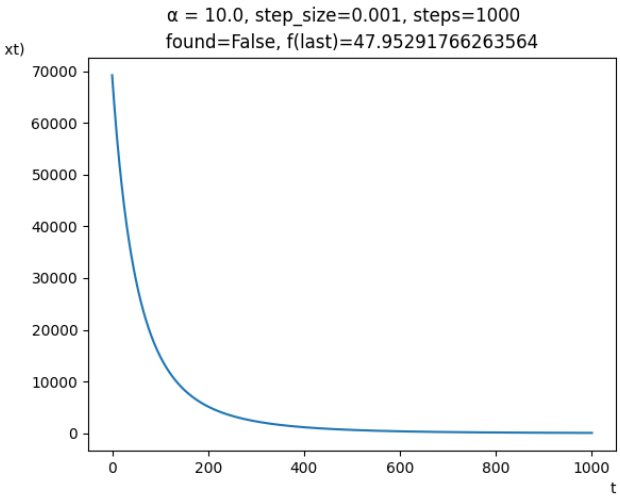

α = 1.0, step_size=0.001, steps=2000
found=False, f(last)=6.609241470376153
xt)
t

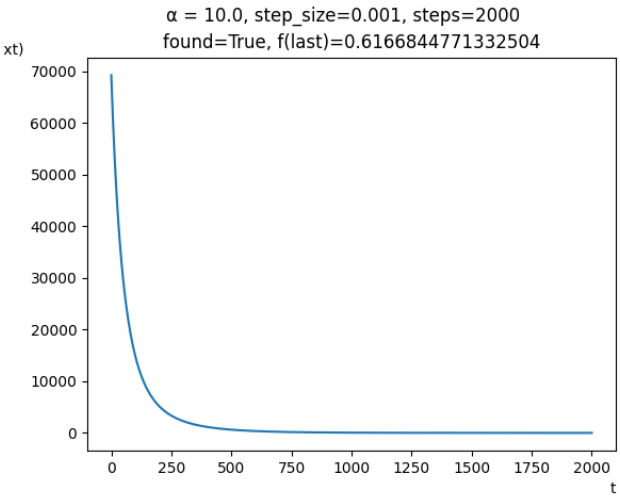

α = 1.0, step_size=0.01, steps=1000
found=True, f(last)=5.625198531682153e-14
xt)
t

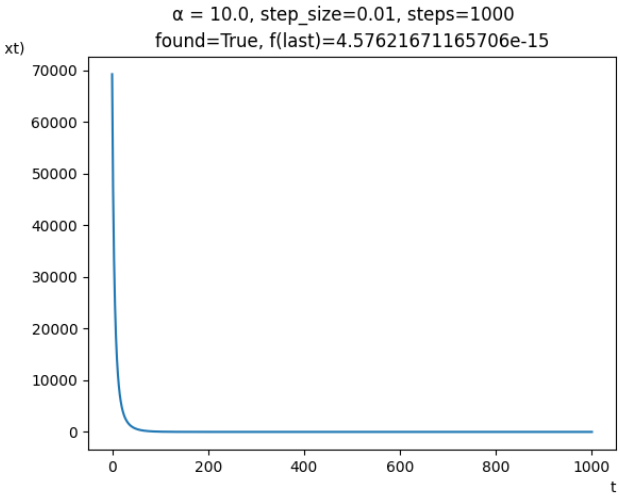

α = 1.0, step_size=0.01, steps=2000
found=True, f(last)=1.5932694897673592e-31
xt)
t

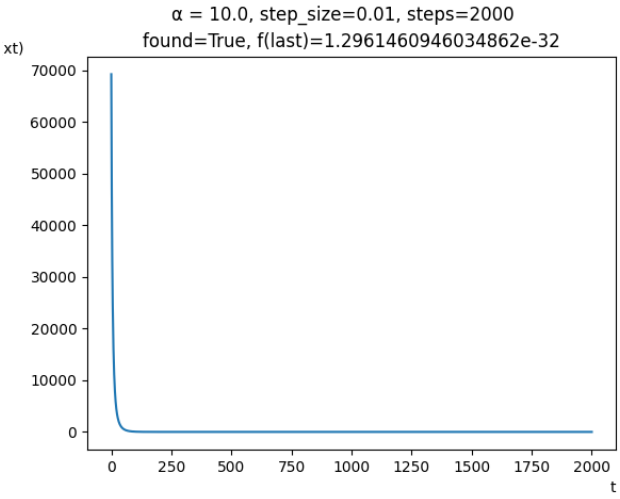

α = 1.0, step_size=0.1, steps=1000
found=True, f(last)=3.005802585116823e-190
xt)
t

