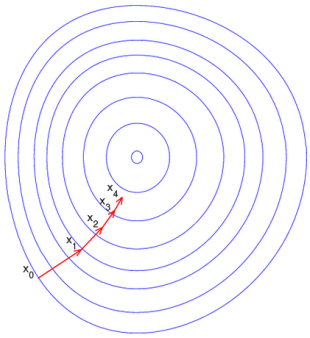
Przeszukiwanie przestrzeni w AI

Przeszukiwanie przestrzeni to koncepcja powszechnie wykorzystywana w dziedzinie sztucznej inteligencji, która odnosi się do procesu analizy i eksploracji różnych możliwości w celu znalezienia optymalnych rozwiązań. Celem przeszukiwania przestrzeni jest znalezienie najlepszej decyzji, opcji lub konfiguracji dla danego problemu lub zadania spośród wielu możliwości.

Przykładowymi algorytmami przeszukiwania przestrzeni, które zostały zaimplementowane w kodzie, to *algorytm gradientu prostego* oraz *algorytm Newtona*.

Algorytm Gradientu Prostego



Algorytm Gradientu Prostego to algorytm numeryczny, którego celem jest znalezienie *minimum lokalnego* podanej funkcji, którą określa się nazwą funkcji celu.

Gradient (∇) to wektor częściowych pochodnych, który wskazuje wejściowy kierunek, w którym wartości wyjściowe funkcji rosną najszybciej. W związku z tym, aby znaleźć maksimum funkcji należy wybrać losowy punkt początkowy, obliczyć gradient i wykonać mały krok w kierunku określonym przez gradient, a następnie powtórzyć tę operację od kolejnego punktu. Jeżeli natomiast chcemy znaleźć minimum funkcji - należy wykonać małe kroki w kierunku przeciwnym do tego określonego przez gradient.

Jeżeli funkcja ma unikalne globalne minimum, to metoda gradientu prostego je znajdzie. Jednak jeżeli funkcja ma wiele lokalnych minimów, to metoda ta może znaleźć niewłaściwe minimum, które nie będzie tak naprawdę globalnym minimum funkcji. Wtedy zaleca się wykonać algorytm kilkukrotnie, wybierając różne funkcje startowe. Jeżeli funkcja nie będzie posiadała minimum, to algorytm ten niestety może działać w nieskończoność.

Schemat algorytmu:

1. Wybierz punkt startowy x_0
2. Oblicz gradient dla badanej funkcji $\nabla f(x_k)$
3. Wykonaj mały krok w kierunku przeciwnym do określonego przez gradient: $x_{k+1} = x_k - \alpha_k \cdot \nabla f(x_k)$, gdzie α_k to współczynnik długości, dla kolejnych kroków. W wielu przypadkach przyjmuje się stałe niewielkie wielkości.

4. W celu określenia, czy punkt w danym kroku dostatecznie dobrze przybliża minimum funkcji sprawdź kryterium stopu. Można użyć następujących kryteriów stopu, gdzie ϵ to precyzja/tolerancja, jaką dopuszczamy podczas szukania minimum:
    a. $||\nabla f(x_k)|| \leq \epsilon$
    b. $||x_{k+1} - x_k|| \leq \epsilon$
5. Jeżeli $f(x_{k+1}) \geq f(x_k)$ to zmniejsz wartość α_k i powtórz punkt 2 dla kroku k-tego.
6. Powtórz punkt 2 dla następnego kroku (k+1).

Algorytm Newtona

Algorytm Newtona to również algorytm numeryczny, którego celem jest znalezienie *minimum lokalnego* podanej funkcji, którą określa się nazwą funkcji celu. Jednak jest on bardziej zaawansowany niż gradient prosty, ze względu na wykorzystywanie drugiej pochodnej funkcji celu. Pomaga szybciej zbiegać do rozwiązania, ponieważ uwzględnia również informacje o krzywiźnie funkcji.

Hessjan (H) to macierz drugich pochodnych cząstkowych funkcji. Pozwala na analizę krzywizn i kształtu funkcji w przestrzeni.

Schemat algorytmu:

1. Wybierz punkt startowy x_0
2. Oblicz hesjan dla badanej funkcji $Hf(x_k)$
3. Oblicz odwroność hasjanu pomnożoną przez gradient: $d_k = (Hf(x_k))^{-1} \cdot \nabla f(x_k)$
4. Wykonaj mały krok w kierunku przeciwnym do określonego przez wartość obliczoną w punkcie 3: $x_{k+1} = x_k - \alpha_k \cdot d_k$, gdzie α_k to współczynnik długości, dla kolejnych kroków. W wielu przypadkach przyjmuje się stałe niewielkie wielkości.
7. W celu określenia, czy punkt w danym kroku dostatecznie dobrze przybliża minimum funkcji sprawdź kryterium stopu. Można użyć następujących kryteriów stopu, gdzie ϵ to precyzja/tolerancja, jaką dopuszczamy podczas szukania minimum:
    a. $||\nabla f(x_k)|| \leq \epsilon$
    b. $||x_{k+1} - x_k|| \leq \epsilon$
8. Jeżeli $f(x_{k+1}) \geq f(x_k)$ to zmniejsz wartość α_k i powtórz punkt 2 dla kroku k-tego.
9. Powtórz punkt 2 dla następnego kroku (k+1).

Implementacja algorytmów

Treść zadania

Zaimplementuj algorytm gradientu prostego oraz algorytm Newtona. Algorytm Newtona powinien działać w dwóch trybach:

- Ze stałym parametrem kroku
- adaptacją parametru kroku przy użyciu metody z nawrotami

Zbadaj zbieżność obu algorytmów, używając następującej funkcji:

$$f(x) = \sum_{i=1}^{n} \alpha^{\frac{i-1}{n-1}} x_i^2, \; x \in \mathbb{R}^n$$

Zbadaj wpływ wartości parametru kroku na zbieżność obu metod. W swoich badaniach rozważ następujące wartości parametru $\alpha \in \{1, 10, 100\}$ oraz dwie wymiarowości $n \in \{10, 20\}$. Porównaj czas działania obu algorytmów.

