RAPORT

Arkadiusz Kojtek 310746

Uruchamianie programów

Do uruchomienia programu korzysta się z linii poleceń w którą wpisuje się po:
-m
Tryb w którym będzie działał program (g od gradientu oraz n od Newtona)
-l
Lista argumentów pozycji startowej
-c
Wartość stałej
-r
Krok wykorzystywany w algorytmie
Np.:
python3 lab1_algorithms.py -m g -l 1 2 3 4 5 6 7 8 9 10 -c 10 -r 0.4

Implementacja algorytmów

W projekcie wykorzystane zostały trzy funkcje, które służyły do obliczania wartości poszczególnych elementów algorytmów to jest gradientu, hesjanu oraz wartości funkcji. Odpowiedzialne za to był kolejno funkcje:
gradient_function_element()
Można zauważyć, że pochodna funkcji dla każdej ze współrzędnych przyjmuje wartość a**(i-1/n-1)*x*2
hessian_function_element()
Można zauważyć, że każdy element hesjanu tej funkcji przyjmuje wartość równą a**(i-1/n-1)*2
function_element()
Funkcja ta oblicza wartość a**(i-1/n-1)*x**2 dla zadanej współrzędnej
W moim projekcie zaimplementowałem dwa algorytmy, mianowicie:

Algorytm gradientu prostego

Funkcja ***gradient_descent()*** przyjmuje jako argumenty trzy zmienne:
listę argumentów początkowych, stałą wykorzystaną we wzorze, oraz krok wykorzystywany do obliczania zmian w gradiencie.
Po otrzymaniu danych, funkcja wywołuje pętlę **while,** w której to liczona jest wartość pochodnej funkcji dla każdej ze współrzędnych oraz zapisywana do zmiennej **fun_result**. Pętla ta ma działać tak długo dopóki chociaż jedna ze współrzędnych zmieniła się o więcej niż ustalony dla tej funkcji **epsilon** (dla tego projektu przyjąłem epsilon równy 0.00001) względem stanu we wcześniejszej iteracji. Następnie dodawana jest wartość funkcji dla danej współrzędnej do zmiennej **function_sum,** w celu uzyskania sumy funkcji w

poszczególnej iteracji pętli. Informacja o ile nieistotna dla samego algorytmu posłuży mi później do zilustrowania działania programu na wykresie.
Kolejnym krokiem jest obliczenie różnicy (zmienna **difference**) przechowującej wartość zmiennej **fun_result** pomnożonej przez krok ustalony przez użytkownika (**learning_rate**). Następnie sprawdzany jest warunek różnicy między wartością współrzędnej w tej oraz poprzedniej iteracji. Jeżeli warunek jest spełniony zmienna **stop_condition** zostaje zmniejszona o 1. Następnie do współrzędnej przypisywana jest nowa wartość.
W ostatnich liniach pętli **while** zostają przypisane wartości do list **values** oraz **keys**, które wykorzystane zostaną do narysowania wykresu.
Gdy warunek stopu zostanie spełniony pętla kończy swoje działanie a funkcja zwraca wartości funkcji względem każdej iteracji.
.

Algorytm Newtona

Funkcja ***newton_algorithm()*** przyjmuje jako argumenty trzy zmienne:
listę argumentów początkowych, stałą wykorzystaną we wzorze, oraz krok wykorzystywany do obliczania zmian w gradiencie. Sama funkcja działa na bardzo zbliżonej zasadzie co funkcja ***gradient_descent().*** Jedyna różnica polega na obliczaniu wartości zmiennej **fun_result**, która przyjmuje wartość funkcji **gradient_function_element()** dla danej współrzędnej dodatkowo podzielonej przez wartość funkcji **hessian_function_element()** dla tej samej współrzędnej.

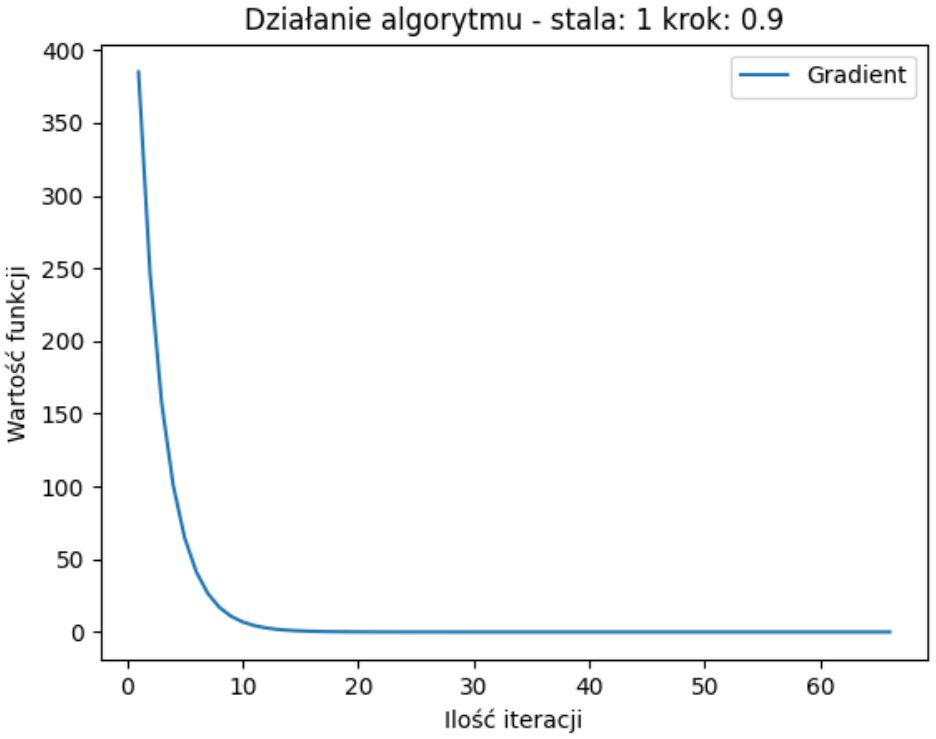
Eksperymenty numeryczne

Zamierzam zbadać wpływ kroku oraz wartości początkowych na działanie algorytmu w zależności od zadanych stałych oraz ilości wymiarów.

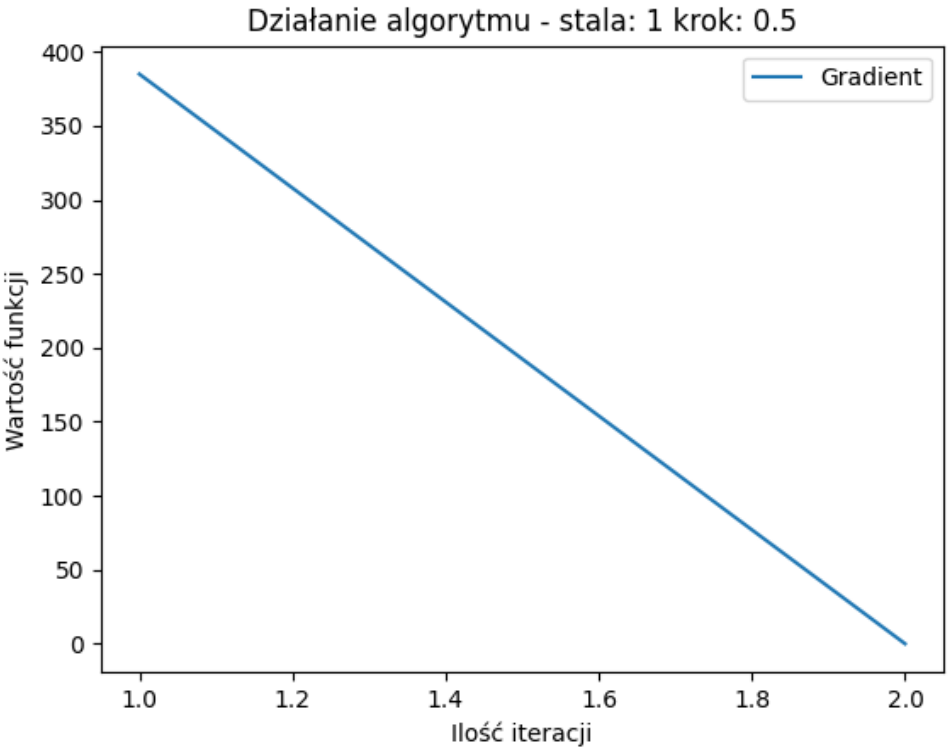
Najpierw zajmę się testowaniem algorytmu gradientu prostego:
Jako pozycję startową wybrałem najpierw 10-sięcio wymiarowy wektor z wartościami od 1 do 10. Stałą ustaliłem równą 1, krok również równych 1.

