Kamil Marszałek 331401

Ćwiczenie 1

Zadanie 1

Jakie rozwiązania i jaką wartość funkcji oceny uzyskano? Czy uzyskano takie same rozwiązania?

Uzyskane rozwiązania różnią się od siebie. Dla zestawu danych podanego w instrukcji:

- przegląd wyczerpujący:

```
Mass limit: 9.0

Generating all combinations:
Max price: 17 Max mass: 8
Time: 0.000974s
Max price: 17 Mass of prods: 8
Time: 0.000974s
```

- rozwiązanie heurystyczne:

```
Mass limit: 9.0


Heuristic solution:
Max price: 14 Mass of prods: 5
Time: 0.000323s
```

Wyniki różnią się, ponieważ rozwiązanie heurystyczne dodaje do plecaka przedmioty w kolejności malejącego stosunku p/m, co nie zawsze daje najlepszy rezultat.

Liczby będące elementami tablic reprezentujących masy i wartości zostały wygenerowane losowo z zakresu od 1 do 100. Wyniki zostały przedstawione w tabeli.

Liczba elementów	Rozwiązanie przez przegląd - ocena	Rozwiązanie heurystyczne - ocena
2	43	43
3	174	174
4	183	153
5	109	103
6	226	220
7	292	272
8	181	181
9	325	317

10	327	327
11	446	419
12	540	540
13	604	604
14	593	581
15	800	800
16	673	650
17	756	719
18	505	505
19	641	637
20	657	657
21	871	859
22	1003	1003
23	739	732
24	871	856
25	1069	1061

Z przedstawionych danych można stwierdzić, że rozwiązanie heurystyczne ma szanse być rozwiązaniem optymalnym, ale nie jest to pewne. Uzyskujemy jakieś przybliżenie rozwiązania optymalnego.

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.

- Przegląd wyczerpujący:

Dane wejściowe:

```
NUM_OF_ELEMENTS = 25

m = np.array([randint(1, 100) for _ in range(NUM_OF_ELEMENTS)])  # mass of the objects
M = np.sum(m) / 2  # maximum mass of the objects
p = np.array([randint(1, 100) for _ in range(NUM_OF_ELEMENTS)])  # price of the objects
```

Dla takich 25 elementów otrzymujemy czas wykonania około 1 minuty.

```
Mass limit: 557.0

Generating all combinations:
Max price: 1100 Mass of prods: 557
Time: 65.093425s
```

-Rozwiązanie heurystyczne:

Dane wejściowe:

```python
NUM_OF_ELEMENTS = 12000000


m = np.array([randint(1, 100) for _ in range(NUM_OF_ELEMENTS)])  # mass of the objects
M = np.sum(m) / 2  # maximum mass of the objects
p = np.array([randint(1, 100) for _ in range(NUM_OF_ELEMENTS)])  # price of the objects
```

Dla 12 milionów elementów uzyskujemy podobny czas.

```
Mass limit: 303075791.0

Heuristic solution:
Max price: 490744300 Mass of prods: 303075791
Time: 69.455713s
```

Jak bardzo wydłuży obliczenia dodanie jeszcze jednego przedmiotu?

- Dla rozwiązania przez przegląd wyczerpujący:

Dla 23 elementów uzyskaliśmy czas około 16 sekund.

Dla 24 elementów uzyskaliśmy czas około 33 sekund.

Dla 25 elementów uzyskaliśmy czas około 65 sekund.

Widzimy że dodanie jednego elementu powoduje wydłużenie czasu około 2-krotnie.

-Dla rozwiązania heurystycznego:

Dla 100 elementów uzyskaliśmy czas około 0.005 sekundy.

Dla 101 elementów uzyskaliśmy czas około 0.005 sekundy.

Dla 102 elementów uzyskaliśmy czas około 0.005 sekundy.

Widzimy, że dodanie jednego elementu nie powoduje szczególnego zwiększenia czasu wykonania.

Jakie wnioski można wyciągnąć na podstawie wyników tego ćwiczenia?