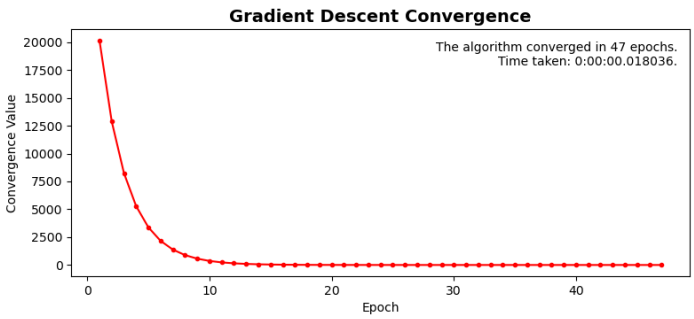
Wyniki

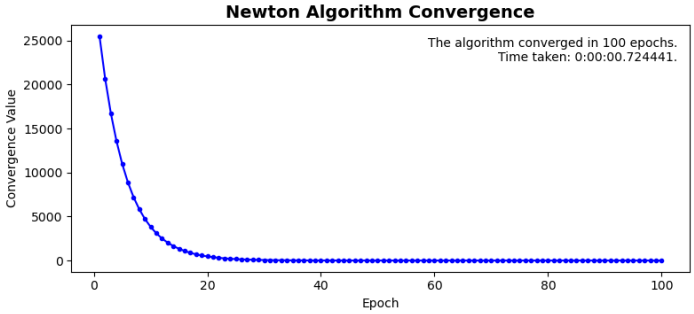
Stałe wartości:

- Tolerancja: precision = 0.01
- Maksymalna liczba iteracji: epoch_limit = 1000

1. Wyniki dla wymiarowości n = 10.
   Wektor wejściowy: theta = [76, -5, 96, 18, -43, -63, 54, 71, 48, 7]

a) Parameter: α = 1.
   Współczynnik długości, dla kolejnych kroków: α_k = 0.1.





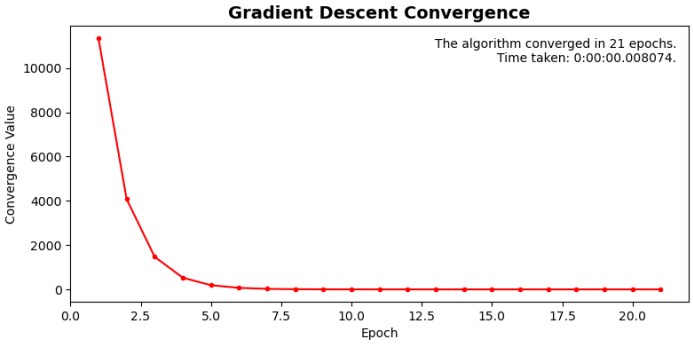
Gradient Prosty:

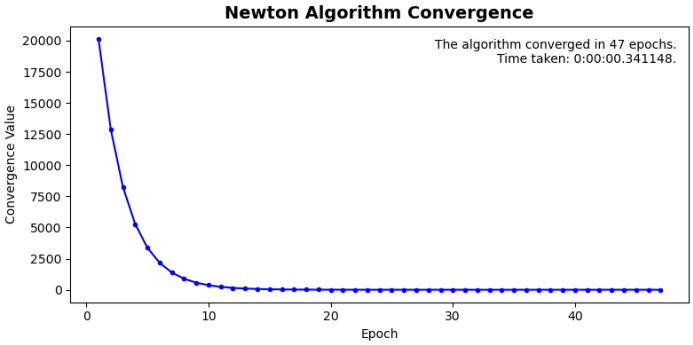
- Liczba iteracji: 47
- Czas: ~18ms

Algorytm Newtona:

- Liczba iteracji: 100
- Czas: ~724ms

b) Parameter: $\alpha = 1$.
Współczynnik długości, dla kolejnych kroków: $\alpha_k = 0.2$.

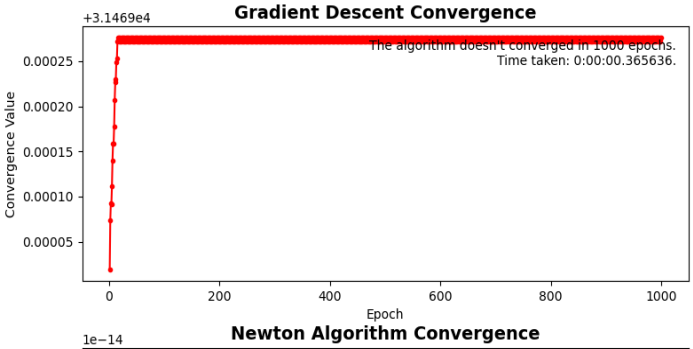


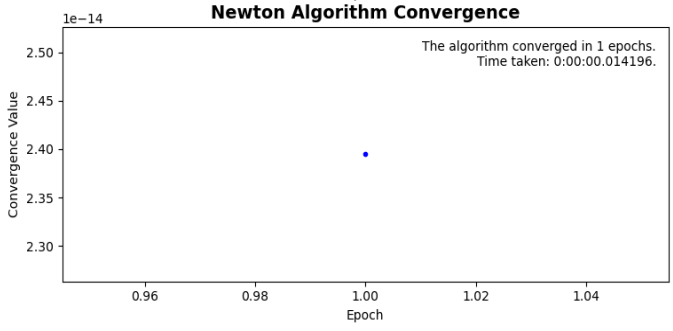


Gradient Prosty:
- Liczba iteracji: 21
- Czas: ~8ms

Algorytm Newtona:
- Liczba iteracji: 47
- Czas: ~341ms

c) Parameter: $\alpha = 1$.
Współczynnik długości, dla kolejnych kroków: $\alpha_k = 1$.





Gradient Prosty:
- Liczba iteracji: algorytm nie znalazł minimum, przy maksymalnej iteracji 1000
- Czas: ~365ms

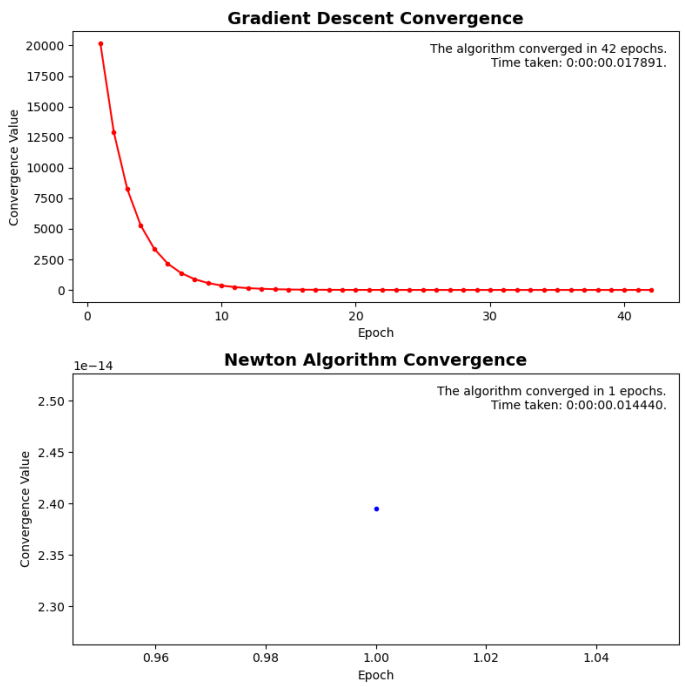
Algorytm Newtona:
- Liczba iteracji: 1
- Czas: ~15ms

Algorytm gradientu prostego dla stałej wartości kroku α_k = 1 może napotkać trudności w znajdowaniu minimum lokalnego lub globalnego. Jest to spowodowane tym, że duży krok może prowadzić do oscylacji wokół minimum lub nawet "przeskoczenia" minimum.

