Wprowadzenie do sztucznej inteligencji - ćwiczenie nr 1

Jan Górski

13 marca 2024

Spis treści

1 Zadania

1.1 Zadanie nr 1

Mamy problem plecakowy:

```
m = np.array([8, 3, 5, 2]) #masa przedmiotów
M = np.sum(m)/2 #niech maksymalna masa plecaka
# będzie równa połowie masy przedmiotów
p = np.array([16, 8, 9, 6]) #wartość przedmiotów
```

1. Znaleźć rozwiązanie optymalne przez przegląd wyczerpujący (analizując wszystkie kombinacje).
2. Rozwiązać problem przy użyciu heurystyki: do plecaka pakujemy przedmioty według kolejności wynikającej ze stosunku p/m. Przeglądamy listę przedmiotów do końca, chyba że plecak jest już pełen lub zostało w nim tak mało miejsca, że już na pewno nic się nie zmieści. Uwaga: heurystyka to nie funkcja heurystyczna. Nie używamy tu jeszcze funkcji heurystycznej i algorytmu A*.

Pytania:

1. Jakie rozwiązania i jaką wartość funkcji oceny uzyskano? Czy uzyskano takie same rozwiązania?
2. Jak dużą instancję problemu (liczba przedmiotów) da się rozwiązać w około minutę metodą przeglądu wyczerpującego? Ile czasu zajmie rozwiązanie tego problemu metodą zachłanną (używając heurystyki)? Odpowiednio długie wektory m i p należy wylosować, M = np.sum(m)/2.
3. Jak bardzo wydłuży obliczenia dodanie jeszcze jednego przedmiotu?
4. Jakie wnioski można wyciągnąć na podstawie wyników tego ćwiczenia?

Uwagi:

1. Nie używać rekurencji.
2. Kod którego czas wykonania mierzymy nie powinien wypisywać niczego na konsolę.
3. Nie pytałem o teoretyczną złożoność obliczeniową. Aby odpowiedzieć na pytania należy wykonać stosowne eksperymenty.
4. Czas działania programu można zgrubnie zmierzyć komendą "time", po której podaje się nazwę badanego programu.

1.2 Zadanie nr 2

- Zaimplementować metodę najszybszego wzrostu (minimalizacja, spodziewam się stałego współczynnika kroku, jeśli jednak ktoś chce zrobić więcej i zastosować zmienny współczynnik to ma taką możliwość). Gradient wyliczamy numerycznie.
- Narysować zachowanie algorytmu (kolejne kroki algorytmu jako strzałki na tle poziomic funkcji celu). Uwaga: w praktycznych zadaniach optymalizacji nie da się narysować funkcji celu ponieważ zadania mają wiele wymiarów (np. 100), oraz koszt wyznaczenia oceny jednego punktu jest duży.
- Zastosować metodę do znalezienia optimum funkcji booth w 2 wymiarach, po czym do znalezienia optimum funkcji o numerach od 1 do 3 z CEC 2017 w 10 wymiarach (na wykresie narysować kroki w wybranych 2 wymiarach z 10). Ograniczenia kostkowe przestrzeni to $[-100, 100]$. Uwzględnianie ograniczeń przez rzutowanie, np. dla $x > 100$ $x = 100$. Uwaga: wszystkie funkcje są unimodalne. Funkcje CEC są bardzo trudne, nie należy spodziewać się znalezienia optimum dla wszystkich. Należy jednak podjąć próby (z różnymi ustawieniami parametru beta).

W sprawozdaniu należy zawrzeć wykresy uzyskane stworzonym oprogramowaniem (np. po 3 dla każdej funkcji, dla różnych punktów startowych). Należy podać wartość funkcji celu w punkcie uznanym za optimum. Pytania:

1. Jak wartość parametru beta wpływa na szybkość dojścia do optimum i zachowanie algorytmu? Jakiej bety użyto dla każdej z funkcji?
2. Zalety/wady algorytmu?
3. Wnioski

2 Część teoretyczna

2.1 Dyskretny problem plecakowy

Mamy do dyspozycji plecak o maksymalnej pojemności W oraz zbiór N elementów $\{x_1, ..., x_i, .., x_N\}$, przy czym każdy element ma określoną wartość p_i oraz wage w_i.

Celem jest wybranie przedmiotów o największej łącznej wartości i łącznej wadze nie większej niż W:

$$max \sum_{i=1}^{n} x_i \cdot p_i$$

$$\sum_{i=1}^{n} x_i \cdot p_i \leq W$$

$$x_i \in 0, 1$$

Problem plecakowy jest problemem NP trudnym, jego rozwiązanie wielomianowe nie jest znane.

2.1.1 Rozwiązanie wyczerpujące

Pierwszy algorytm rozwiązujący problem plecakowy polega na przeglądzie wyczerpującym. Analizowane są w nim wszystkie kombinacje (tudzież wszystkie podzbiory zbioru N) i wybierane jest rozwiązanie optymalne.

2.1.2 Rozwiązanie przy użyciu heurystyki

Drugi algorytm rozwiązujący problem korzysta z nastepującej heurystyki: do plecaka pakowane są przedmioty według kolejności wynikającej ze stosunku ich wartości do masy.

2.2 Metoda najszybszego wzrostu

Niech $\{\beta_t, t = 1, 2, 3, ...\}$ będzie ciągiem liczb dodatnich, $\mathcal{X} = \mathcal{R}^d$, $J : X \to \mathbb{R}$, zaś $\{x_t, t = 1, 2, 3, ...\}$ - ciągiem elementów $\mathcal{X}$ wyznaczanych w następującej iteracji:

$$x_{t+1} = x_t - \beta_t \cdot \nabla J(x_t), \quad t = 1, 2, ...$$

dla pewnego $x_1 \in \mathcal{X}$. Procedura ta modyfikuje wektory x_t przeciwnie do kierunku wzrostu funkcji J.