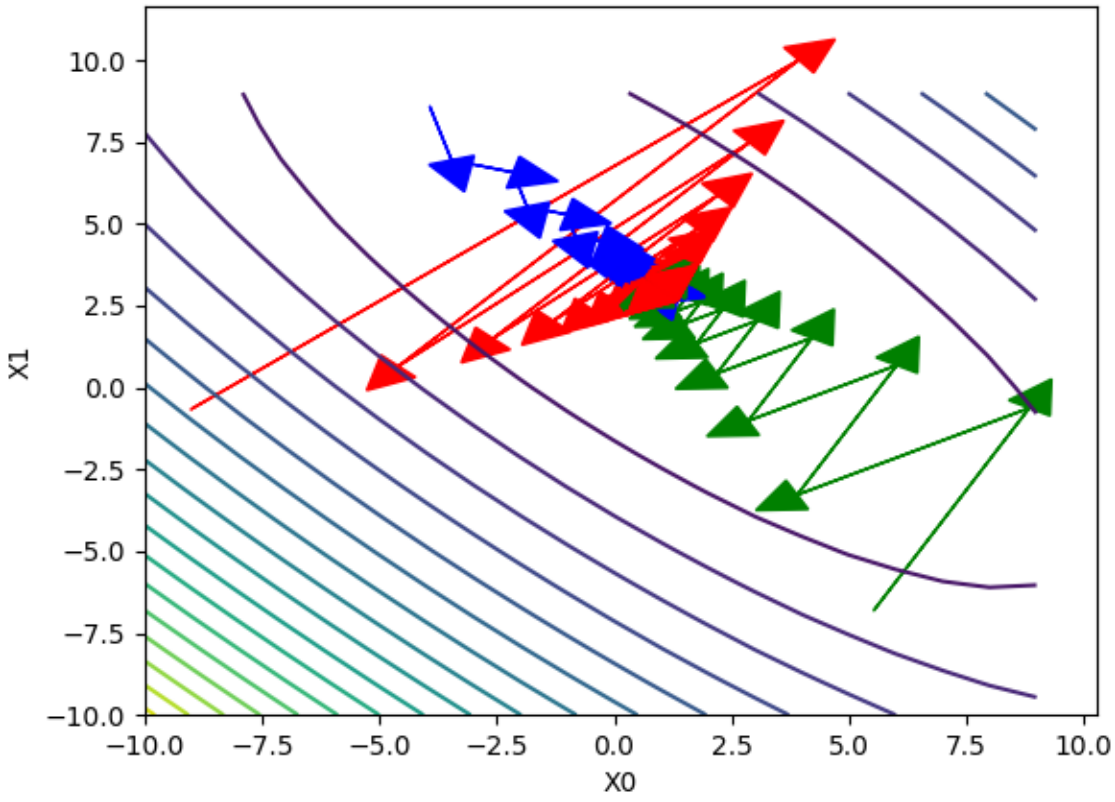
Można zauważyć, że rozwiązanie heurystyczne może nie znaleźć najlepszego wyniku, ale pozwala rozwiązywać w sposób przybliżony problem plecakowy dla dużo większej ilości elementów. Nie jest możliwe w rozsądnym czasie znalezienie dokładnego rozwiązania metodą przeglądu wyczerpującego dla dużej ilości elementów (w przypadku mojej konfiguracji w ciągu ok. jednej minuty da się rozwiązać problem dla maksymalnie 25 elementów).

Zadanie 2

Wersja klasyczna – stały parametr beta

Booth

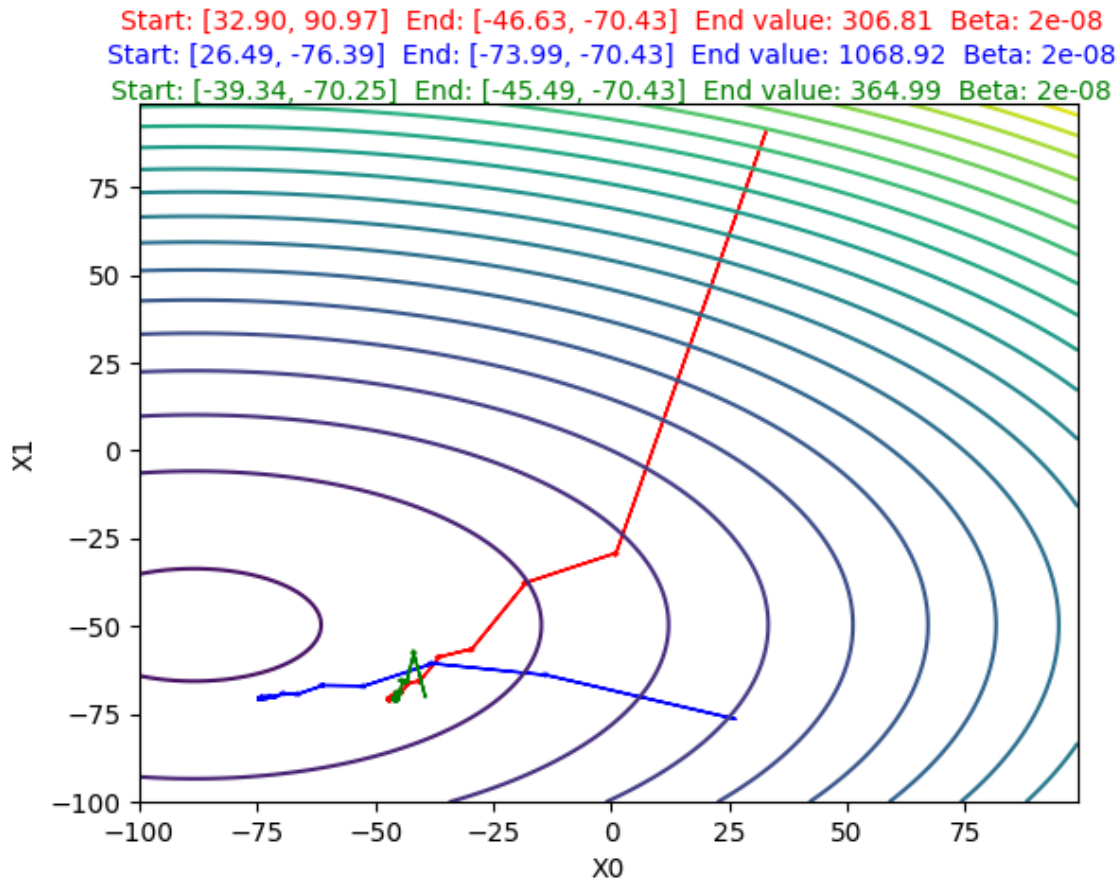
Dla prostej funkcji booth wersja ze stałym parametrem beta w zupełności wystarczy. Optimum jest szybko znalezione, potrzebna jest niewielka liczba iteracji. Maksymalna wartość bety dla której jest znajdowane minimum to 0.11.