Algorytm Newtona, który jest bardziej zaawansowaną metodą optymalizacji, może znaleźć minimum po zaledwie jednym kroku, ponieważ wykorzystuje informacje o drugiej pochodnej funkcji celu, co pozwala na bardziej precyzyjne i szybkie kierowanie się w kierunku minimum.

Aby rozwiązać problem zbyt dużego kroku w algorytmie gradientu prostego, można zastosować technikę ***adaptacyjnego zmniejszania kroku***. W praktyce, w trakcie uczenia, można dynamicznie dostosowywać wartość kroku α_k w zależności od postępu optymalizacji. To pozwala na większą stabilność procesu optymalizacji i może przyspieszyć osiągnięcie minimum. Istnieje odpowiedni warunek stopu, który można zastosować w trakcie optymalizacji, aby kontrolować i regulować wartość kroku. Warunek ten można sformułować jako $y_{k+1} - y_k < step\_condition$, gdzie $step\_condition$ jest ustalonym progiem zmiany funkcji celu, w moim przypadku jest to 0.0001. Jeśli ten warunek jest spełniony, oznacza to, że proces optymalizacji jest stabilny, a wartość kroku może zostać odpowiednio zmniejszona, na przykład o 10%, aby uniknąć zbyt dużych kroków.

Początkowy współczynnik długości: α_k = 1 - zmniejszanie o 10% jeśli zostanie spełniony warunek:



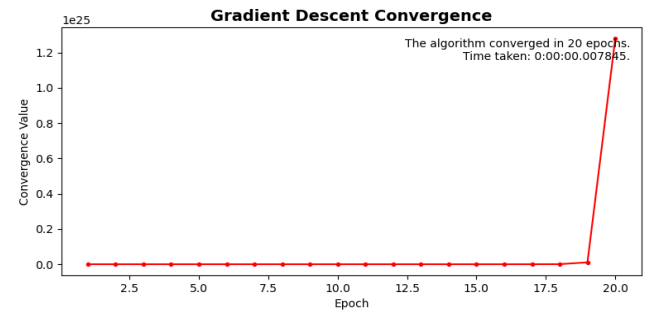
Gradient Prosty:

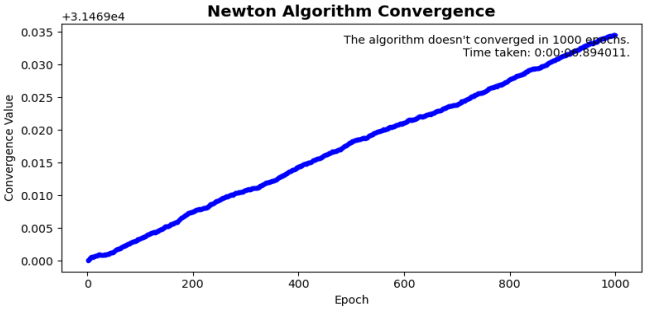
- Liczba iteracji iteracji: 42
- Czas: ~18ms

Algorytm Newtona:

- Liczba iteracji: 1
- Czas: ~15ms

d) Parameter: α = 1.
Współczynnik długości, dla kolejnych kroków: α_k = 2.





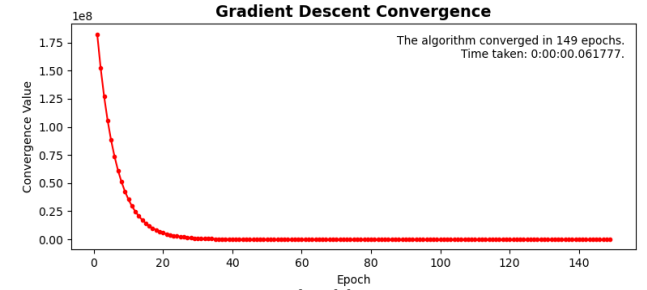
Gradient Prosty:

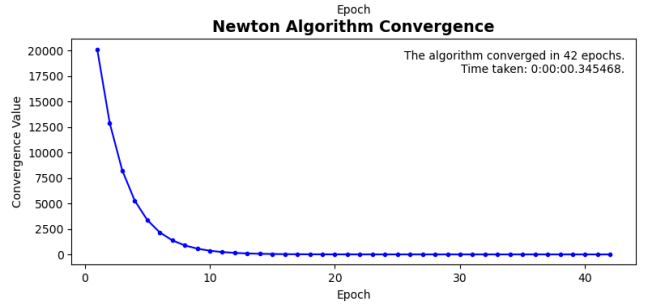
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 20 iteracjach
- Czas: ~7ms

Algorytm Newtona:

- Liczba iteracji: algorytm nie znalazł minimum, przy maksymalnej iteracji 1000
- Czas: ~894ms

Zarówno algorytm gradientu prostego jak i algorytm Newtona nie poradził sobie ze stałą wartością kroku α_k = 2. Algorytm Newtona nie znalazł minimum, przy ustawionej maksymalnej ilości 1000 iteracji, natomiast dla algorytmu gradientu prostego wystąpił nagły wzrost wartości funkcji kosztu w ostatniej iteracji, co sugeruje, że ustawienie zbyt dużego współczynnika kroku spowodowało, że algorytm "skoczył" z obszaru minimum lokalnego do obszaru, w którym funkcja zmienia się znacznie szybciej. Ten problem związany z niestabilnością numeryczną może prowadzić do błędnych wyników. W związku z tym, tutaj również został zastosowany krok adaptacyjny, który dostosowuje rozmiar kroku podczas optymalizacji, co pomaga uniknąć ekstremalnych skoków i zapewnia stabilniejszą zbieżność algorytmu.
Początkowy współczynnik długości: α_k = 2 - zmniejszanie o 10% jeśli zostanie spełniony odpowiedni warunek:





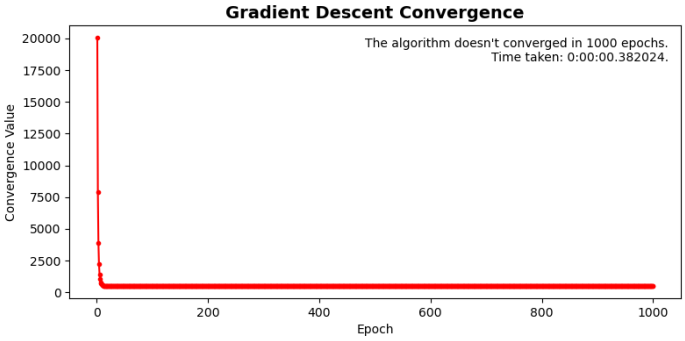
Gradient Prosty:

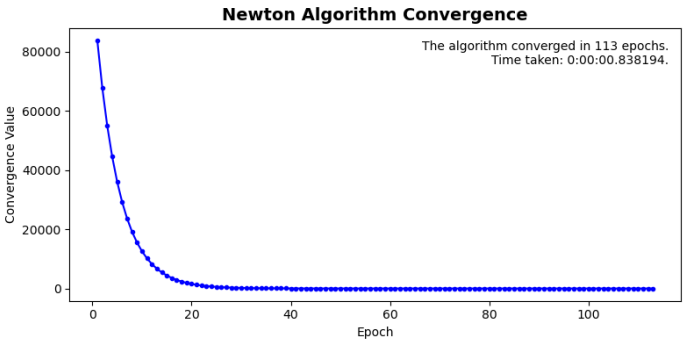
- Liczba iteracji iteracji: 149
- Czas: ~61ms

Algorytm Newtona:

- Liczba iteracji: 42
- Czas: ~345ms

e) Parameter: α = 10.
Współczynnik długości, dla kolejnych kroków: α_k = 0.1.





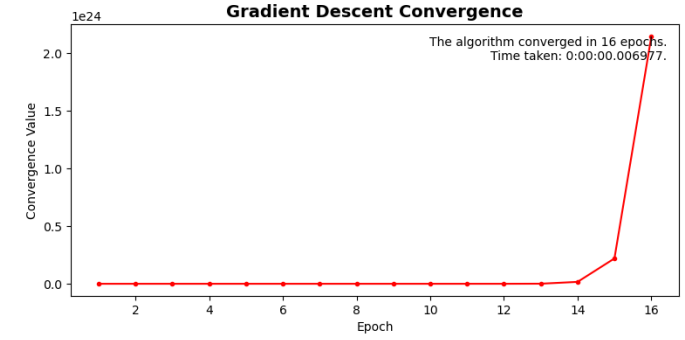
Gradient Prosty:

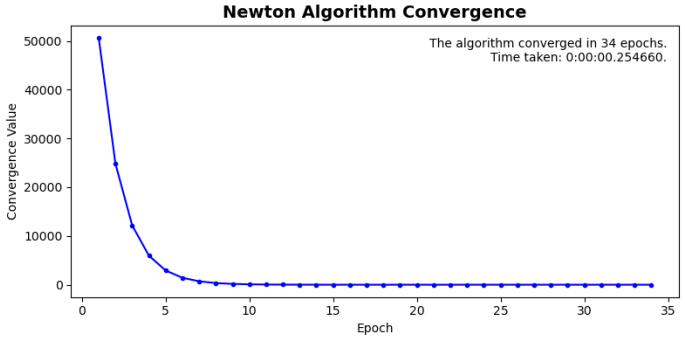
- Liczba iteracji: algorytm nie znalazł minimum, przy maksymalnej iteracji 1000
- Czas: ~382ms

Algorytm Newtona:

- Liczba iteracji: 113
- Czas: ~838ms

f) Parameter: α = 10.
Współczynnik długości, dla kolejnych kroków: α_k = 0.3.





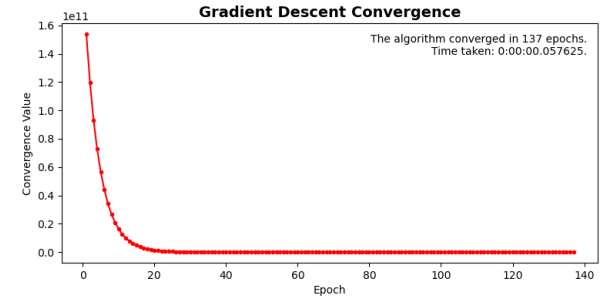
Gradient Prosty:

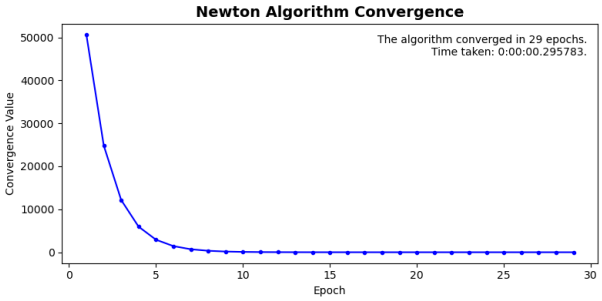
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 16 iteracjach
- Czas: ~7ms

Algorytm Newtona:

- Liczba iteracji: 34
- Czas: ~255ms

g) Parameter: α = 10.
Początkowy współczynnik długości: α_k = 0.3 - zmniejszanie o 10% jeśli zostanie spełniony odpowiedni warunek.

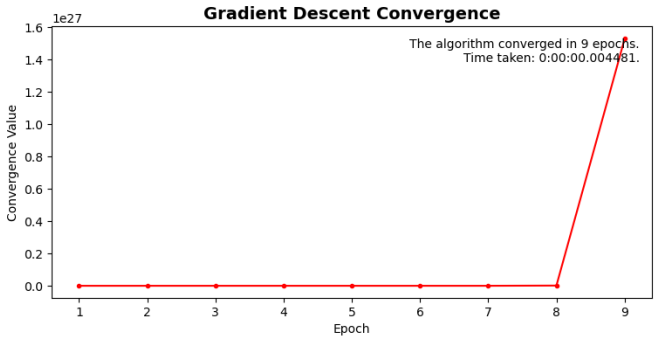


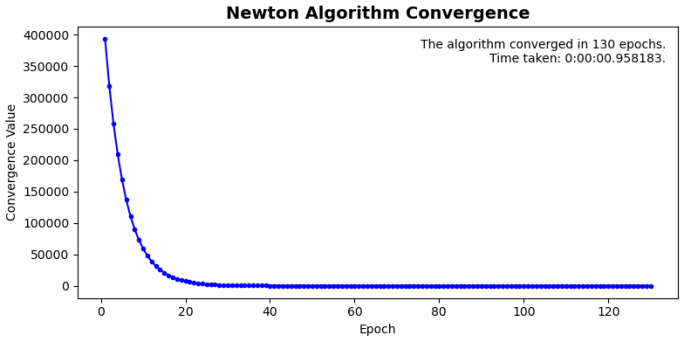


Gradient Prosty:
- Liczba iteracji iteracji: 137
- Czas: ~58ms

Algorytm Newtona:
- Liczba iteracji: 29
- Czas: ~296ms

h) Parameter: α = 100.
Współczynnik długości, dla kolejnych kroków: α_k = 0.1.