3 Część praktyczna

3.1 Dyskretny problem plecakowy

3.1.1 Rozwiązanie wyczerpujące

Przegląd wyczerpujący wymaga przeszukania całego zbioru rozwiązań i znalezienia w ten sposób rozwiązania optymalnego. Zbiór rozwiązań jest zbiorem wszystkich podbiorów zbioru elementów, z których wybieramy te, które mają się znaleźć w plecaku. W związku z tym, takich podzbiorów jest 2^n, gdzie n to liczba elementów, z których wybieramy.

3.1.2 Wartości funkcji oceny

Celem pierwszego eksperymentu było zbadanie parametrów rozwiązania uzyskanych za pomocą heurystyki i porównanie ich z metodą przeglądu wyczerpującego. Za pomocą liczb pseudolosowych wybrano wartości oraz masy przedmiotów dla zadanej ich liczby. Maksymalna masa plecaka, w każdym powtórzeniu programu wynosiła połowę sumy mas wszystkich wylosowanych przedmiotów. Zakres mas przedmiotów oraz ich wartości zawierały się w zadanym przedziale i miały wartości całkowite. Z uwagi na fakt, że eksperyment uwzględniał użycie liczb pseudolosowych, Był on powtarzany wielokrotnie.

W tabeli przedstawione zostały rezultaty badań dla różnych przedziałów wartości i mas przedmiotów dla stałej liczby przedmiotów.

Parametry rozwiązania za pomocą heurystyki w porównaniu do przeglądu wyczerpującego				
liczba iteracji	25	25	25	25
liczba elementów	12	12	25	12
Zakres wartości przedmiotów i mas	(1, 10)	(1, 100)	(90,100)	(900, 1000)
współczynnik rozwiązań optymalnych	0.73	0.71	0.12	0.17
maksymalny błąd bezwzględny	4	32	10	119
maksymalny błąd względny	0.09	0.07	0.02	0.02
średnia wartość błędu bezwględnego	0.46	4.19	3.78	35.97
średnia wartość błędu względnego	0.01	0.01	0.01	0.01
odchylenie standardowe błędu bezwzględnego	0.93	7.82	2.57	31.21
odchylenie standardowe błędu bezwzględnego	0.02	0.02	0.01	0.01

Tablica 1: Rezultaty pomiarów

Z pomiarów wynika, że pomimo tego że, wraz ze wzrostem zakresu wartości i mas przedmiotów maleje współczynnik rozwiązań optymalnych osiąganych przez algorytm heurystyczny. Pomimo jednak, że wzrasta również średnia wartość błędu bezwględnego, w tym samym czasie wartość średniego błędu względnego i maksymalnego błędu średniego są stosunkowo małe - Stanowią zaledwie kilka procent wartości optymalnej.

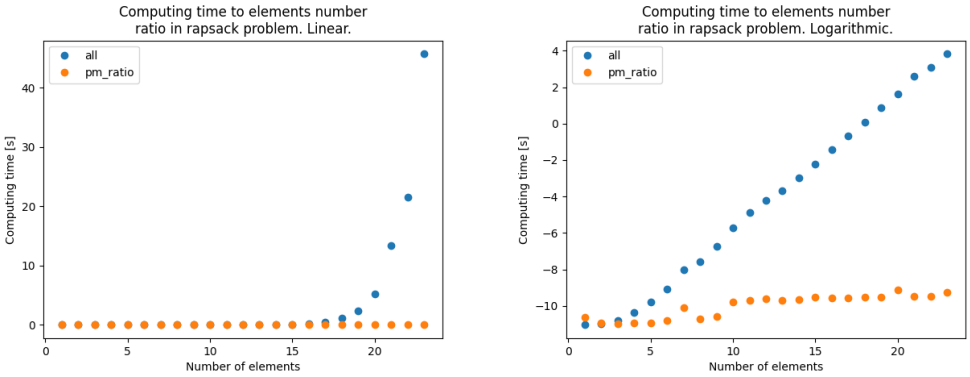
3.1.3 Czas działania programu

W miarę wzrostu liczby przedmiotów zwiększa się liczba możliwych rozwiązań, a więc i czas wyznaczenia rozwiązania. Celem drugiego eksperymentu było zbadanie, w jaki sposób rośnie ta zależność. Mierzony był czas działania programu. Z uwagi na to, że liczba operacji jest niemal taka sama niezależnie od parametrów przedmiotów, dla jednej ich liczby był wykonowany tylko jeden pomiar, w celu redukcji trwania ekperymentu. Proces był powtarzany tak długo, aż czas działania solwera optymalnego wyniósł więcej niż 1 minuta. Ten sam test, dla tych samych zmiennych był wykonywany dla obu solwerów.

Na wykresach przedstawiono zależność czasu działania programu od liczby przedmiotów dla obu solwerów w skalach liniowej oraz logarytmicznej.

Pomiar czasu wyższy niż 60 sekund zaobserwowano dla $n = 23$.

Warto zauważyć, że wyniki przyjmują postać liniową w skali logarytmiczej, co sugeruje, że w skali liniowej wraz ze wzrostem liczby elementów czas obliczeń rośnie wykładniczo. Dla 22 przedmiotów plecaku czas obliczeń wyniósł ok. 42 sekundy, natomiast po dodaniu jednego przedmiotu



(a) Skala liniowa

(b) Skala logarytmiczna

Rysunek 1: Wykresy zależności czasu obliczeń od liczby przedmiotów

Czas wyniósł ponad 80 sekund, czyli ok. dwukrotnie więcej niż dla 22 sekund. Zgadza się to z teoretyczną złożonością.