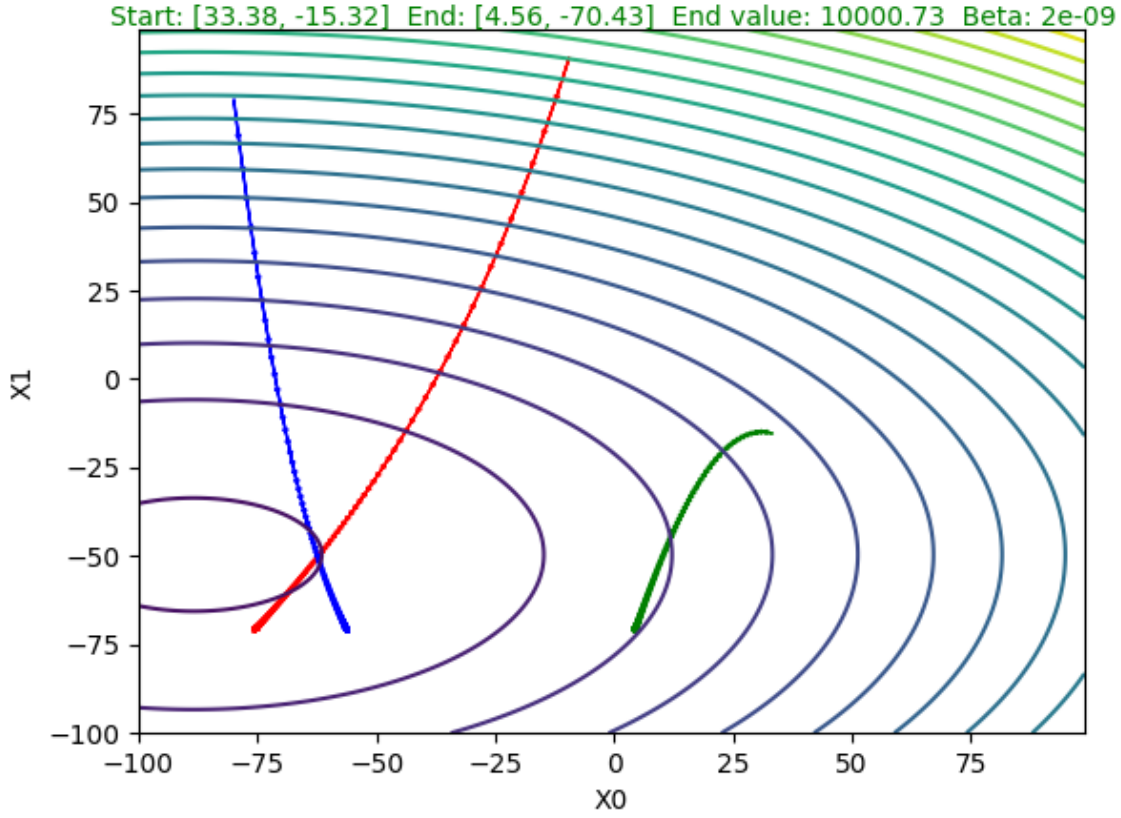
Funkcja f1

Parametr beta musi być mniejszy bądź równy 2e-08 inaczej dostajemy oscylacje i nie dojdziemy nigdzie. Czym większa beta tym algorytm działa szybciej, ale jeśli przekroczymy wartość graniczną nie jesteśmy w stanie uzyskać rezultatu. Algorytm zatrzymuje się w różnych miejscach, niekoniecznie równych sobie, z czego wynika że nie jest on w stanie znaleźć optimum w sposób efektywny. Wynik zależny jest od punktu startowego, który wybierany jest losowo.

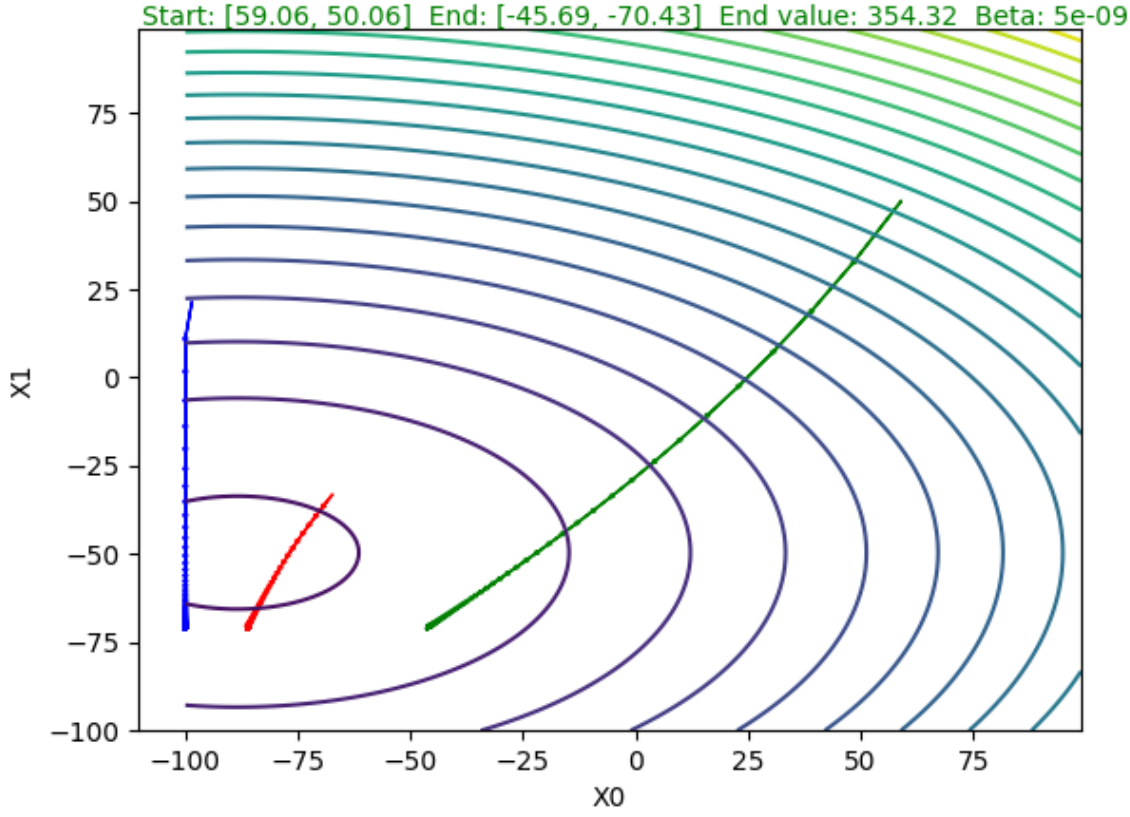
Wykresy dla f1:



Start: [-9.40, 90.80] End: [-75.27, -70.43] End value: 1205.95 Beta: 2e-09
Start: [-79.99, 79.01] End: [-56.34, -70.43] End value: 103.13 Beta: 2e-09
Start: [33.38, -15.32] End: [4.56, -70.43] End value: 10000.73 Beta: 2e-09



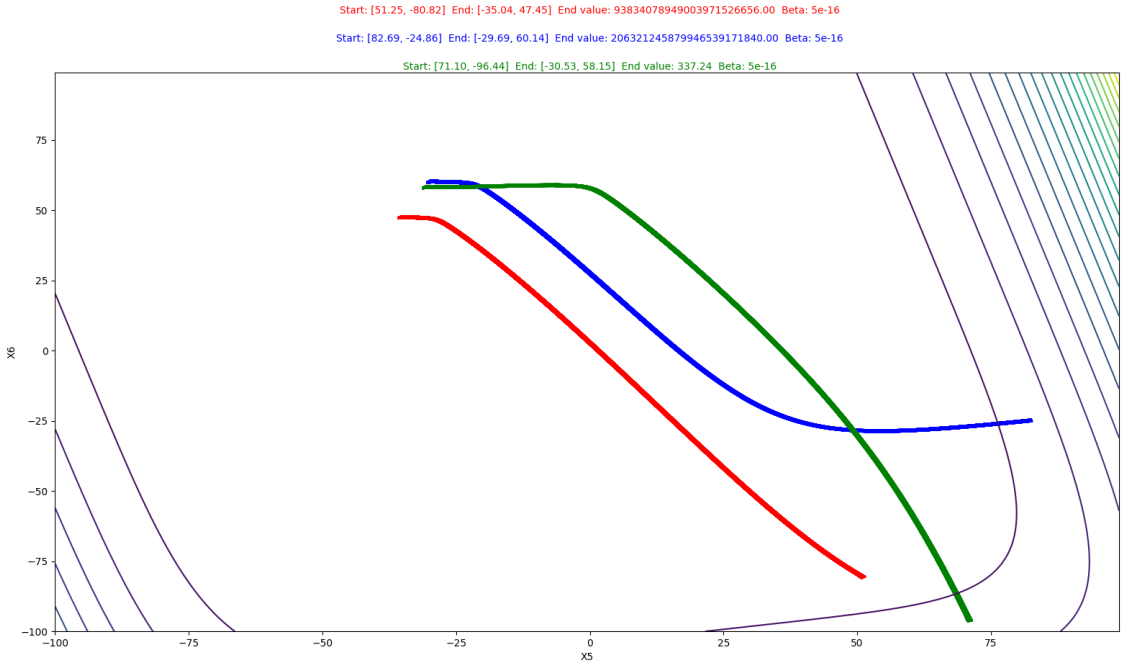
Start: [-67.30, -33.06] End: [-86.08, -70.43] End value: 2723.54 Beta: 5e-09
Start: [-98.54, 21.56] End: [-100.00, -70.43] End value: 5632.00 Beta: 5e-09
Start: [59.06, 50.06] End: [-45.69, -70.43] End value: 354.32 Beta: 5e-09



Zdecydowanie najmniej iteracji otrzymujemy dla największego parametru beta. Aczkolwiek najlepszy rezultat udało się uzyskać dla parametru beta wynoszącego 2e-09, jest to kwestia wylosowanego punktu początkowego aniżeli parametru beta. Optimum w tym przypadku wynosiło 103,13. Z kolei dla zbyt małej bety nie znajdujemy optimum przed osiągnięciem limitu iteracji (max. 10000).