Program nie był w stanie się zakończyć. Oznacza to że wartości funkcji prawdopodobnie minęły minimum przez zbyt duży krok przez co nie potrafiły już znaleźć miejsca w którym mógłby się zatrzymać.

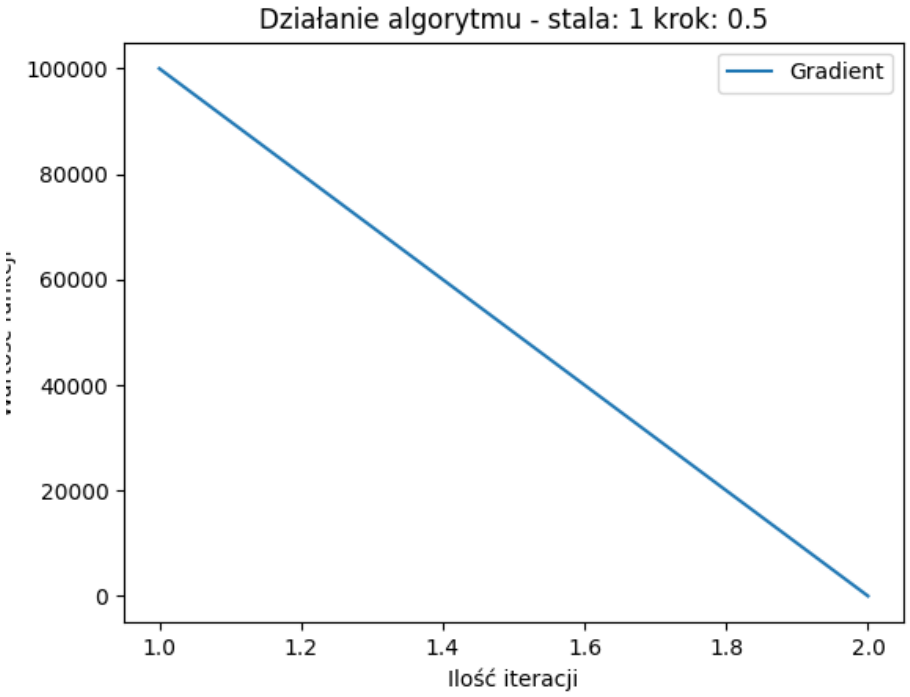
W kolejnych testach zmniejszyłem krok do wartości poniżej 1, zaczynając od 0.9. Program wykonał się znajdując odpowiednie minimum

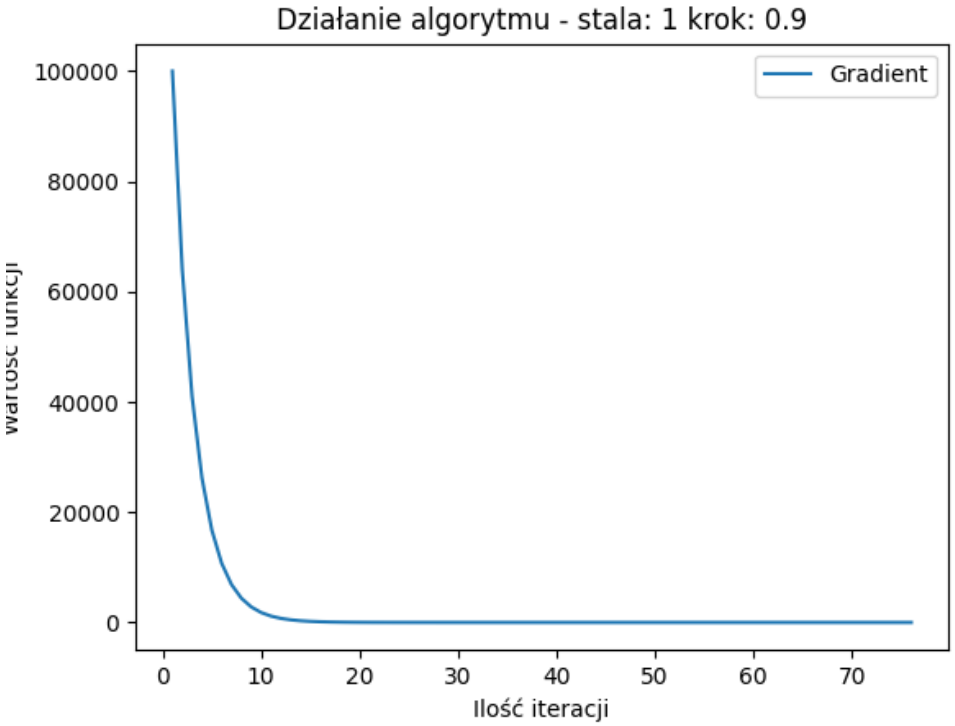


Jednocześnie zauważyłem ciekawą zależność dla wartości kroku równej 0.5, dla niej algorytm znalazł minimum po wykonaniu tylko dwóch iteracji!