3.2 Metoda najszybszego wzrostu

3.2.1 Wpływ współczynnika β

Współczynnik β wpływa na to, o jakiej długości krok przesunie się obecnie badany punkt. Jego wartość ma zatem kluczowy wpływ na zachowanie algorytmu. Wartości współczynnika β użyte w trakcie powyższego eksperymentu zostały wyznaczone w taki sposób, aby dla punku startowego przy uwzględnieniu ograniczeń kostkowych, działanie algorytmu było zbieżne. Ciekawym się okazało, to, że po pewnej liczbie iteracji, można zmniejszyć wartość współczynnika nawet o kilka rzędów wielkości, a działanie algorytmu dalej będzie zbieżne.

3.2.2 Zachowanie algorytmu

W celu zobaczenia zachowywania się algorytmu zostały zrobione generatory wykresów przedstawiające kolejne kroki algorytmu na tle poziomic. Dla każdej z badanych funkcji zostały otworzone 3 różne wykresy dla różnych punktów startowych dla zadanej funkcji. Funkcja booth jest funkcją, która przyjmuje wektory o wymiarze równym 2. Funkcje CEC2017 natomiast przyjmują jako argumenty wektory o wymiarze równym 10. W związku z tym, dla każdej funkcji wykres został utworzony trzykrotnie dla różnych kombinacji wymiarów przedstawianych. Założono, że podczas generowania poziomic, wartości argumentów, których wymiar nie jest badanych, mają wartości równe 0.

- Funkcja booth
  - Funkcja booth była stosunkową dobrze uwarunkowaną funkcją i już po kilkudziesięciu iteracjach algorytmu uzyskiwało się przyzwoity wynik.

- Minimum funkcji jest w punkcie $x = (1, 3)$ i wynosi $f(x) = 0$.

- CEC2017 f1
  - Funkcja CEC2017 f1 była najlepiej uwarunkowaną funkcją z testowanych funkcji z grupy CEC2017. Stosunkowo szybko zbiegała do minimum. Wartość współczynnika β, dla którego funkcja zbiegała z niemal każdego punku wyniosła 10^{-8}.
  - Minimum funkcji jest w punkcie:

```
x = [
        -49.24189985,
        -70.42955972,
        -29.61018187,
        -58.32676328,
        22.08960188,
        59.93874989,
        36.76670148,
        18.55873627,
        76.68042093,
        -37.25371714
]
```

    i wynosi $f(x) = 200.7140013364138$.

- CEC2017 f2 oraz f3
  - Funkcje CEC2017 f2 i f3 oraz ich gradienty przyjmują bardzo duże wartości. W związku z tym Potrzebne są w ich przypadku bardzo małe wartości współczynnika β. Przy stałym współczynniku, funkcje coraz wolniej zbiegają do swojego minimum i dojście do celu jest bardzo czasochłonne i kosztowne. Jak widać na rysunkach, pomimo dużej liczby iteracji oprogramowania, droga widoczna na wykresach jest stosunkowo krótka w porównaniu do wcześniej badanych funkcji. Widoczna jest tutaj główna wada algorytmu - brak zmiennego kroku, który pozwoliłby funcji na podtrzymywanie tempa docierania do minimum.
  - Minimum funkcji f2 jest w punkcie:

```
x = [
        -29.31995891,
        -16.76540101,
        -75.46834234,
        -0.63716325,
        4.30364919,
        -37.55999386,
        42.94548049,
        74.7345723,
        -63.04848011,
        71.32898298,
]
```

i wynosi $f(x) = 200.0295524387396$.

- Minimum funkcji f3 jest w punkcie:

```
x = [
        -56.00495921,
        4.49247818,
        35.32391443,
        8.23436017,
        -47.24491346,
        7.29982139,
        6.42709164,
        -2.54886095,
        -61.00295813,
        -47.12330431
]
```

i wynosi $f(x) = 300.0533132041394$.

4 Wnioski

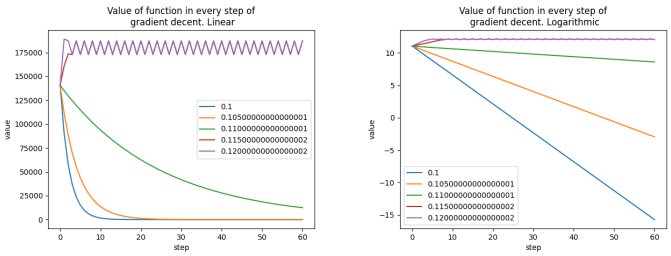
Udało się przetestować wszystkie solwery.

Zyski z rozwiązania nieoptymalnego potrafią być większe niż w przypadku rozwiązania optymalnego. W problemie plecakowym metoda heurystyczna pozwala na znaczące zmniejszenie czasu obliczeń kosztem uzyskania rozwiązania potencjalnie nieoptymalnego, różniącego się jednak od niego o stosunkowo małą wartość.

Zaletą algorytmu największego wzrostu jest fakt, że jesteśmy w stanie znaleźć często współczynnik beta taki, że w końcu uzyskamy rozwiązanie równe lub bliskie minimum funkcji. Wystarczy, żeby znaleźć wystarczająco małą tę wartość. Wada algorytmu jednak to fakt, że wartość gradientu funkcji zmienia się wraz ze zbliżaniem do minimum - gradient maleje, więc kroki są coraz krótsze i osiągnięcie minimum zajmuje bardzo dużo czasu. Bardzo dobrze było to widać w przypadku funkcji CEC2017, w których to przypadku na początku wartość współczynnika beta musiała być bardzo niska, aby algorytm zbiegał, ale po przerwaniu działania programu i nadpisaniu wartości startowej można było zwiększyć wartość współczynnika β a kilka rzędów wielkości, co bardzo zmniejszało sumaryczny czas obliczeń.