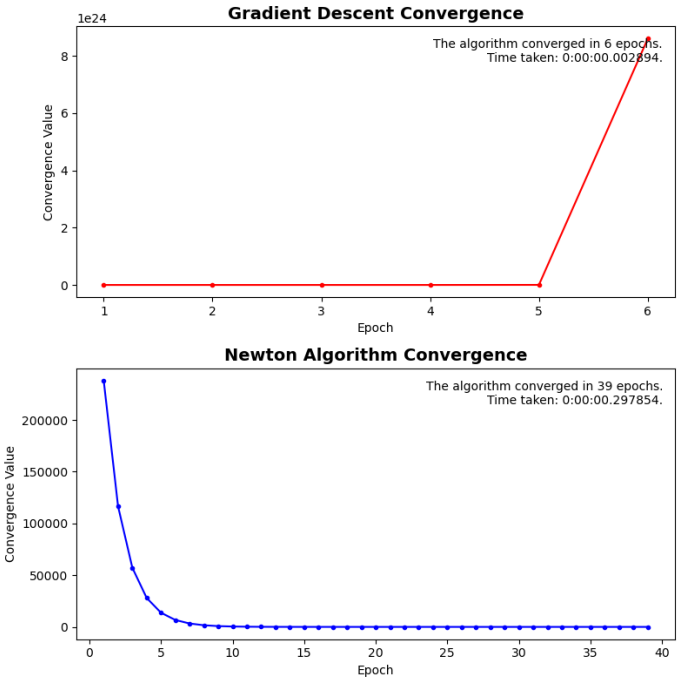




Gradient Prosty:
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 9 iteracjach
- Czas: ~4ms

Algorytm Newtona:
- Liczba iteracji: 130
- Czas: ~958ms

i) Parameter: α = 100.
Współczynnik długości, dla kolejnych kroków: α_k = 0.3.



Gradient Prosty:

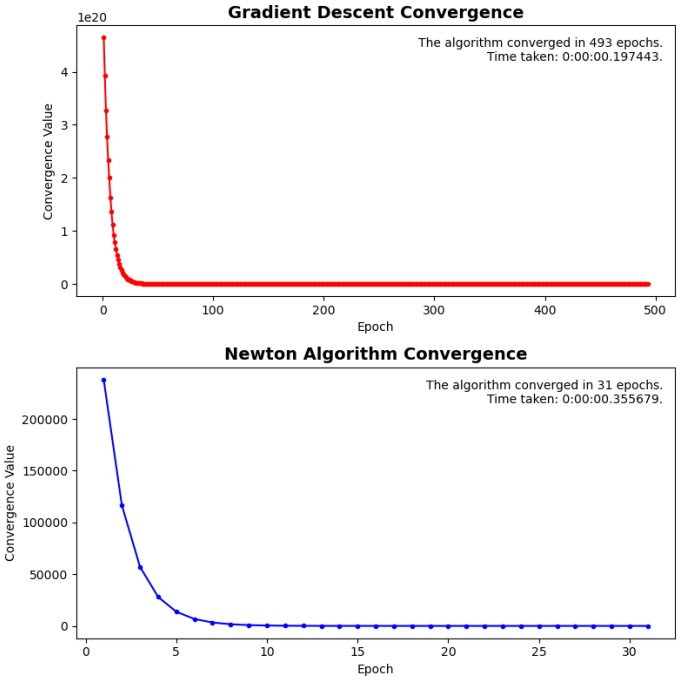
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 6 iteracjach
- Czas: ~3ms

Algorytm Newtona:

- Liczba iteracji: 39
- Czas: ~298ms

j) Parameter: α = 100.
Początkowy współczynnik długości: α_k = 0.3

Dopiero w momencie, kiedy step_condition został zmieniony na 0.000001 oraz krok adaptacyjny ustawiony na 35% algorytm gradientu prostego znalazł minimum. W algorytmie Newtona natomiast zostało zastosowane zmniejszanie o 10% jeśli zostanie spełniony odpowiedni warunek.



Gradient Prosty:

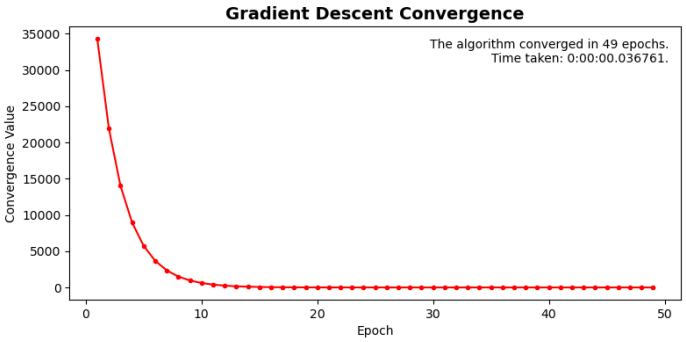
- Liczba iteracji iteracji: 493
- Czas: ~197ms

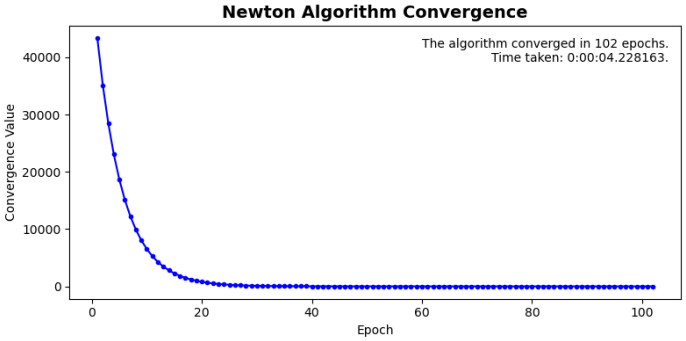
Algorytm Newtona:

- Liczba iteracji: 31
- Czas: ~355ms

2. Wyniki dla wymiarowości n = 20.
   Wektor wejściowy: theta = [-3, 84, -11, -57, 84, -15, 56, 69, 70, -57, -28, -80, 10, 20, 73, -28, -28, -44, -54, -19]

a) Parameter: α = 1.
   Współczynnik długości, dla kolejnych kroków: α_k = 0.1.