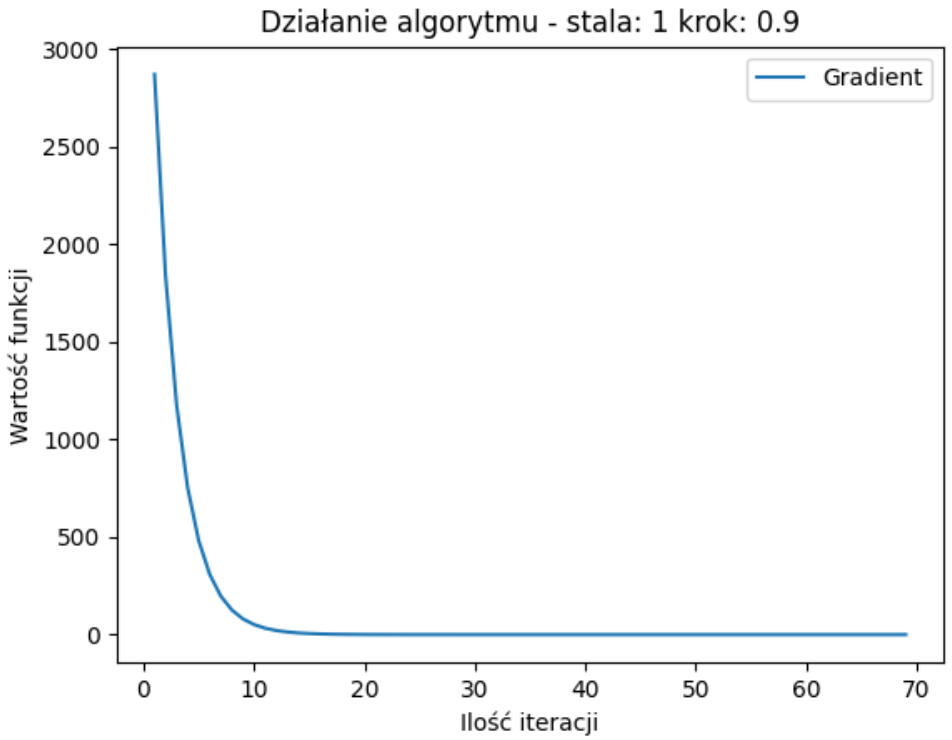


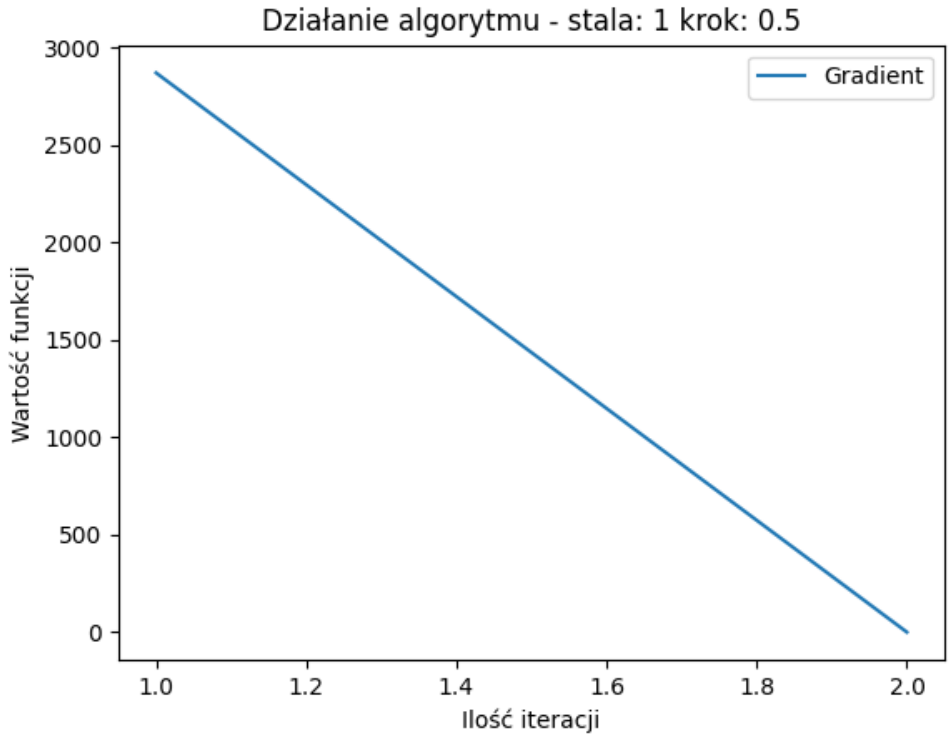
Po stwierdzeniu że dla kroku mniejszego od 0.1 program znajduje bez problemu minimum postanowiłem zmienić wartość punktu początkowego do wektora 10 wymiarowego zapełnionego 100.





Oraz w przypadku 20 wymiarów wektora

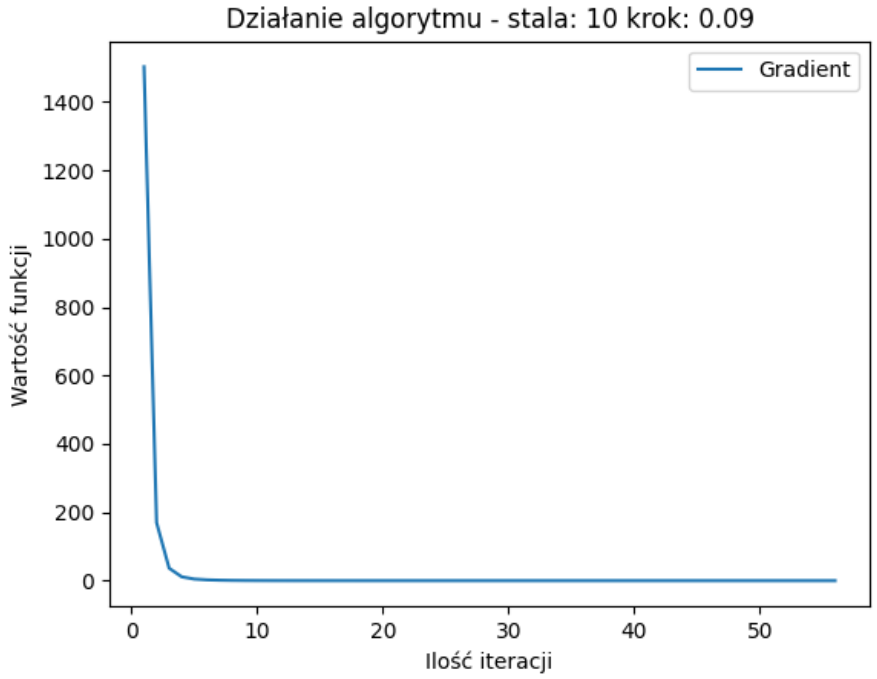


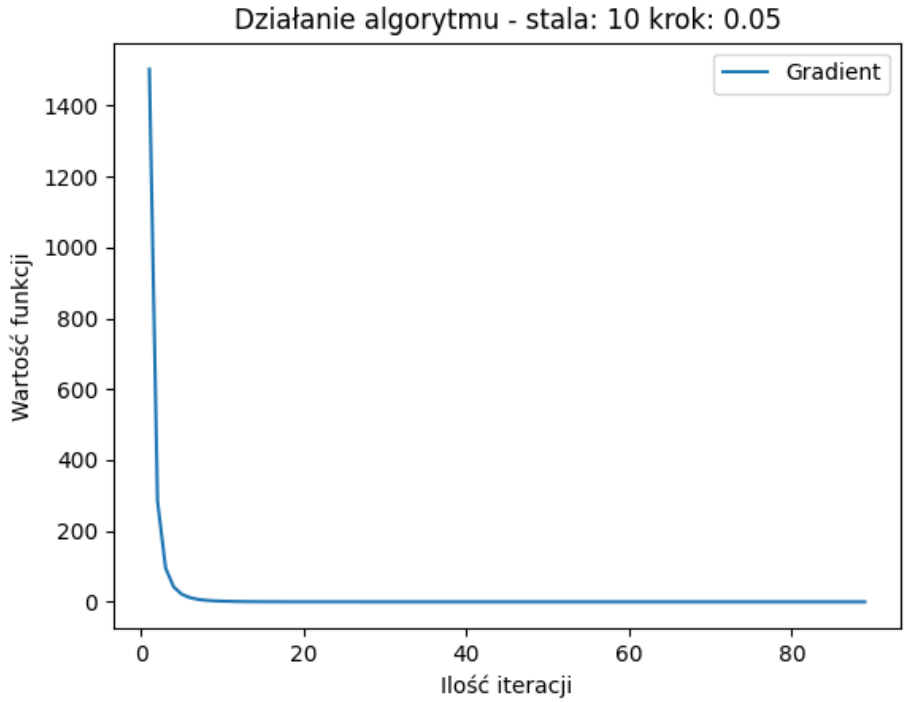


Jak można zauważyć program działa w ten sam sposób niezależnie od wybranej pozycji startowej dla zadanej funkcji. Jedyna różnica pozostaje w wartości samej funkcji.

W następnym kroku zmieniłem wartość stałej na 10 oraz postanowiłem powtórzyć poprzednie testy zaczynając od przestrzeni 10 wymiarów.