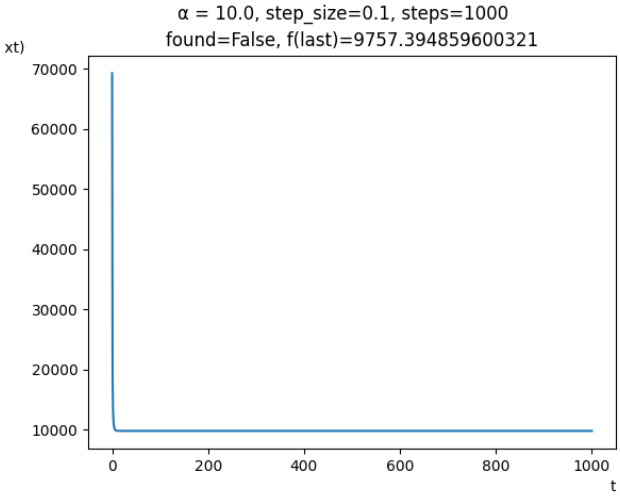

α = 1.0, step_size=0.1, steps=2000
found=True, f(last)=0.0
xt)
t

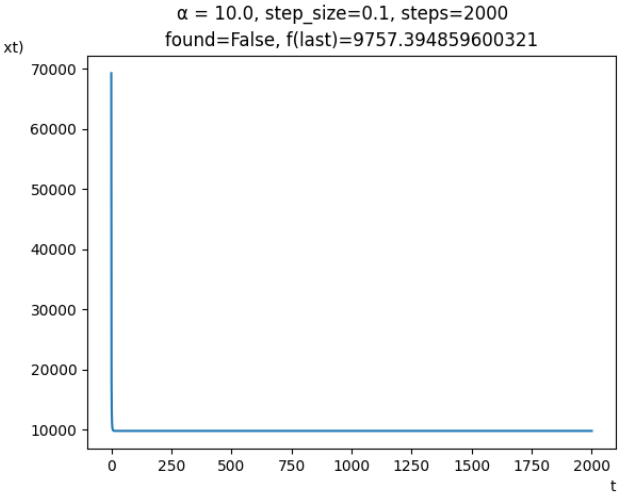

α = 10.0, step_size=0.001, steps=1000
found=False, f(last)=47.95291766263564
xt)
t

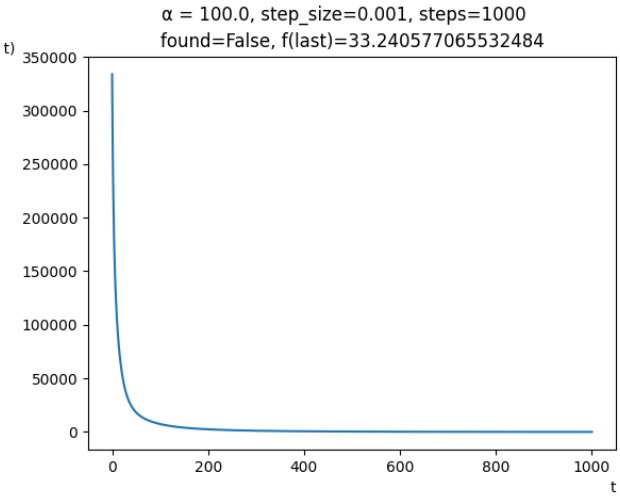

α = 10.0, step_size=0.001, steps=2000
found=True, f(last)=0.6166844771332504
xt)
t

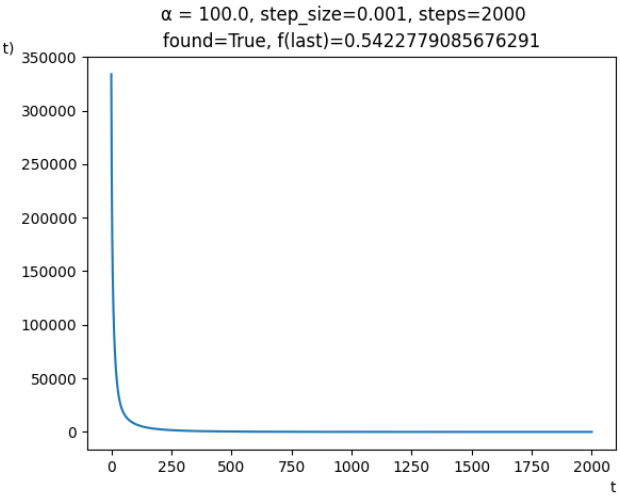

α = 10.0, step_size=0.01, steps=1000
found=True, f(last)=4.57621671165706e-15
xt)
70000
60000
50000
40000
30000
20000
10000
0
0
200
400
600
800
1000
t