Funkcja f2

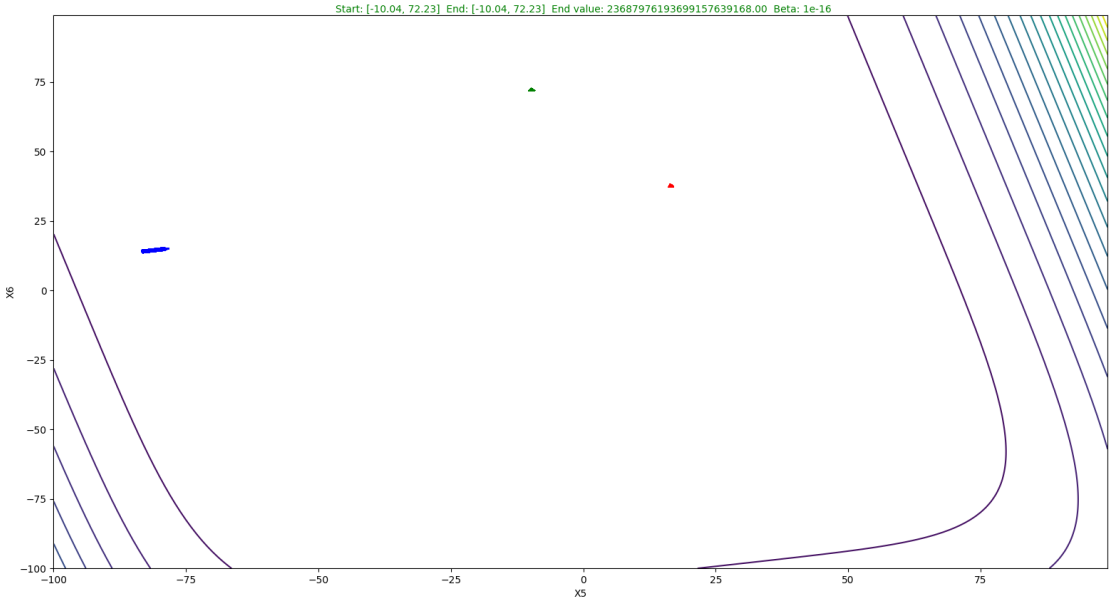
Dla funkcji f2 ciężko dobrać parametr beta, aby uzyskać sensowny rezultat w rozsądnym czasie. Funkcja f2 jest mocno niestabilna co skutkuje częstym zatrzymywaniem się w minimach lokalnych lub lokalnych oscylacjach. Z kolei dla zbyt małej bety nie dochodzimy do końca algorytmu tylko osiągamy limit iteracji. Rezultat mocno zależy od punktu startowego.



Start: [16.59, 37.27] End: [16.59, 37.27] End value: 11752205514267667149619 2.00 Beta: 1e-16

Start: [-83.33, 13.96] End: [-79.22, 14.85] End value: 2368797624692602463846 4.00 Beta: 1e-16

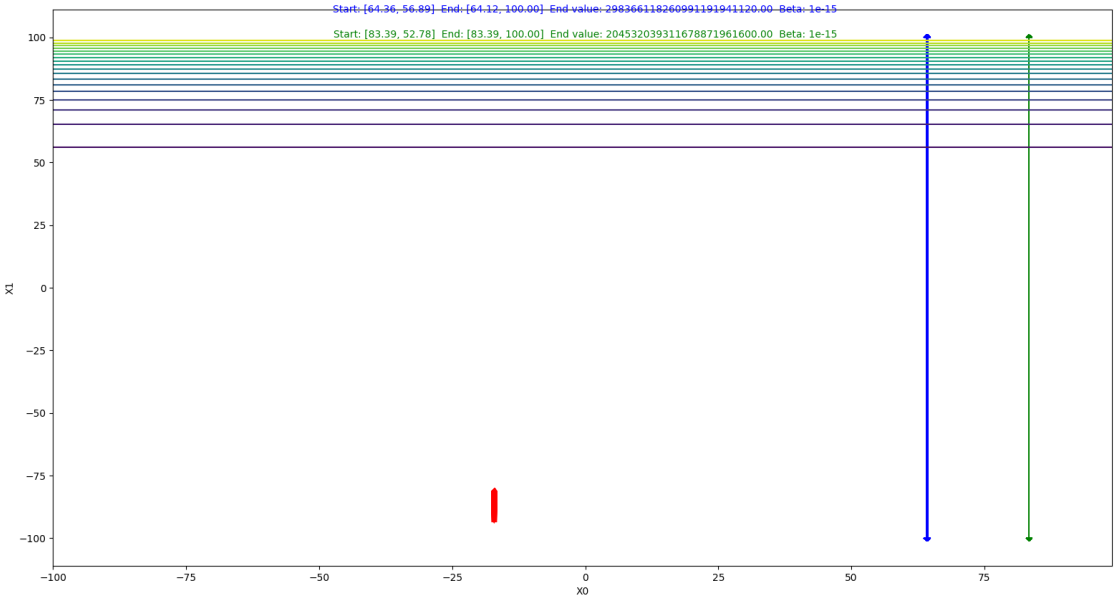
Start: [-10.04, 72.23] End: [-10.04, 72.23] End value: 2368797619369915763916 8.00 Beta: 1e-16



Start: [-17.12, -93.96] End: [-17.08, -81.37] End value: 9383407895171448176640 0.00 Beta: 1e-15

Start: [64.36, 56.89] End: [64.12, 100.00] End value: 2983661182609911919411 20.00 Beta: 1e-15