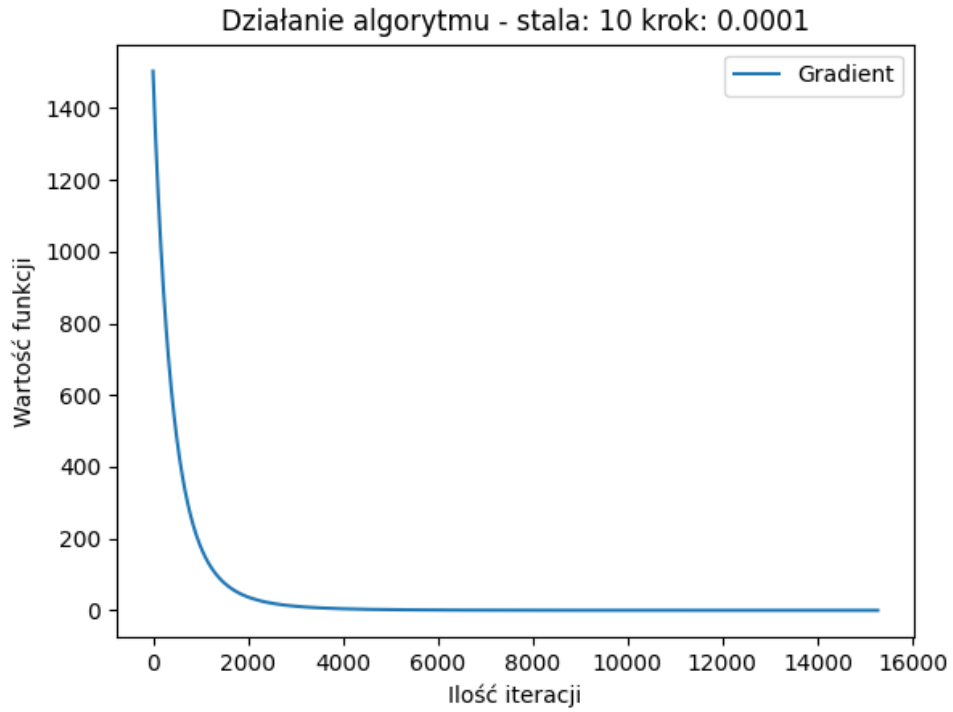
Podczas testów zauważyłem pewną zależność, program znowu nie był w stanie znaleźć minimum dla pewnych wartości kroku. Tym razem jednak funkcja przestawała działać nie od kroku równego 1 a 0.1, znajdując zależność pomiędzy stałą a krokiem postanowiłem przeprowadzić testy jak uprzednio tym razem jednak z podzielonym przez 10 krokiem.



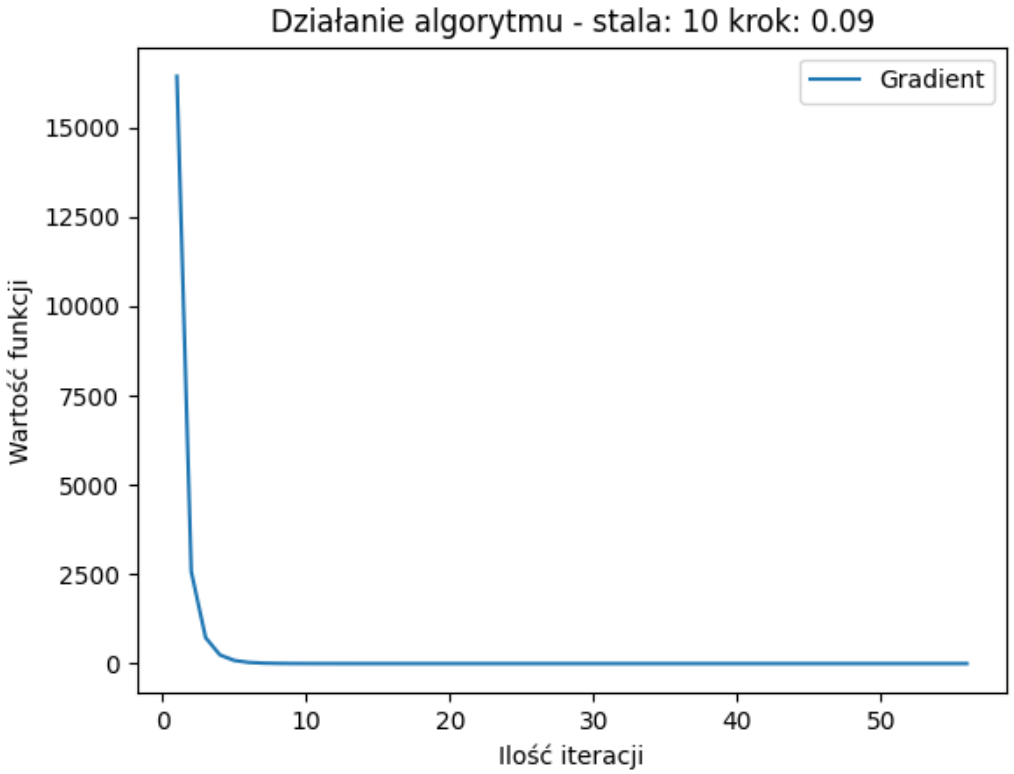


Niestety dla kroku 0.05 nie udało mi się uzyskać minimum przy dwóch iteracjach.

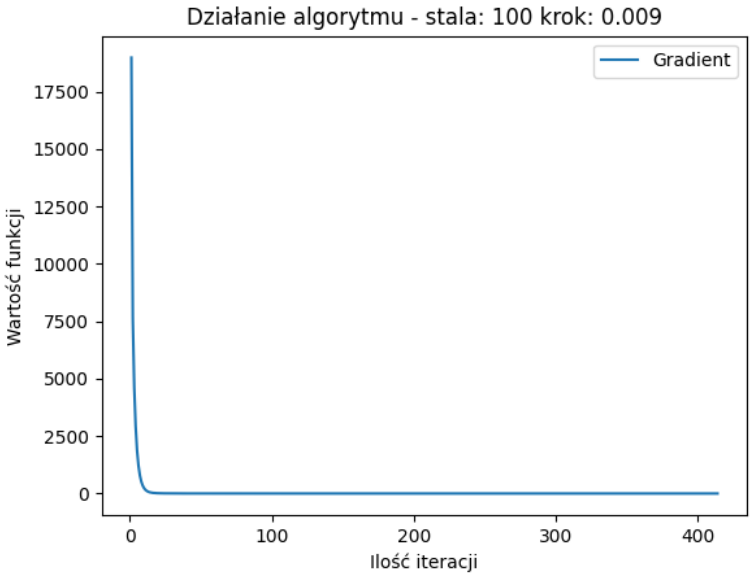
Zauważyłem jednak że wraz z malejącym krokiem zwiększała się iteracji pętli która coraz dłużej musiała szukać miejsca z dostatecznie małymi zmianami.