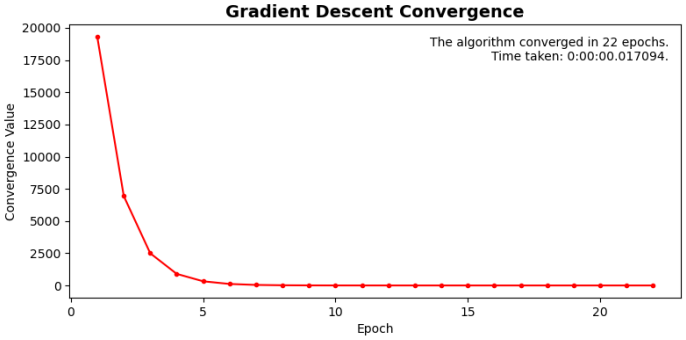


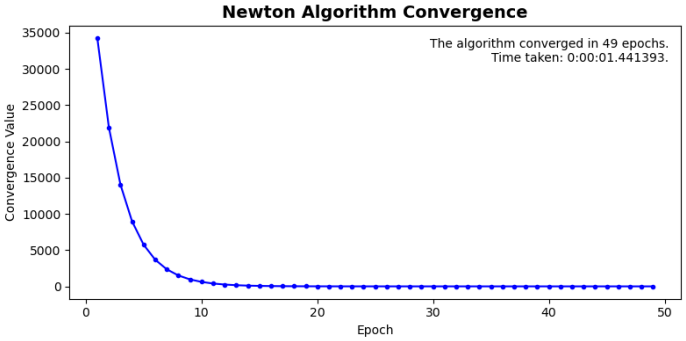


Gradient Prosty:
- Liczba iteracji: 49
- Czas: ~37ms

Algorytm Newtona:
- Liczba iteracji: 102
- Czas: ~4228ms

b) Parameter: α = 1.
   Współczynnik długości, dla kolejnych kroków: α_k = 0.2.

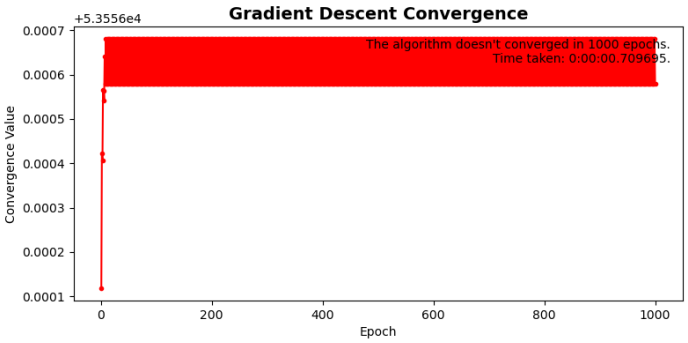


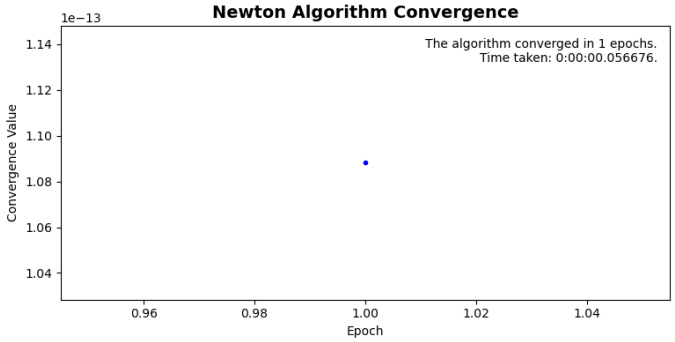


Gradient Prosty:
- Liczba iteracji: 22
- Czas: ~17ms

Algorytm Newtona:
- Liczba iteracji: 49
- Czas: ~1441ms

c) Parameter: $\alpha = 1$.
Współczynnik długości, dla kolejnych kroków: $\alpha_k = 1$.

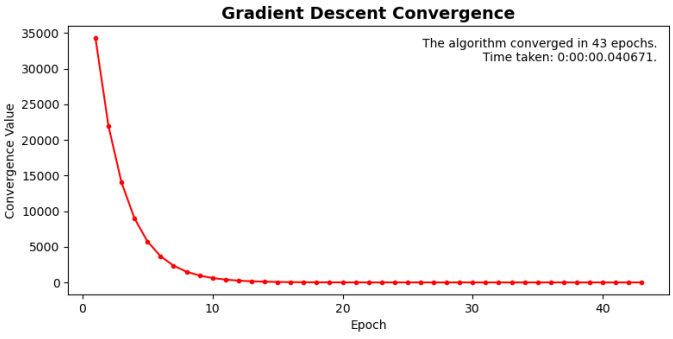


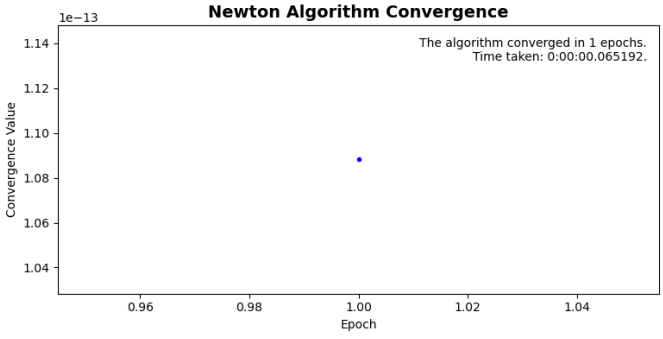


Gradient Prosty:
- Liczba iteracji: algorytm nie znalazł minimum, przy maksymalnej iteracji 1000
- Czas: ~710ms

Algorytm Newtona:
- Liczba iteracji: 1
- Czas: ~57ms

d) Parameter: $\alpha = 1$.
Współczynnik długości, dla kolejnych kroków: $\alpha_k = 1$ - zmniejszanie o 10% jeśli zostanie spełniony warunek:

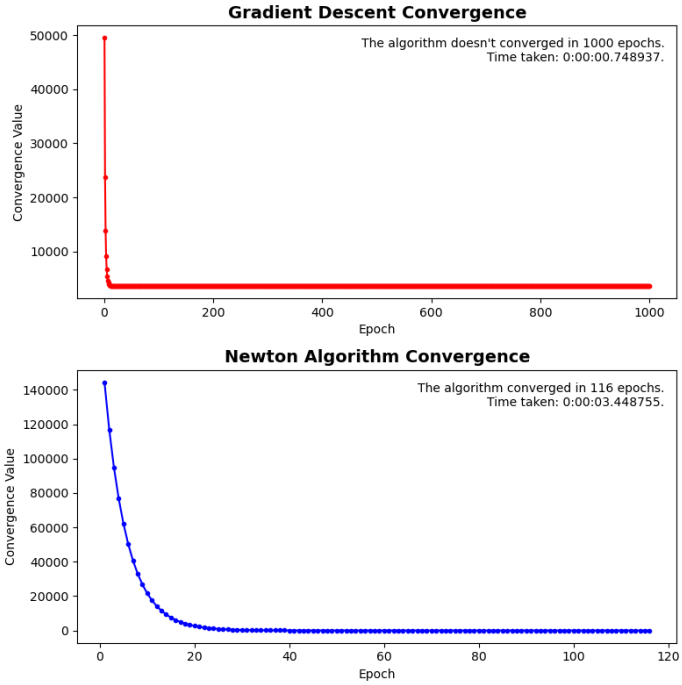




Gradient Prosty:
- Liczba iteracji: 43
- Czas: ~41ms

Algorytm Newtona:
- Liczba iteracji: 1
- Czas: ~65ms

e) Parameter: $\alpha = 10$.
Współczynnik długości, dla kolejnych kroków: $\alpha_k = 0.1$.