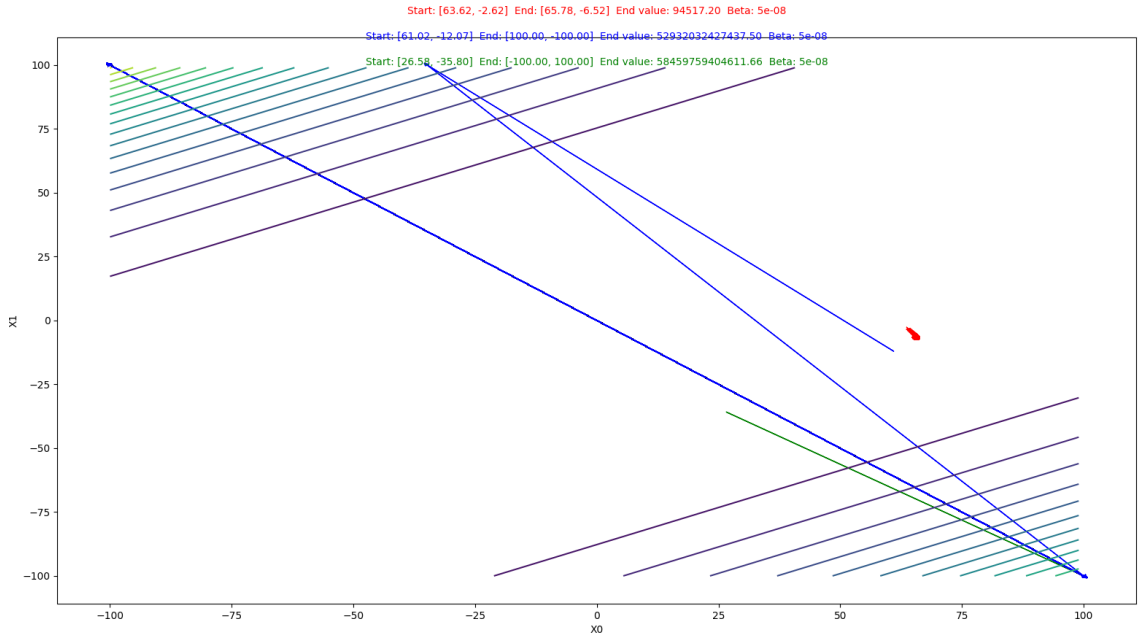
Start: [83.39, 52.78] End: [83.39, 100.00] End value: 2045320393116788719616 00.00 Beta: 1e-15



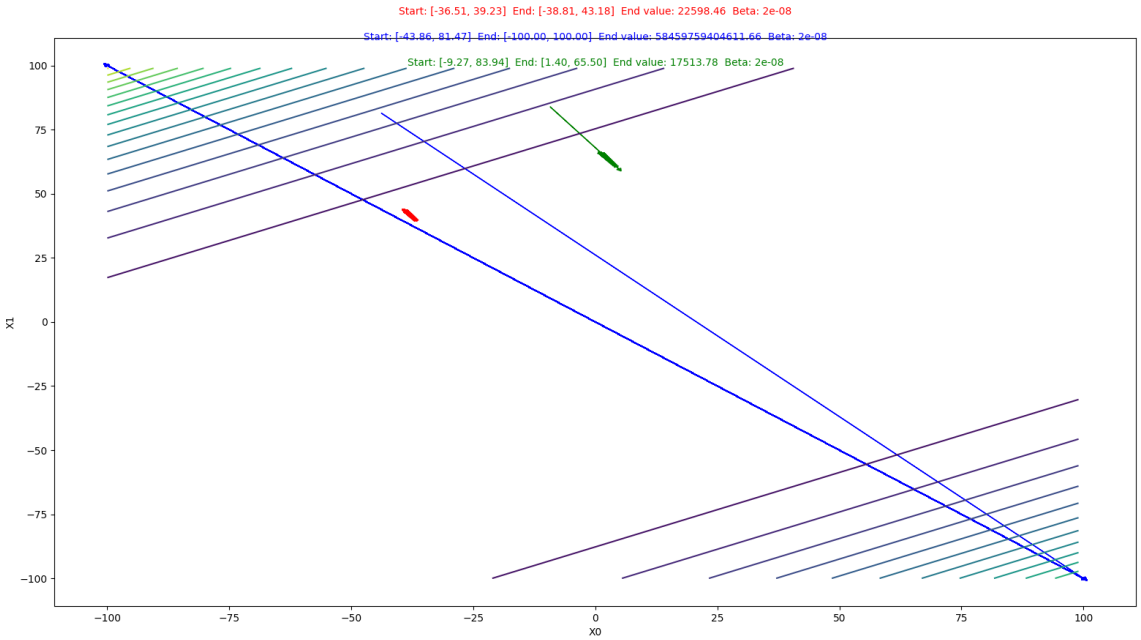
Funkcja f3

Dla funkcji f3 znalezienie minimum przy stałym współczynniku beta jest możliwe, ale bardzo czasochłonne, ponieważ f3 podczas zbliżania się do minimum mocno „zwalnia”.