Następnie znowu zbadałem wpływ pozycji startowej, ponownie sama pozycja nie miała znaczącego wpływu na samo działanie programu a jedynie na wartość funkcji

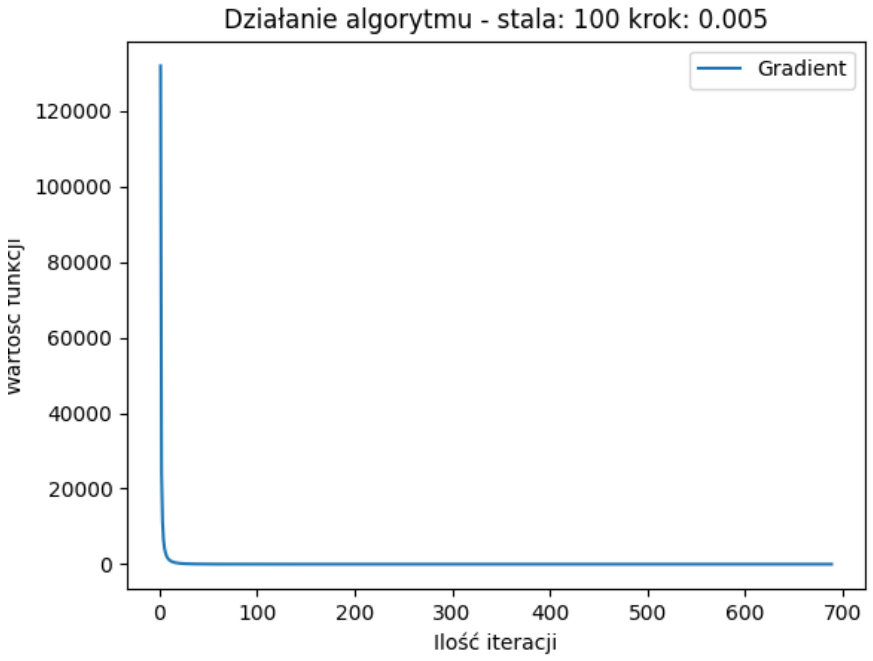


Kolejny test przeprowadziłem na stałej równej 100. Próbując sprawdzić moje spostrzeżenie z wcześniej badanie zacząłem od 0.01. Tak jak podejrzewałem program nie był stanie znaleźć minimum. Natomiast dla wartości kroku nieco mniejszej jak np 0.009 program działał bez zarzutu

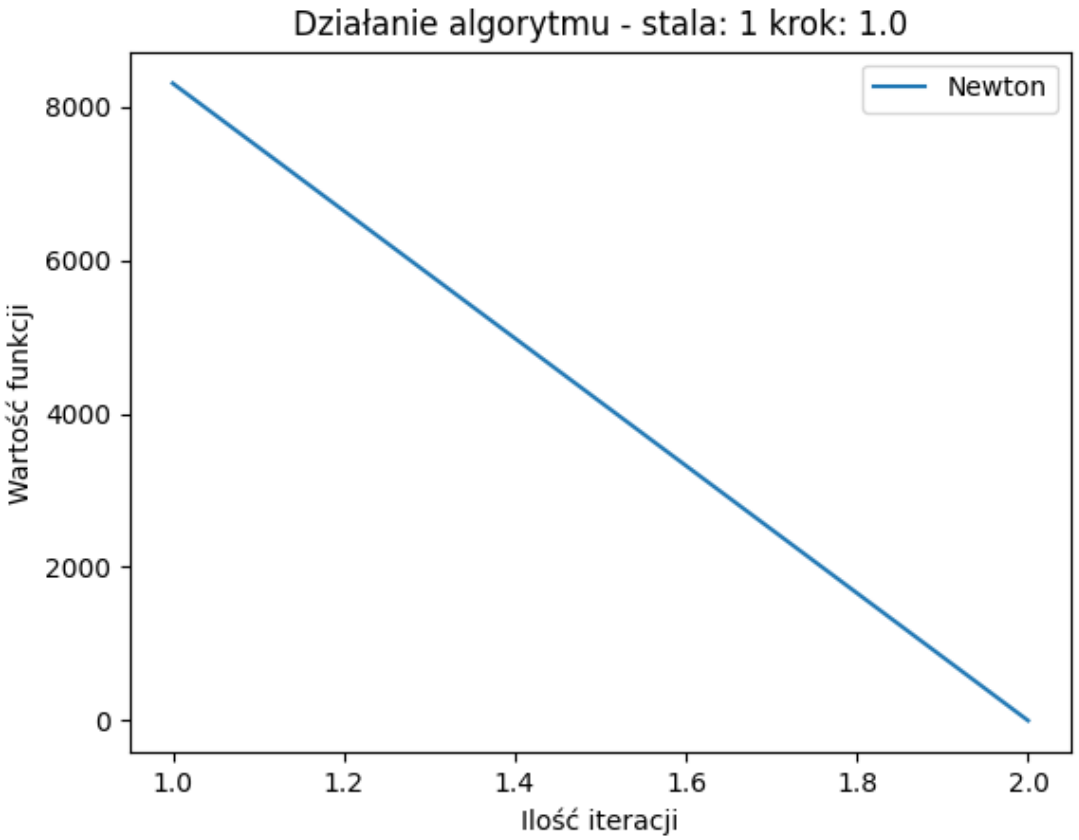


.

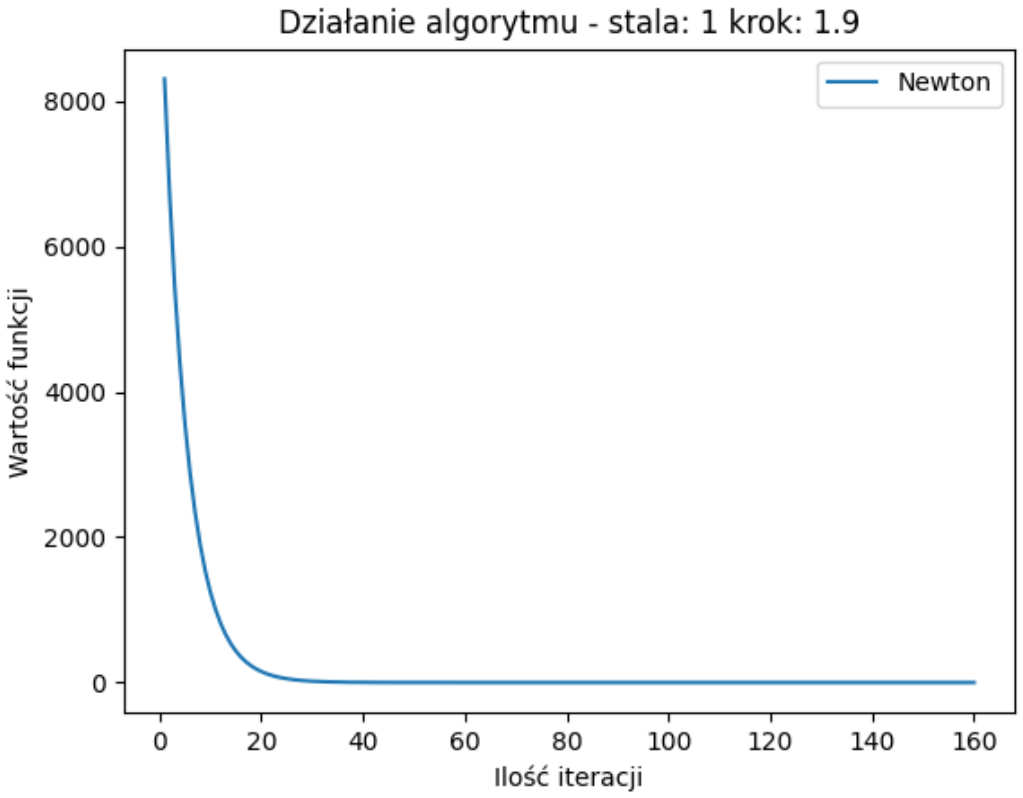
Na podstawie tego doszedłem do wniosku że dla zadanej funkcji gradient prosty działał dla następujących wartość:
Gdy stała równa jest 1 krok musi być mniejszy od 1
Gdy stała równa jest 10 krok musi być mniejszy od 0.1
Gdy stała równa jest 100 krok musi być mniejszy od 0.01
Następnie zbadałem wpływ pozycji startowej na działanie programu. Również tutaj jak uprzednio pozycja startowa nie ma znacznego wpływu na działanie samego programu.