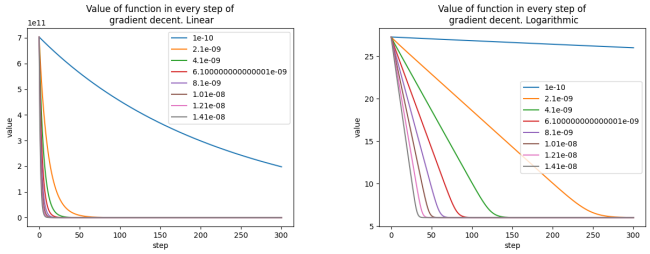
Potencjalnie możnaby porównać działanie algorytmu z jego modyfikacjami: algorytmem stochastycznego największego wzrostu, algorytmem ze zmiennym krokiem lub z pędem.

5 Wykresy



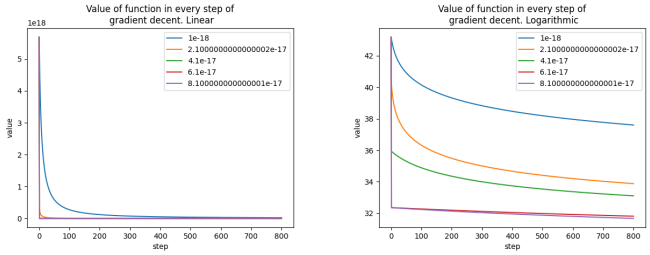
(a) Skala liniowa (b) Skala logarytmiczna

Rysunek 2: Charakterystyka wartości funkcji booth od kroku w zależności od wartości β.



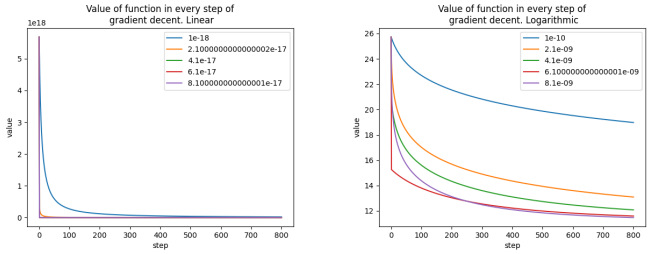
(a) Skala liniowa (b) Skala logarytmiczna

Rysunek 3: Charakterystyka wartości funkcji CEC2017 f1 od kroku w zależności od wartości β.



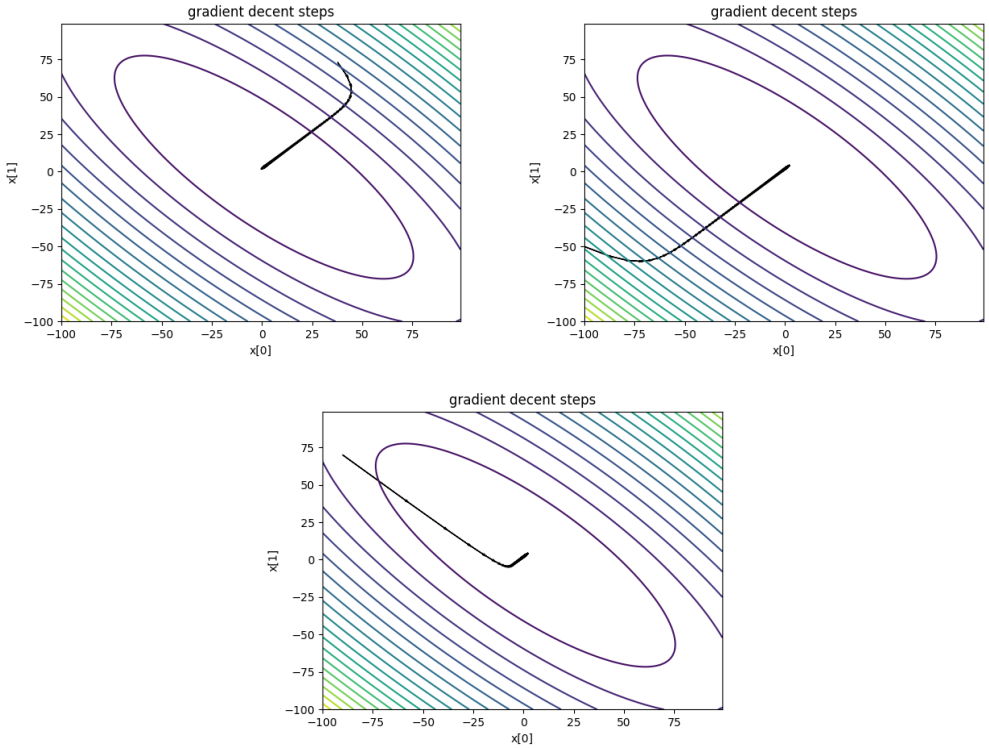
(a) Skala liniowa (b) Skala logarytmiczna

Rysunek 4: Charakterystyka wartości funkcji CEC2017 f2 od kroku w zależności od wartości β.