Dla Beta = 5e-08 zachodzą oscylacje:



Dla beta = 2e-08 funkcja raz wpadła w oscylacje, a w pozostałych przypadkach znajduje się daleko od optimum:



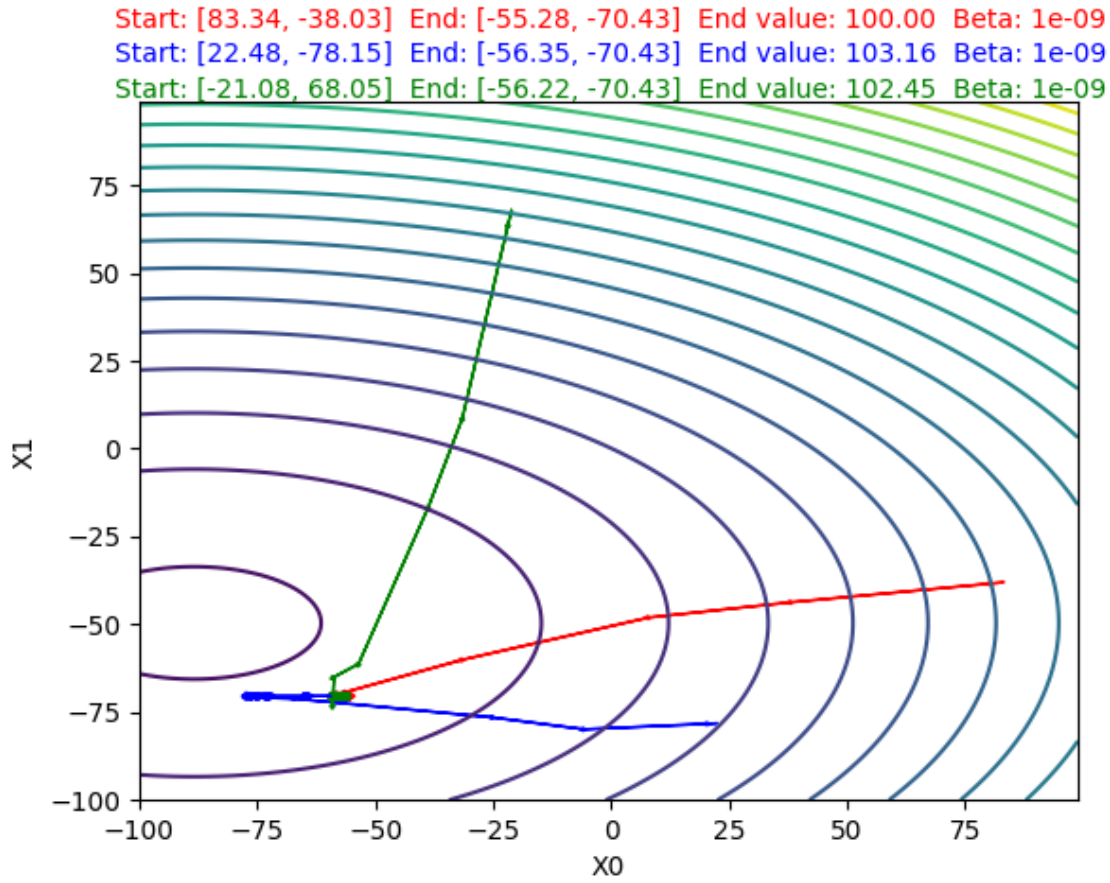
Metoda Barzilai-Borwein:

Tym razem beta będzie się zmieniała zgodnie z wzorem na tak zwany ‘short-step’:

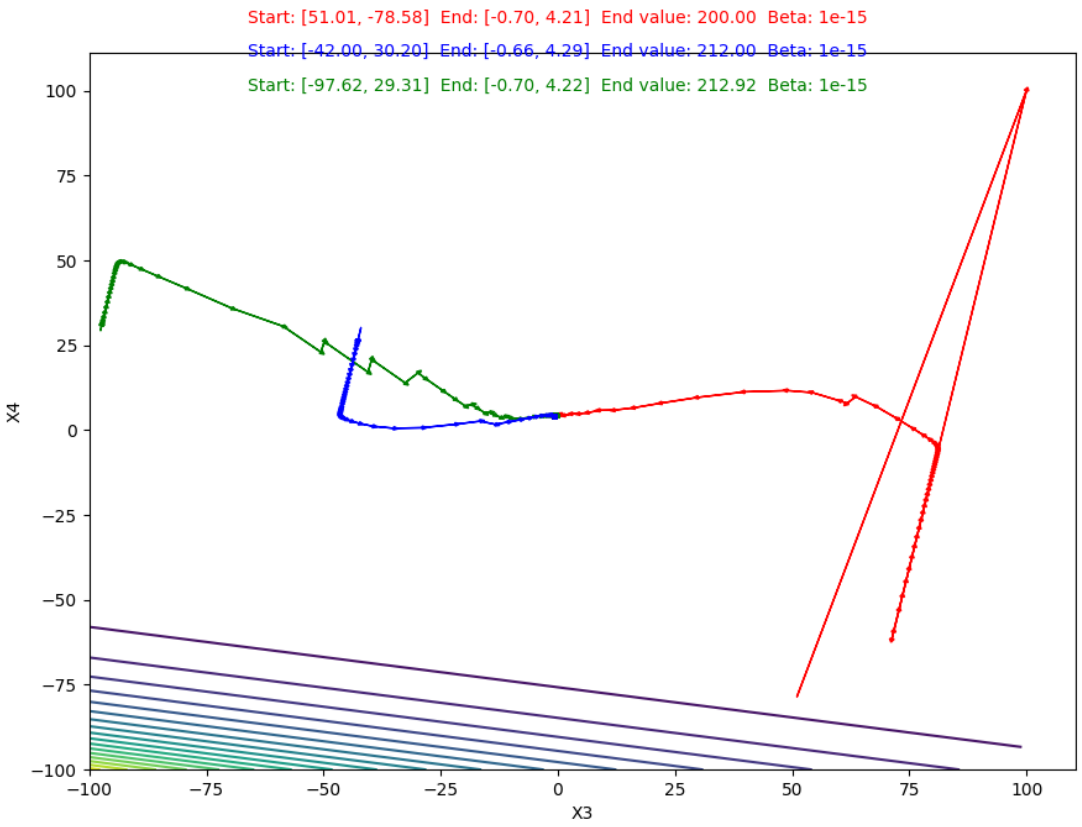
$$[\text{short BB step}]\ \alpha_k^{SHORT} = \frac{\Delta x \cdot \Delta g}{\Delta g \cdot \Delta g}.$$