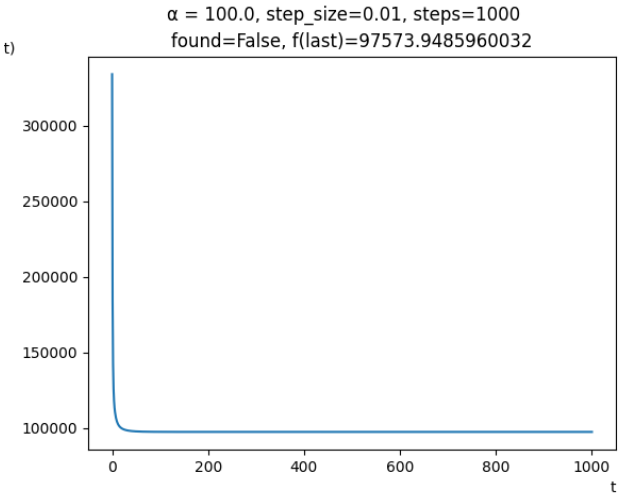

α = 10.0, step_size=0.01, steps=2000
found=True, f(last)=1.2961460946034862e-32
xt)
70000
60000
50000
40000
30000
20000
10000
0
0
250
500
750
1000
1250
1500
1750
2000
t

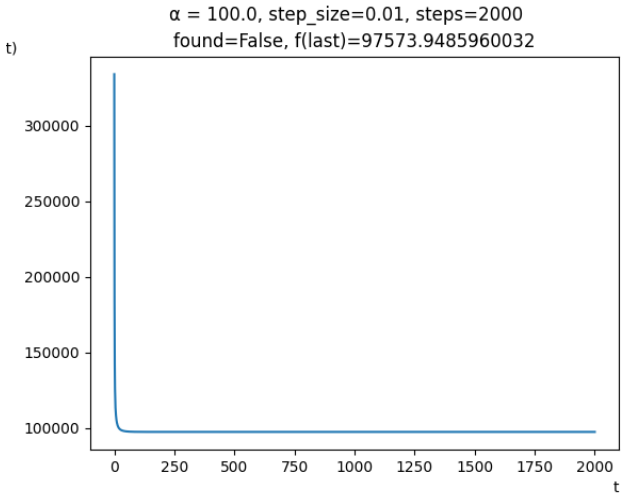

α = 10.0, step_size=0.1, steps=1000
found=False, f(last)=9757.394859600321
xt)
70000
60000
50000
40000
30000
20000
10000
0
200
400
600
800
1000
t

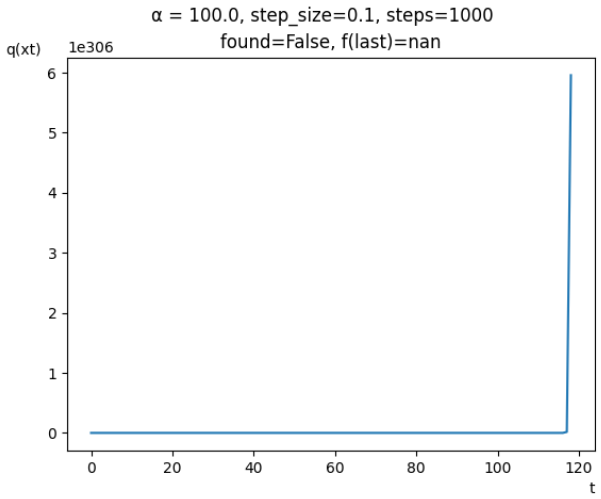

α = 10.0, step_size=0.1, steps=2000
found=False, f(last)=9757.394859600321
xt)
70000
60000
50000
40000
30000
20000
10000
0
250
500
750
1000
1250
1500
1750
2000
t

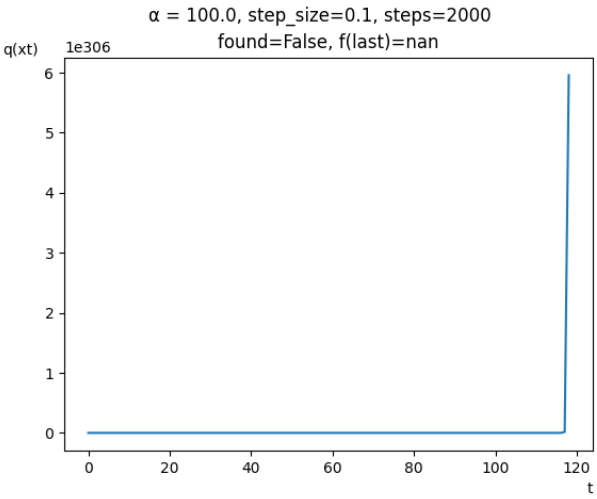

α = 100.0, step_size=0.001, steps=1000
found=False, f(last)=33.240577065532484
t)
350000
300000
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000
t