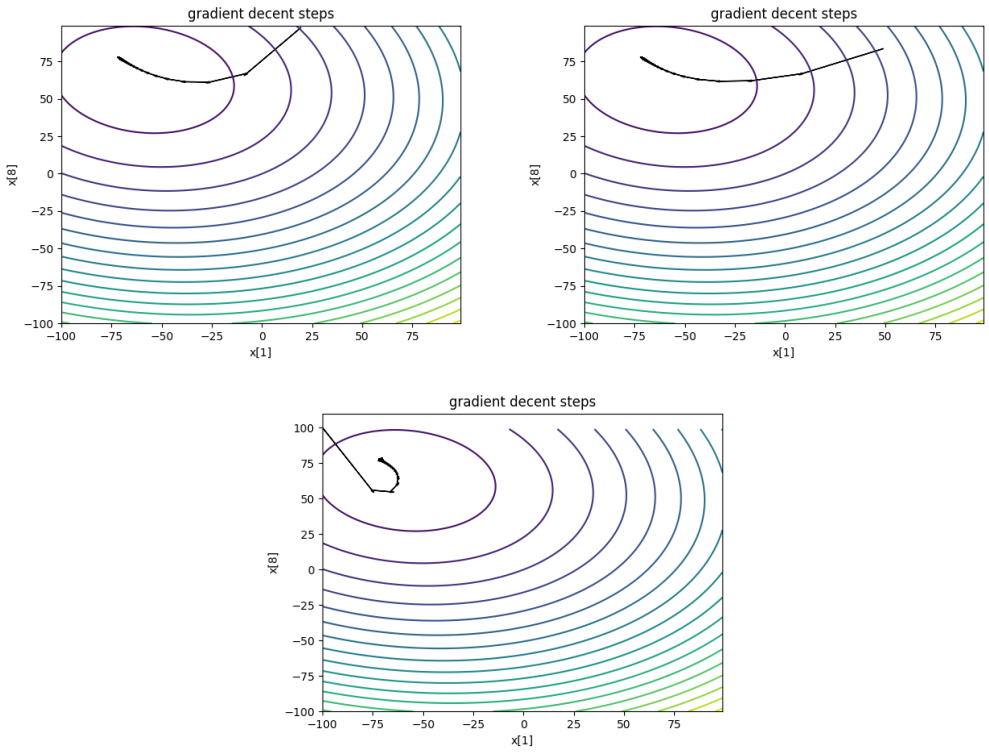


(a) Skala liniowa (b) Skala logarytmiczna

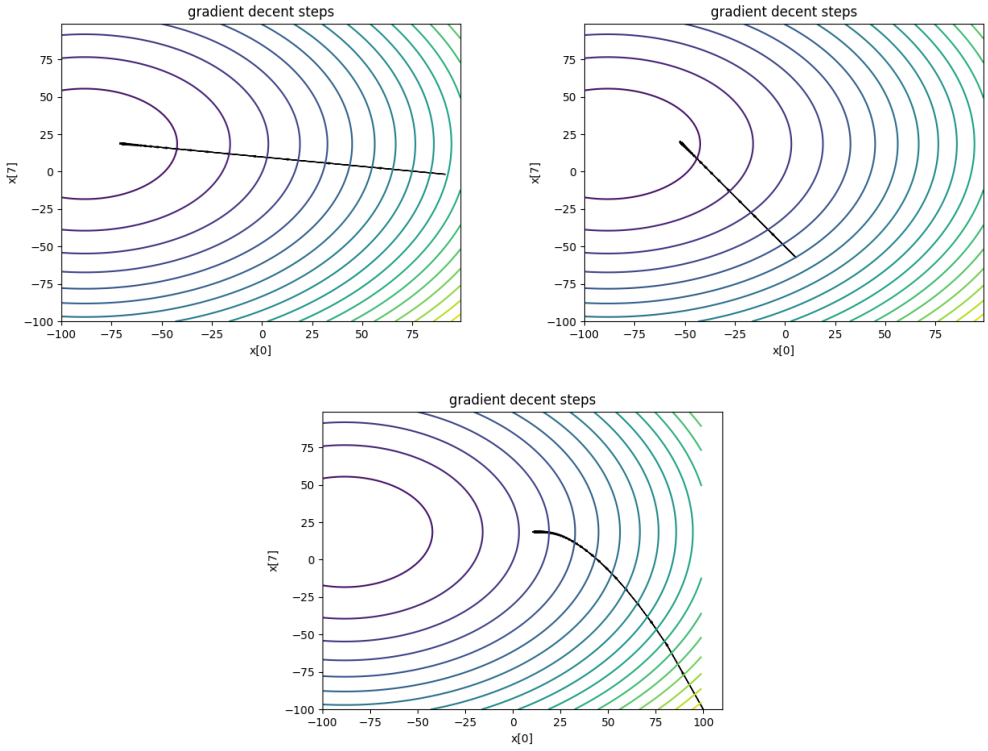
Rysunek 5: Charakterystyka wartości funkcji CEC2017 f3 od kroku w zależności od wartości β.