Dzięki zastosowaniu tej metody jesteśmy w stanie w sposób dokładny wyznaczyć optima:

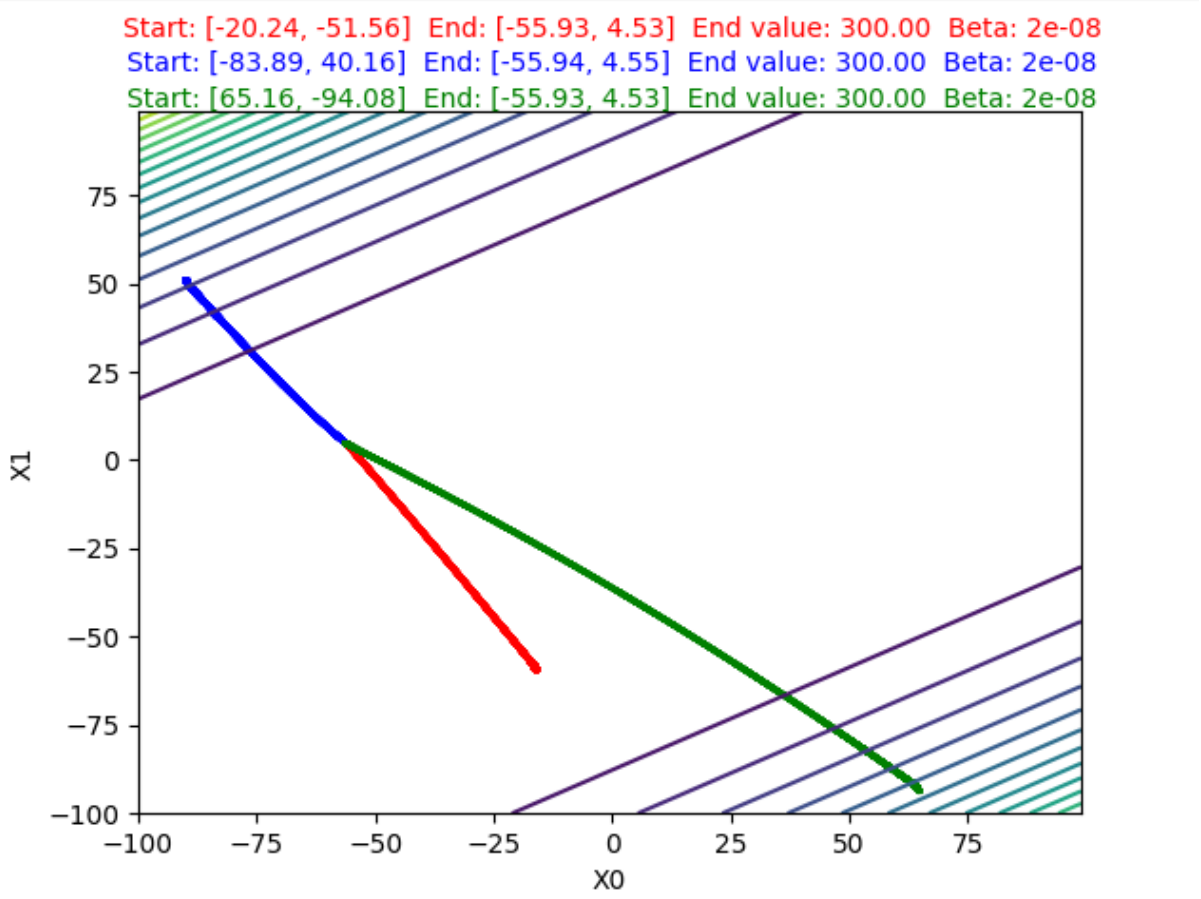
Funkcja f1:



Funkcja f2:


Start: [51.01, -78.58] End: [-0.70, 4.21] End value: 200.00 Beta: 1e-15
Start: [-42.00, 30.20] End: [-0.66, 4.29] End value: 212.00 Beta: 1e-15
Start: [-97.62, 29.31] End: [-0.70, 4.22] End value: 212.92 Beta: 1e-15
X4
X3
100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100

Funkcja f3:


Start: [-20.24, -51.56] End: [-55.93, 4.53] End value: 300.00 Beta: 2e-08
Start: [-83.89, 40.16] End: [-55.94, 4.55] End value: 300.00 Beta: 2e-08
Start: [65.16, -94.08] End: [-55.93, 4.53] End value: 300.00 Beta: 2e-08
X1
X0
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75