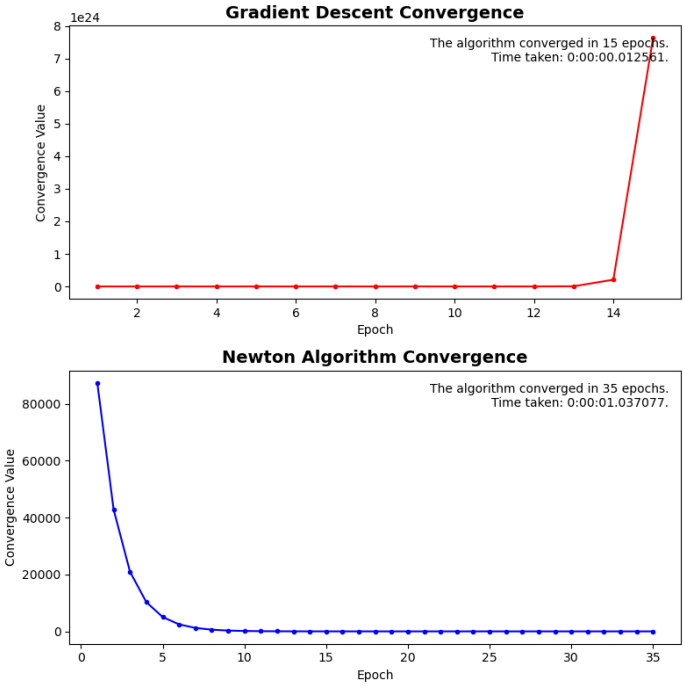
Gradient Prosty:

- Liczba iteracji: algorytm nie znalazł minimum, przy maksymalnej iteracji 1000
- Czas: ~749ms

Algorytm Newtona:

- Liczba iteracji: 116
- Czas: ~3448ms

f) Parameter: $\alpha = 10$.
Współczynnik długości, dla kolejnych kroków: $\alpha_k = 0.3$.



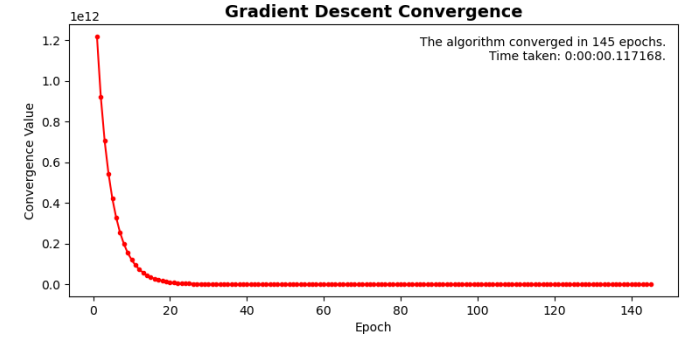
Gradient Prosty:

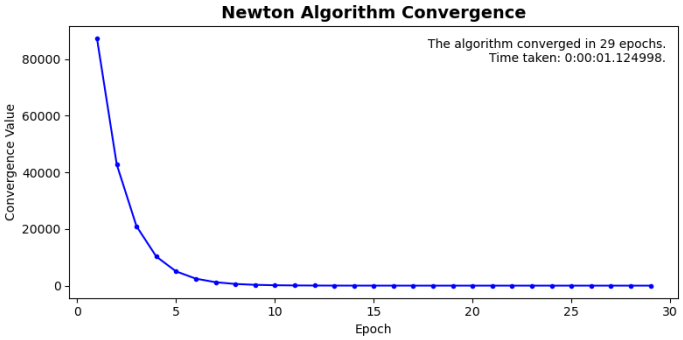
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 15 iteracjach
- Czas: ~13ms

Algorytm Newtona:

- Liczba iteracji: 35
- Czas: ~1037ms

g) Parameter: α = 10.
Początkowy współczynnik długości: α_k = 0.3 - zmniejszanie o 10% jeśli zostanie spełniony odpowiedni warunek.

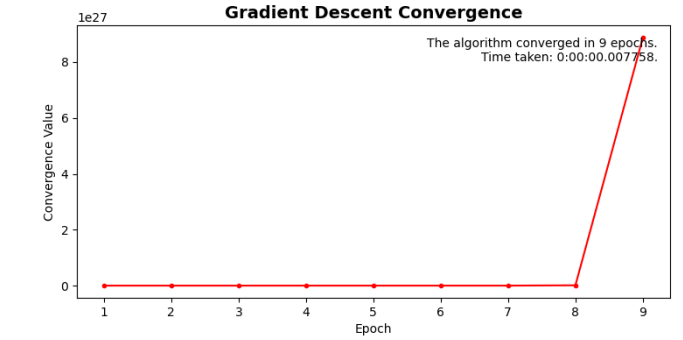


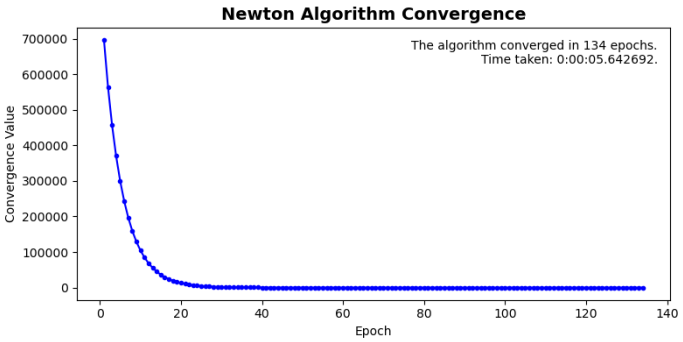


Gradient Prosty:
- Liczba iteracji iteracji: 145
- Czas: ~117ms

Algorytm Newtona:
- Liczba iteracji: 29
- Czas: ~1250ms

h) Parameter: α = 100.
Współczynnik długości, dla kolejnych kroków: α_k = 0.1.