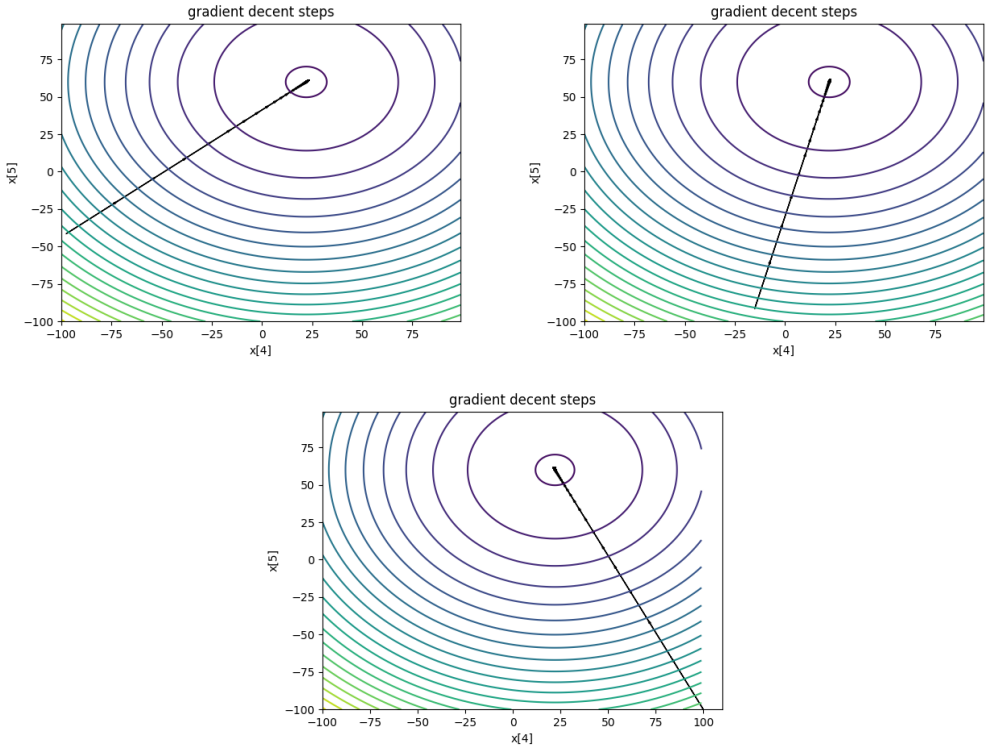
Rysunek 6: Zbieżność funkcji booth. $\beta = 0.11$.



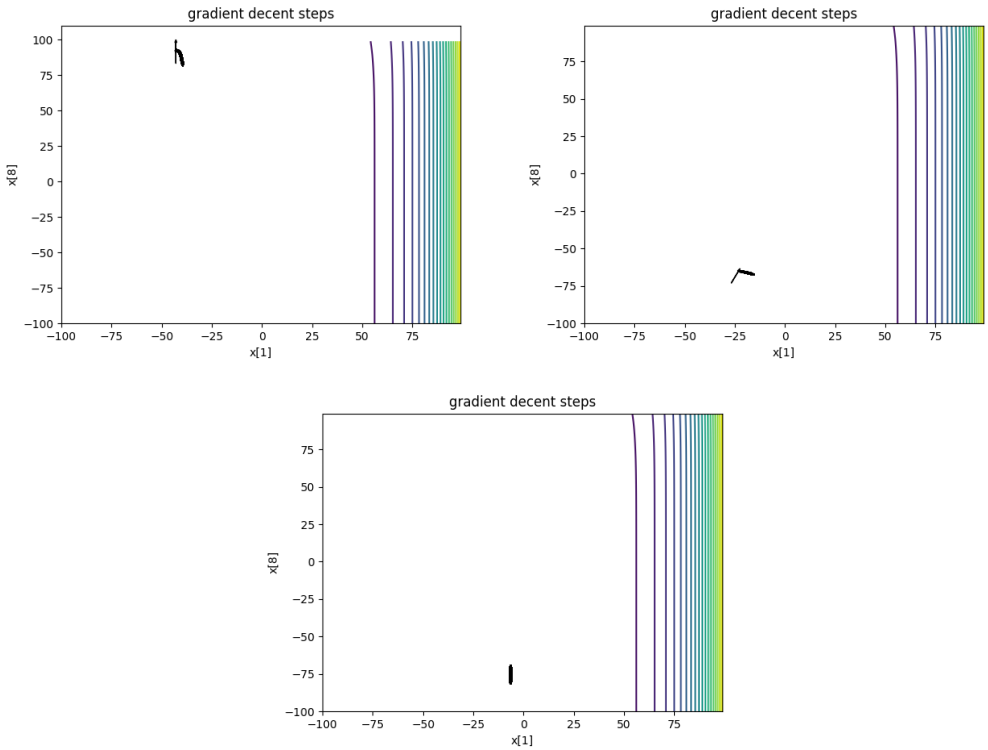
Rysunek 7: Zbieżność CEC2017 f1. Wymiary 1 i 8.



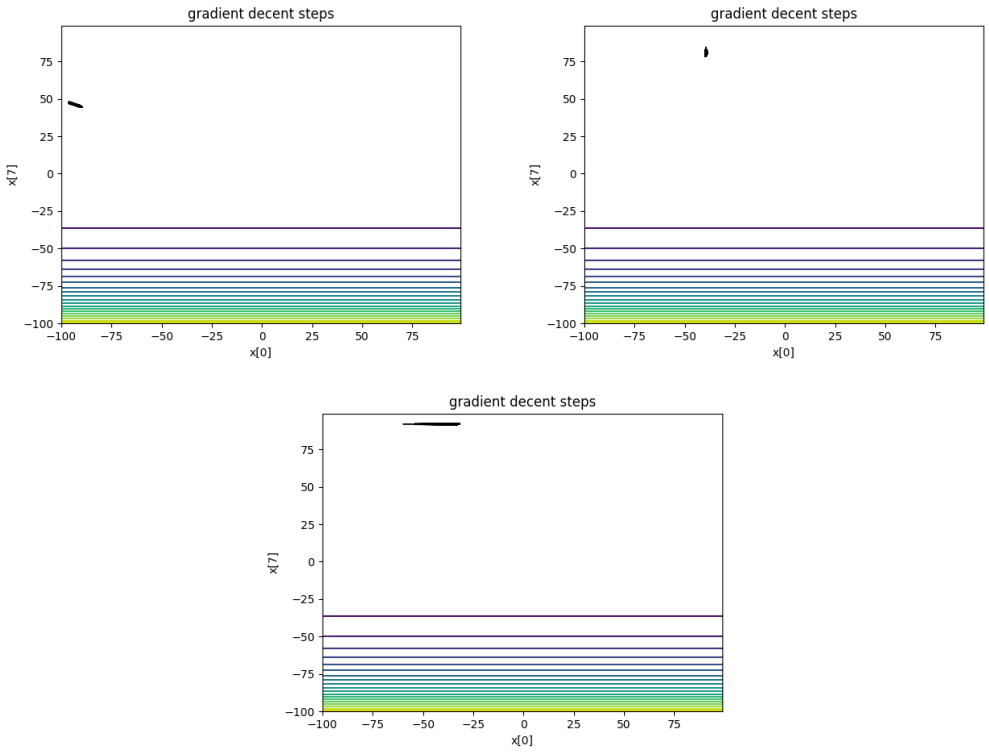
Rysunek 8: Zbieżność CEC2017 f1. Wymiary 0 i 7.