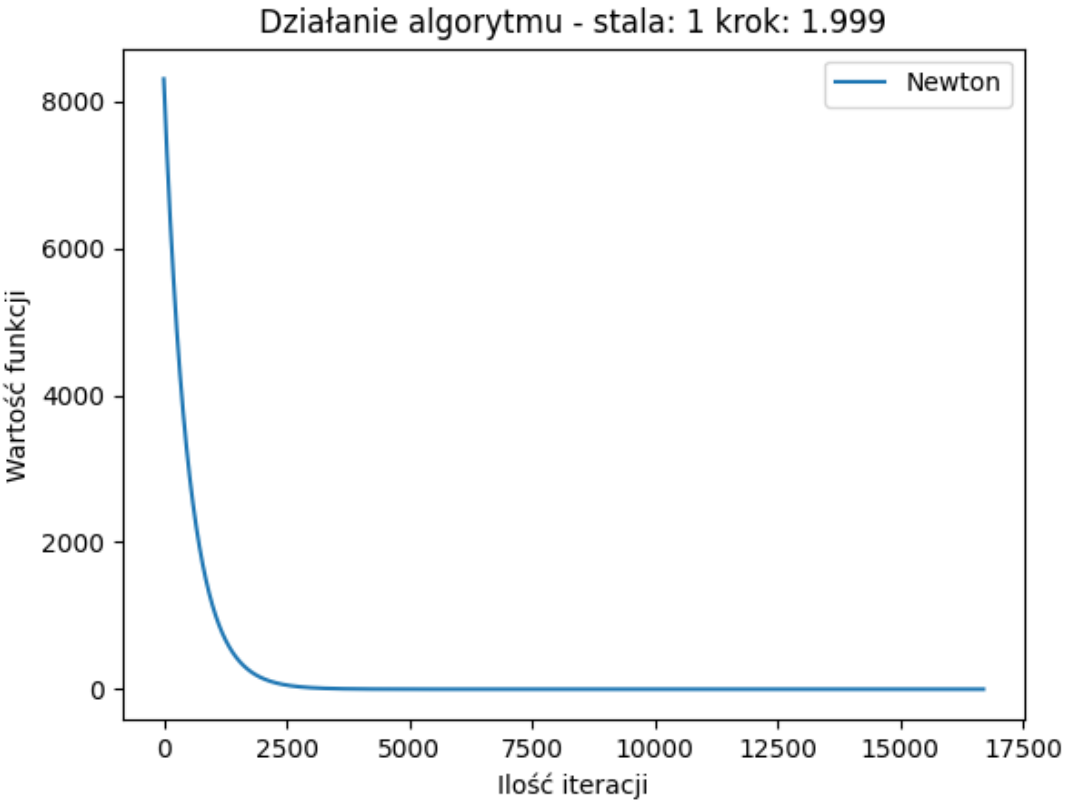
Następnie zająłem się testowaniem algorytmu Newtona. Jako krok ustaliłem 1, stała 1 oraz 10 wymiarów wektora z wartościami od 1 do 10.



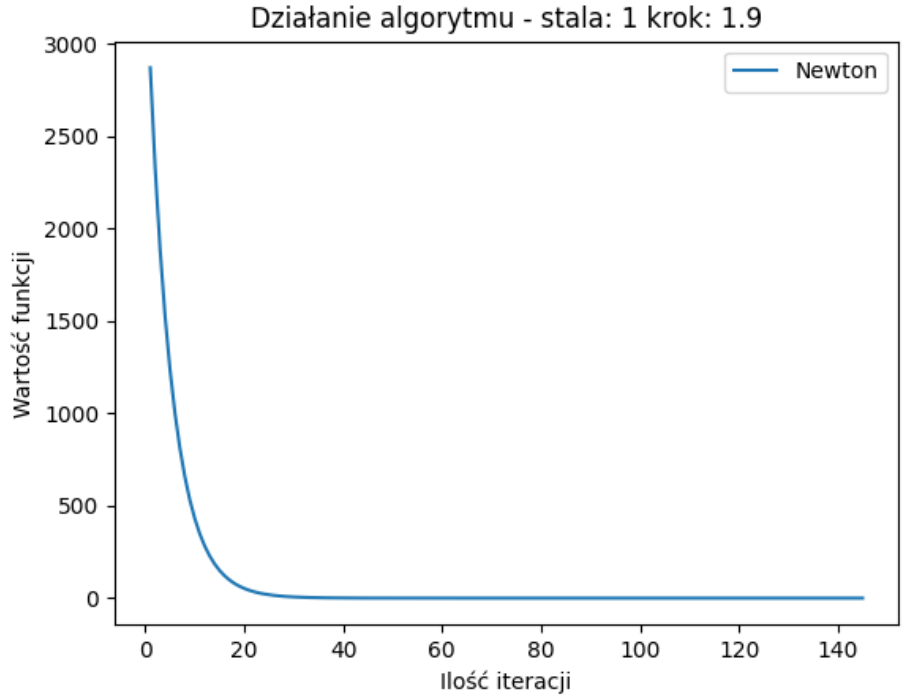
Tym razem uzyskałem ponownie znalezienie minimum w ciągu 2 iteracji. Sprawdziłem więc też większe wartości. Tym razem program wykonuje się poprawnie dla wartości kroku mniejszej od 2


Działanie algorytmu - stala: 1 krok: 1.9
Newton
Wartość funkcji
Ilość iteracji
0
2000
4000
6000
8000
0
20
40
60
80
100
120
140
160

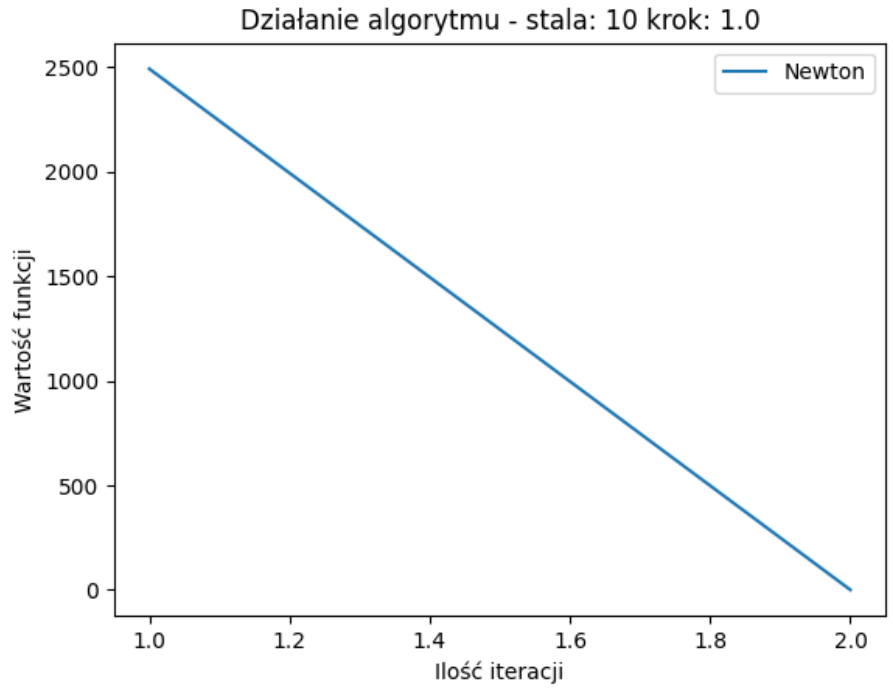
.