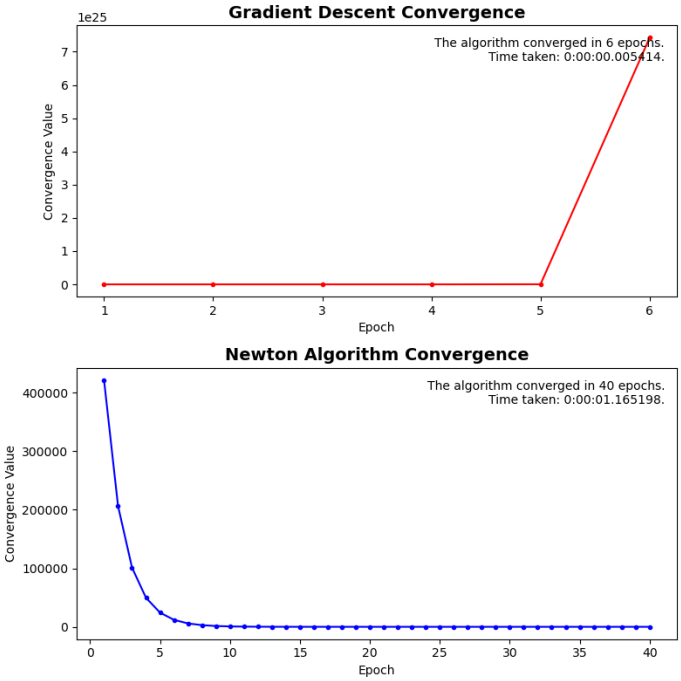




Gradient Prosty:
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 9 iteracjach
- Czas: ~8ms

Algorytm Newtona:
- Liczba iteracji: 134
- Czas: ~563ms

i) Parameter: α = 100.
Współczynnik długości, dla kolejnych kroków: α_k = 0.3.



Gradient Prosty:

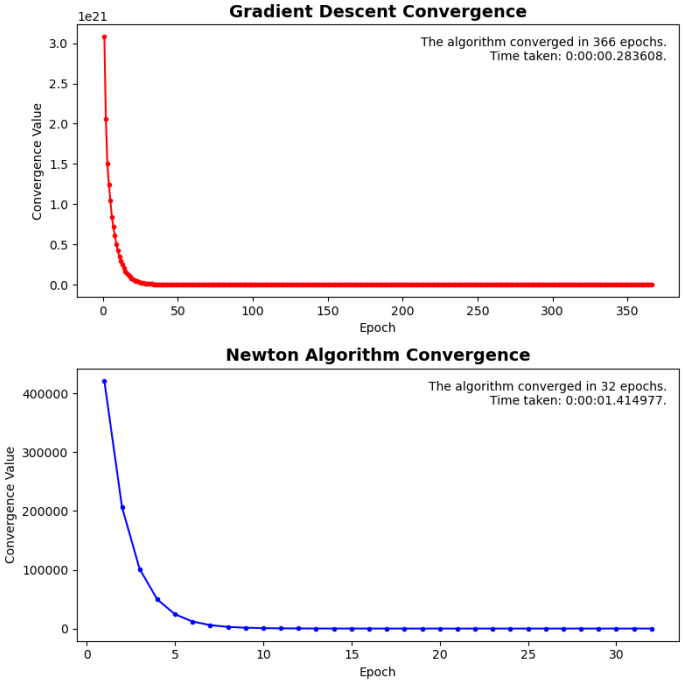
- Liczba iteracji: algorytm nie znalazł minimum, pomimo tego, że zakończył swoje działanie po 6 iteracjach
- Czas: ~5ms

Algorytm Newtona:

- Liczba iteracji: 40
- Czas: ~1165ms

j) Parameter: α = 100.
Początkowy współczynnik długości: α_k = 0.3

Dopiero w momencie, kiedy step_condition został zmieniony na 0.000001 oraz krok adaptacyjny ustawiony na 35% algorytm gradientu prostego znalazł minimum. W algorytmie Newtona natomiast zostało zastosowane zmniejszanie o 10% jeśli zostanie spełniony odpowiedni warunek.



Gradient Prosty:

- Liczba iteracji iteracji: 366
- Czas: ~284ms

Algorytm Newtona:

- Liczba iteracji: 32
- Czas: ~1415ms

Wnioski

1. Wybór algorytmu optymalizacji (algorytm gradientu prostego, czy algorytm Newtona) powinien zależeć od konkretnej charakterystyki problemu. Metoda Newtona może zbiegać szybciej, zwłaszcza w przypadku funkcji o wyraźnym kształcie i odpowiednich warunków początkowych, ale może być bardziej wymagająca obliczeniowo, szczególnie w przypadku problemów o większej ilości wymiarów.

2. Współczynnik uczenia (α_k) ma kluczowe znaczenie w algorytmie gradientu prostego. Odpowiedni współczynnik uczenia może znacząco wpłynąć na zbieżność. Dynamiczna dostosowywanie współczynnika uczenia w oparciu o warunki kroków jest korzystne w wielu przypadkach.

3. Mniejszy warunek kroku może prowadzić do bardziej precyzyjnej zbieżności, ale może wymagać większej liczby iteracji i czasu. Ważne jest znalezienie odpowiedniej równowagi między warunkiem kroku, a wydajnością algorytmu.

4. Dla niektórych problemów algorytm gradientu prostego korzysta z adaptacyjnych rozmiarów kroków, podczas gdy dla innych stały współczynnik uczenia może działać lepiej. Przeprowadzenie eksperymentów jest kluczowe dla znalezienia właściwego podejścia.

5. Początkowy współczynnik uczenia (α_k) może wpłynąć na wydajność algorytmu optymalizacji. Przeprowadzanie eksperymentów z różnymi początkowymi wartościami jest pomocne w znalezieniu odpowiedniego punktu wyjścia.

6. W przypadku trudnych problemów optymalizacji dla algorytmu gradientu prostego eksperymentowanie z adaptacyjnymi rozmiarami kroków i zmniejszanie warunkiem tego kroku może być skuteczne w osiągnięciu zbieżności.

7. Zwiększenie wymiarowości problemu optymalizacji prowadzi do wydłużenia czasu obliczeń, szczególnie w przypadku metody Newtona, która jest bardziej kosztowna obliczeniowo. Mimo to, zmiany wartości funkcji celu przy wzroście wymiarowości były stosunkowo niewielkie, co sugeruje, że osiągnięcie dokładnego minimum jest trudniejsze i bardziej czasochłonne w przypadku większej ilości wymiarów. Wybór odpowiedniego algorytmu optymalizacji staje się kluczowy w przypadku problemów o większej ilości wymiarów.