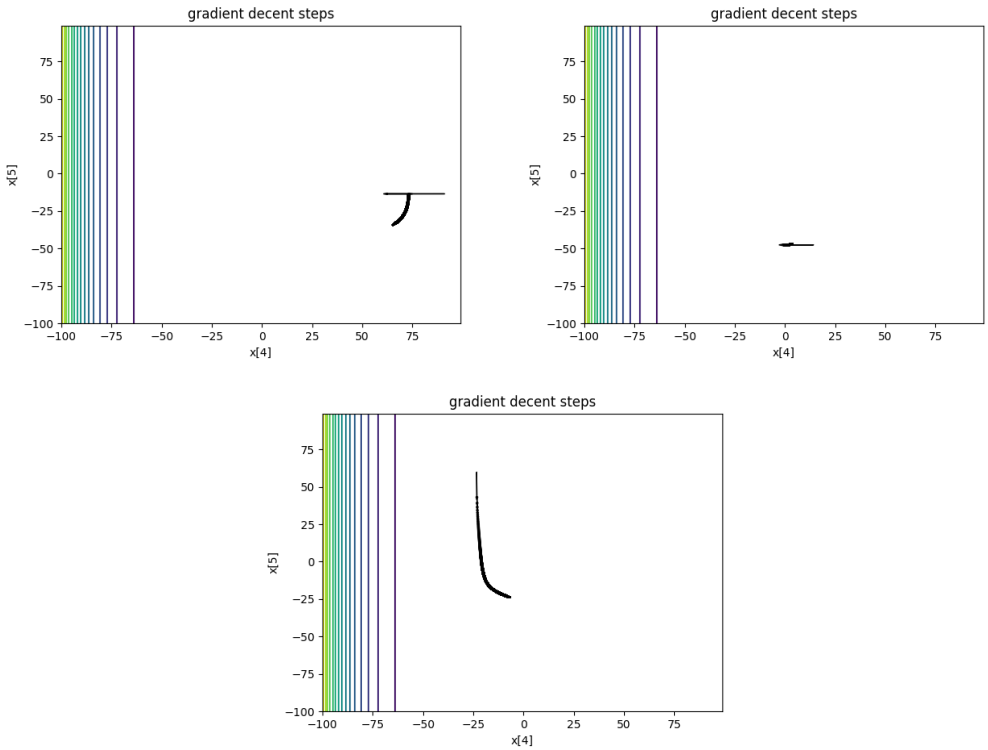
Rysunek 9: Zbieżność CEC2017 f1. Wymiary 4 i 5.



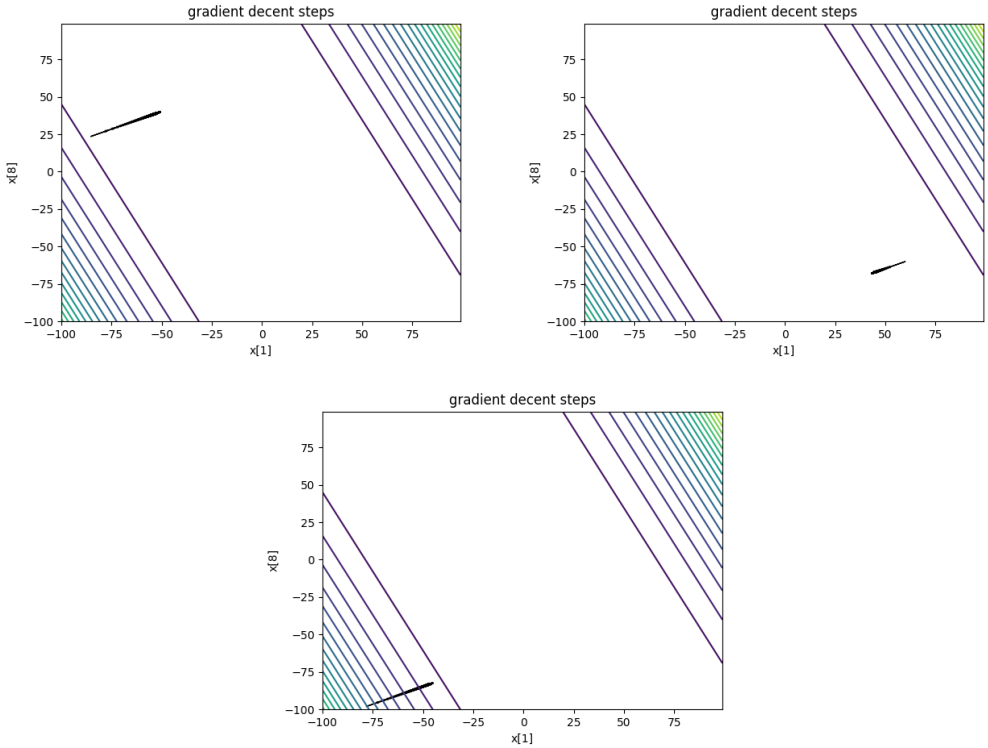
Rysunek 10: Zbieżność CEC2017 f2. Wymiary 1 i 8.