Działanie algorytmu - stala: 1 krok: 1.999
Newton
Wartość funkcji
Ilość iteracji
0
2000
4000
6000
8000
0
2500
5000
7500
10000
12500
15000
17500

Następnie zbadałem wpływ pozycji startowej i jej wymiaru.
Dla wektora dwudziestu wymiarów z wartościami od 1 do 20 program znowu działa bardzo podobnie, z wyłącznie wyraźną zmianą w wartości samej funkcji.

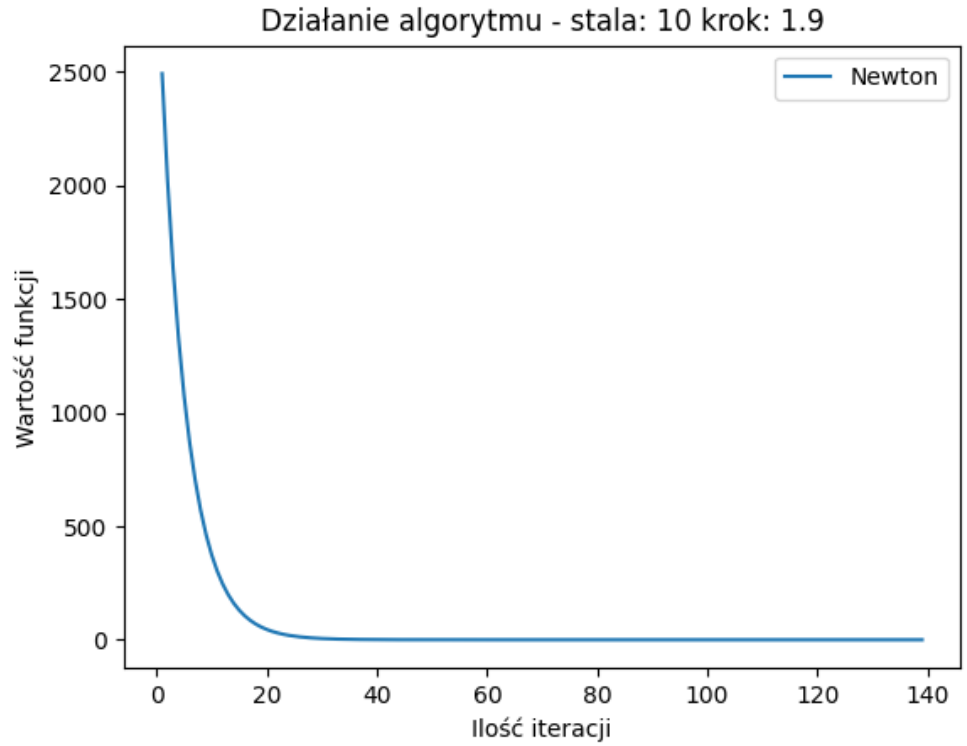


Następnie zacząłem testować dla stałej równej 10. Ponownie zaczynając od kroku 1.

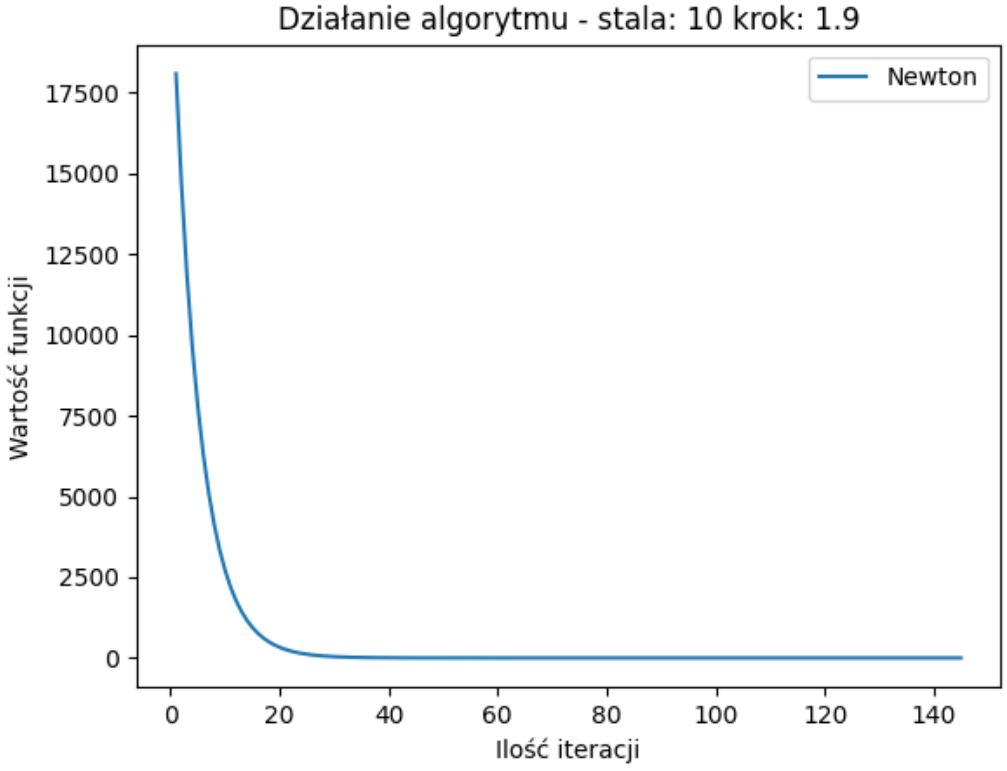


Program ponownie znalazł minimum potrzebując tylko 2 iteracji. Sprawdziłem więc czy znowu wartość 2 będzie punktem granicznym, co okazało się prawdą.

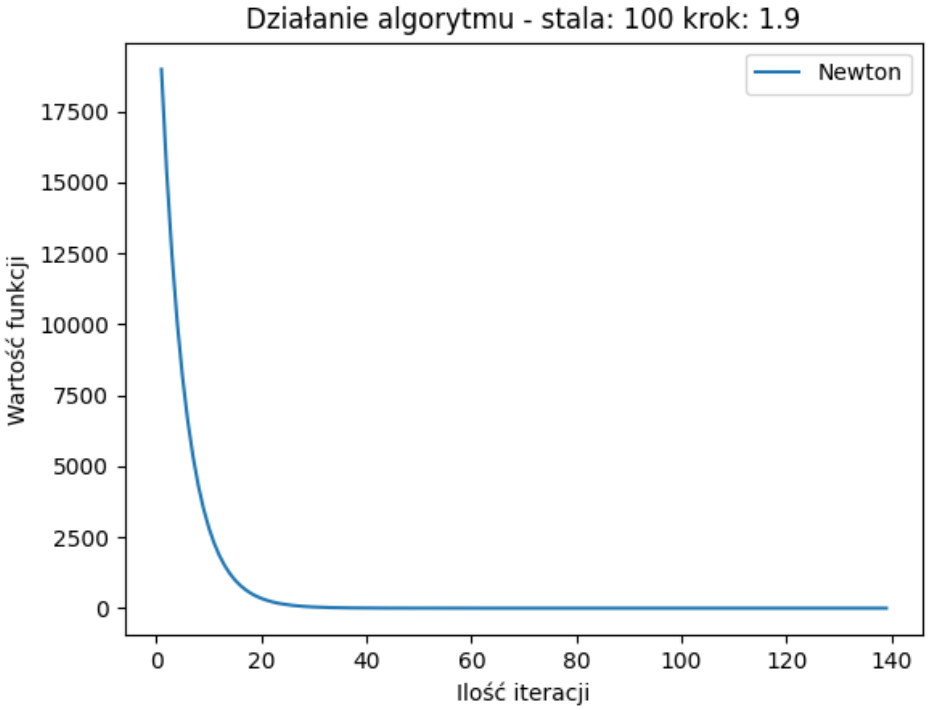
Ponownie wartości poniżej dwóch okazały się wartościami dla których program był w stanie znaleźć minimum