α = 100.0, step_size=0.001, steps=2000
found=True, f(last)=0.5422779085676291
t)
350000
300000
250000
200000
150000
100000
50000
0
0
250
500
750
1000
1250
1500
1750
2000
t









Czy znaleziono minimum? (dla epsilon=1)

steps	1000			2000		
α \ stepsize	0.001	0.01	0.1	0.001	0.01	0.1
1	NIE	TAK	TAK	NIE	TAK	TAK
10	NIE	TAK	NIE	TAK	TAK	NIE
100	NIE	NIE	NIE	TAK	NIE	NIE

Algorytm był zbieżny dla wszystkich wartości parametrów oprócz kombinacji α=100, stepsize=0.1. Z wykresów wynika, że od ok. 120 iteracji algorytm znajdował punkty leżące coraz dalej od minimum. Spowodowane to jest niewłaściwym doborem parametrów α i stepsize. Funkcja dla α=100 funkcja q(x) rośnie bardzo szybko. Duża wartość parametru step_size sprawia, że znajdowane punkty oscylują nad minimum funkcji. Oscylacja ta rośnie coraz szybciej, w skutek czego algorytm szybko staje się rozbieżny do nieskończoności.

W pozostałych przypadkach algorytm był zbieżny. Jako kryterium znalezienia minimum przyjąłem warunek, że odległość każdej współrzędnej punktu końcowego od zera wynosi maksymalnie 1. Dla 1000 iteracji algorytm nie znalazł minimum ani razu, gdy współczynnik stepsize był równy 0.001. W przypadku wykonania 2000 iteracji minimum nie znaleziono jedynie dla α=1. Wynika z tego, że przy małym rozmiarze kroku algorytm potrzebuje więcej powtórzeń do znalezienia minimum.

Z wykresów wynika też, że dla większego parametru kroku algorytm szybciej staje się zbieżny.

Wartość funkcji q(x) w ostatniej iteracji argumentu.

steps	1000			2000		
α \ stepsize	0.001	0.01	0.1	0.001	0.01	0.1
1	362.3	5.63e-14	3.0e-190	6.61	1.59e-31	0.0
10	47.95	4.58e-15	9757.39	0.62	1.3e-32	9757.39
100	33.24	97573.95	nan	0.54	97573.95	nan

Po dwukrotnym zwiększeniu ilości powtórzeń wartości funkcji q(x) w większości przypadków zmniejszyły się. Oznacza to, że w przypadku α=1, stepsize=0.001 zwiększenie liczby iteracji powinno spowodować znalezienie minimum. W przypadku α=100, stepsize=0.01 i α=10, stepsize=0.1 wartości nie uległy zmianie. Algorytm zatrzymał się w punkcie, z którego nie jest w stanie wyjść. Zwiększenie liczby iteracji nie zmniejszy wartości q(x) dla tych parametrów.

Czas wykonania algorytmu. [sekund]

steps	1000			2000		
α \ stepsize	0.001	0.01	0.1	0.001	0.01	0.1
1	0.84	0.85	0.91	1.78	1.74	1.75
10	0.88	0.91	0.88	1.76	1.74	1.79
100	0.96	0.88	0.92	1.80	1.80	1.76

Czas wykonywania algorytmu nie jest zależny od parametrów α oraz stepsize. Dwukrotne zwiększenie ilości iteracji spowodowało w przybliżeniu dwukrotne wydłużenie wykonywania algorytmu. Jest on zatem liniowo zależny od liczby iteracji.

4. Wnioski końcowe.

Metoda gradientu prostego pozwala skutecznie obliczyć minimum funkcji. Kuczem do jego poprawnego działania jest poprawne dobranie współczynnika rozmiaru kroku do przebiegu danej funkcji.

Gdy funkcja zmienia się wolno dobrym rozwiązaniem będzie większa wartość współczynnika rozmiaru kroku. Dzięki temu algorytm będzie potrzebował mniejszej liczby iteracji do znalezienia minimum, co za tym idzie wzrośnie jego wydajność.

Gdy funkcja zmienia się szybko współczynnik rozmiaru kroku powinien mieć mniejszą wartość. Wymagana będzie przez to większa liczba iteracji. Unikniemy jednak sytuacji, w której algorytm staje się chaotyczny lub, co gorsze, całkowicie rozbieżny.