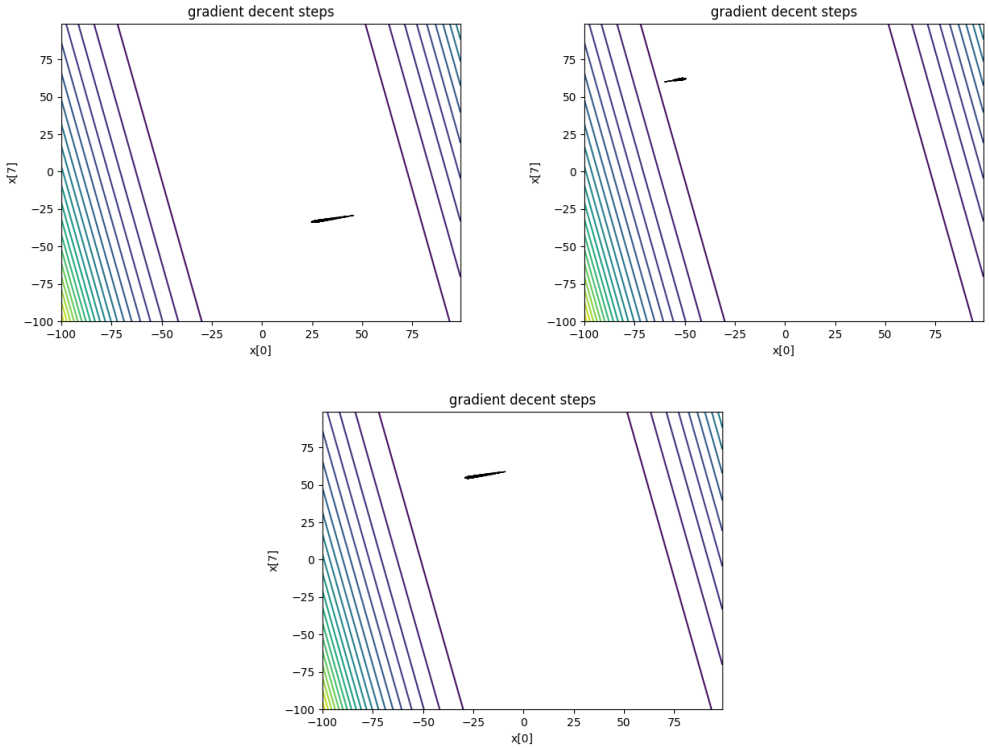
Rysunek 11: Zbieżność CEC2017 f2. Wymiary 0 i 7.



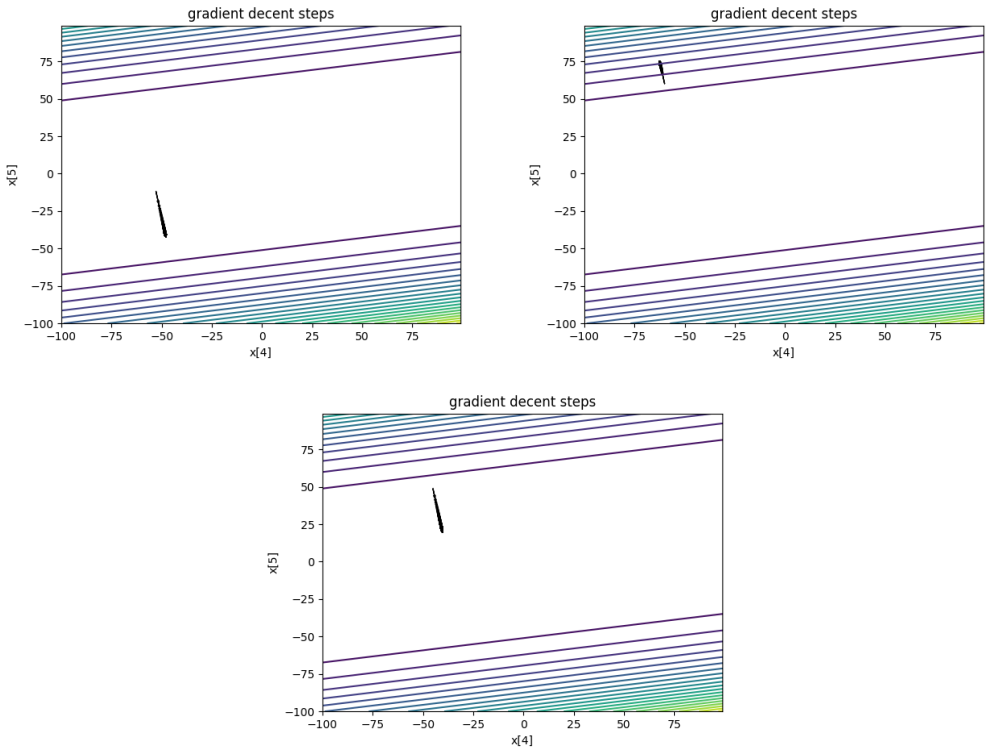
Rysunek 12: Zbieżność CEC2017 f2. Wymiary 4 i 5.



Rysunek 13: Zbieżność CEC2017 f3. Wymiary 1 i 8.



Rysunek 14: Zbieżność CEC2017 f3. Wymiary 0 i 7.



Rysunek 15: Zbieżność CEC2017 f3. Wymiary 4 i 5.