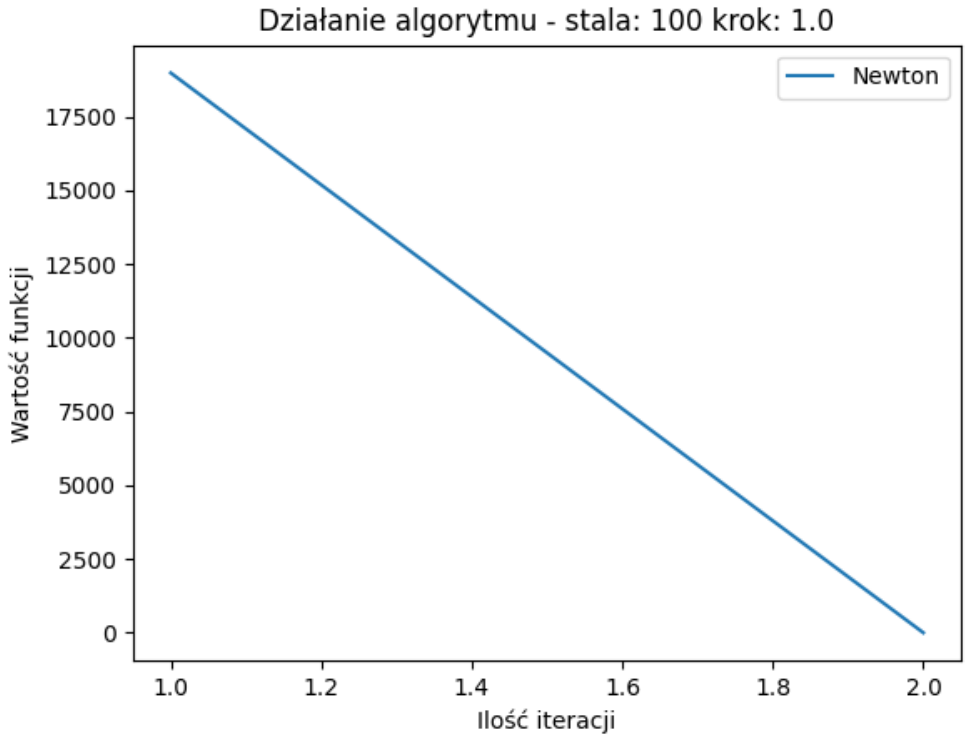
Następnie znowu sprawdziłem wpływ pozycji początkowej na działanie samego programu, tak jak w poprzednim teście wektor 20 elementów od 1 do 20. Ponownie brak wpływu na działanie samego programu.



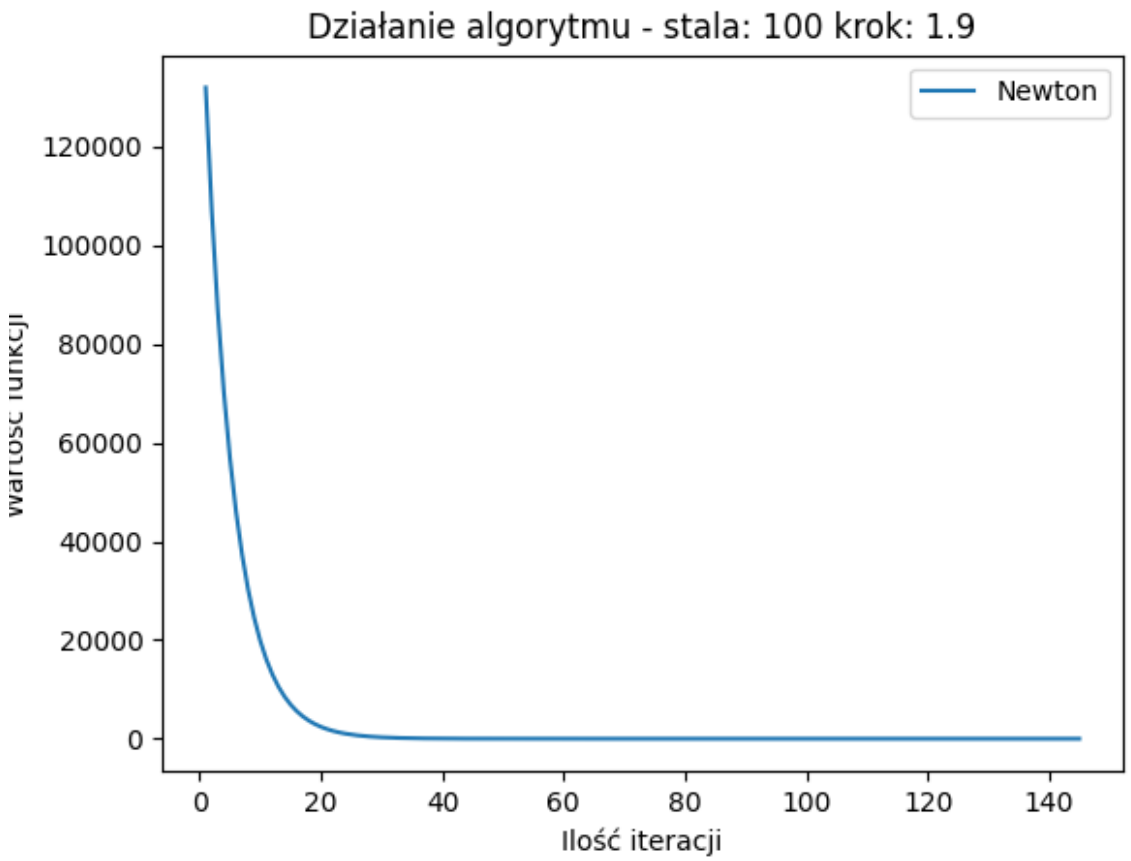
Jako ostatnie przeprowadziłem testy dla stałej równej 100. Zacząłem od badania czy i tu ponownie występuje granica przy 2.
Tak jak wcześniej program działa bez zarzutów dla wartości poniżej 2 nie mogąc znaleźć minimum w pozostałych wypadkach.



Również i tu sprawdziłem działanie dla kroku równego 1. Tożsamo z poprzednimi testami i tutaj program znalazł minimum po 2 iteracjach.



Jako ostatnie sprawdziłem wpływ pozycji startowej, ponownie wybierając taki sam wektor 20 elementów. Również i tu brak wpływu pozycji startowej na ogólne działanie programu.



Wnioski

Podsumowując pracę obu algorytmów zauważyłem następujące własności
Na temat gradientu prostego:
Algorytm dla zadanej funkcji znajduje minimum gdy:
1. stała równa jest 1 krok, wtedy musi być mniejszy od 1
2. stała równa jest 10 krok, wtedy musi być mniejszy od 0.1
3. stała równa jest 100 krok, wtedy musi być mniejszy od 0.01
Oraz dla stałej równej 1 i kroku 0.5 algorytm znajduje wynik tylko w dwóch iteracjach.
Na temat algorytmu Newtona:
Algorytm znajduje minimum niezależnie od stałej, jednak wartość kroku musi być zawsze mniejsza od dwóch. Oraz jeżeli wartość kroku równa jest 1 to ilość iteracji niezależnie od pozycji początkowej wynosić będzie 1.