Tobiasz Kownacki WSI lab1

Zadanie 1

Mamy problem plecakowy jak na wykładzie:

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 (analizuje 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?

Rozwiązanie

- Brute Force – należy wziąć drugi i trzeci przedmiot
- Heurystyczne – należy wziąć drugi i czwarty przedmiot

```
BruteForce Backpack: [0, 1, 1, 0] has value 17
Heuristic Backpack: [0, 1, 0, 1] has value 14
```

Pytania

1. Uzyskano rozwiązanie, które widać powyżej. Według metody Brute Force należy wrzucić do plecaka drugi i trzeci przedmiot. Wartość funkcji oceny wynosi 17. Korzystając z heurystyki otrzymujemy plecak z drugim i czwartym przedmiotem. Wartość funkcji oceny wynosi 14.
2. W około minutę za pomocą Brute Force da się rozwiązać problem z 24 przedmiotami. Rozwiązanie takiego samego problemu za pomocą heurystyki zajmuje milisekundy

```
BruteForce method can handle 24 items in 56.1377 seconds
Heuristic method can handle 24 items in 0.0000 seconds
```

3. Dodanie kolejnego elementu spowoduje, że czas potrzebny na rozwiązanie problemu metodą Brute Force wydłuży się dwukrotnie.

```
After adding one more item, BruteForce method takes 122.6501 seconds
After adding one more item, Heuristic method takes 0.0000 seconds
```

4. Metoda Brute Force zawsze da nam poprawny wynik, ale wymaga bardzo dużo czasu. Wykorzystanie heurystyki skraca czas wykonywania programu do minimum. Płacimy za to otrzymaniem nie zawsze najlepszego wyniku, ale wciąż wystarczającego

Zadanie 2

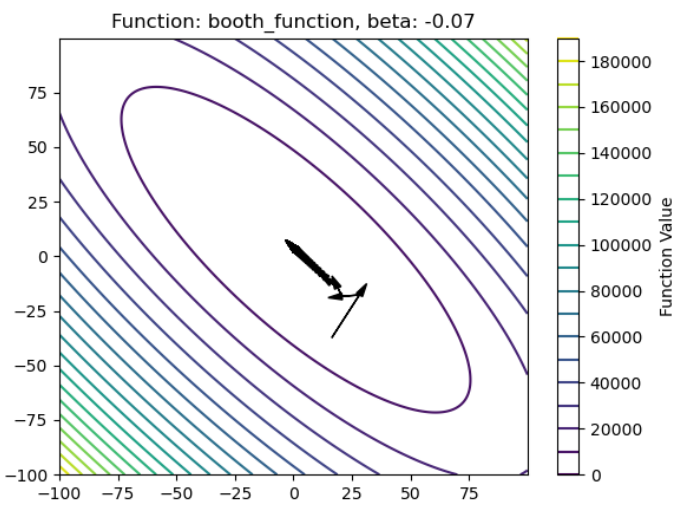
1. Zaimplementować metodę najszybszego wzrostu (minimalizacja, spodziewam się stałego współczynnika kroku, jeśli jednak ktoś chce zrobić więcej i zastosować zmienny współczynnik to ma taką możliwość). Gradient wyliczamy numerycznie.
2. Narysować zachowanie algorytmu (kolejne kroki algorytmu jako strzałki 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.
3. 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).

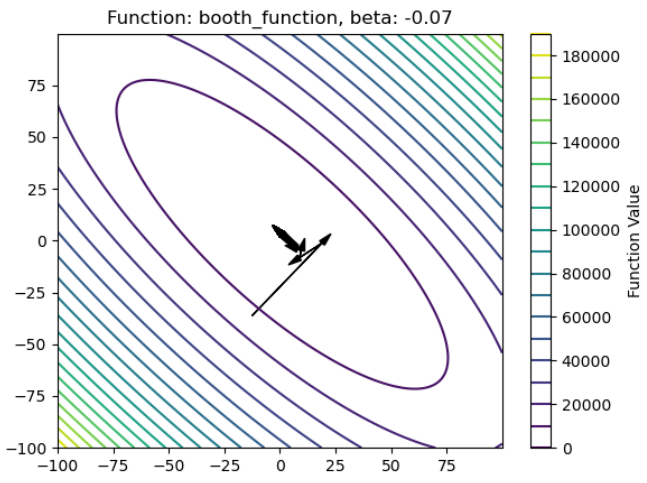
Pytania:

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

Wyniki

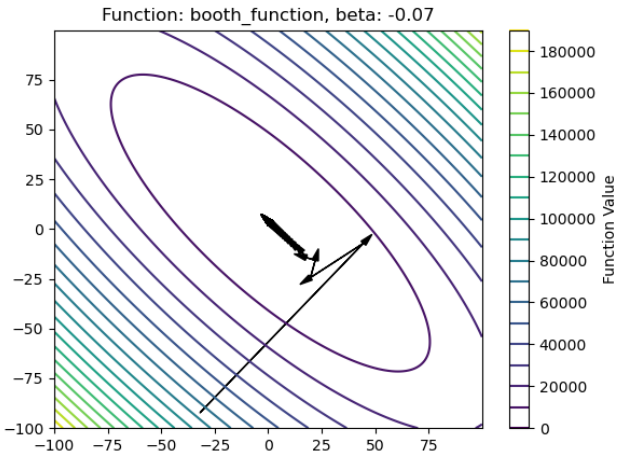
1. Funkcja Booth:





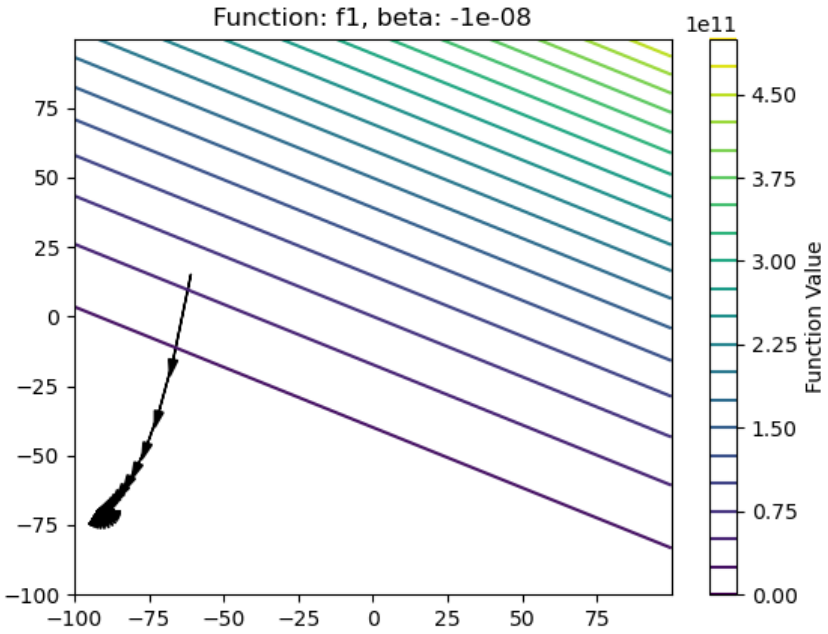

```
Function: booth_function, Iterations: 122, beta = -0.07
Initial point: [ 16.36970318 -37.25550956]
Final point: [1.00000033 2.99999967]
Function value: 0.000000
```

```
Function: booth_function, Iterations: 117, beta = -0.0
Initial point: [-12.55118058 -36.22831041]
Final point: [1.00000032 2.99999968]
Function value: 0.000000
```

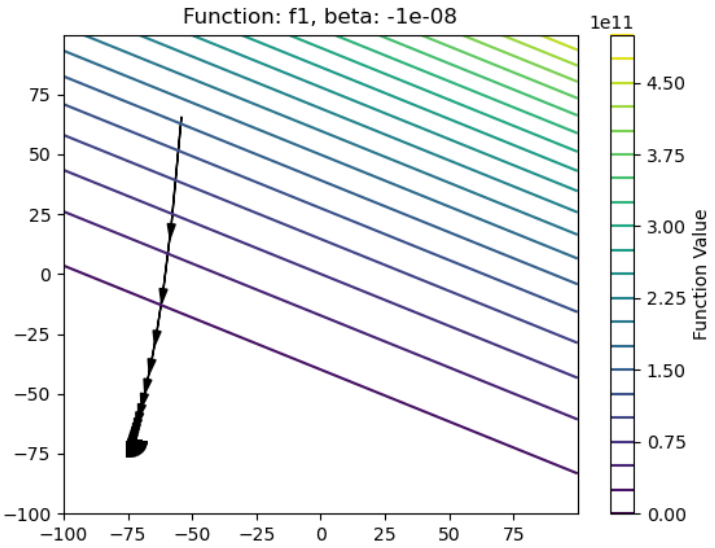



```
Function: booth_function, Iterations: 123, beta = -0.07
Initial point: [-31.89010333 -92.2526601 ]
Final point: [1.00000032 2.99999968]
Function value: 0.000000
```

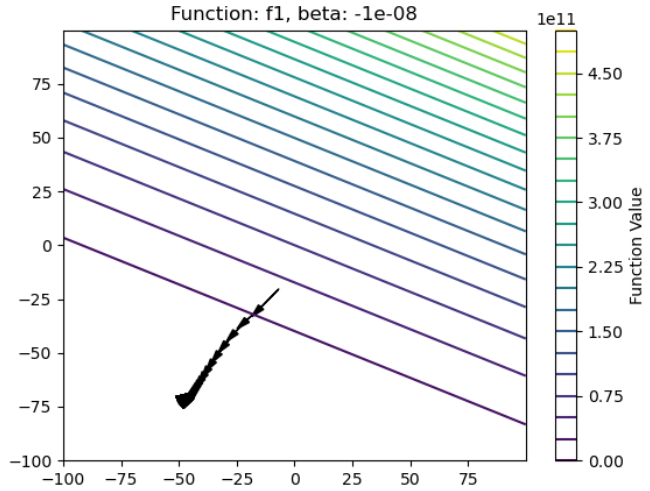
2. F1 Cec2017:




```
Function: f1, Iterations: 30002, beta = -1e-08
Initial point: [-60.96988504  15.18285311 -20.77532401 -85.69583009  23.9441975
  33.65729079   1.70428903  77.50112805  91.76057158  42.06252511]
Final point: [-90.66275558 -70.42955972 -29.61018187 -58.32676328  22.08960188
  59.93874989  -5.77218455  18.55873627  76.68042093  -2.32724638]
Function value: 3563.212763
```

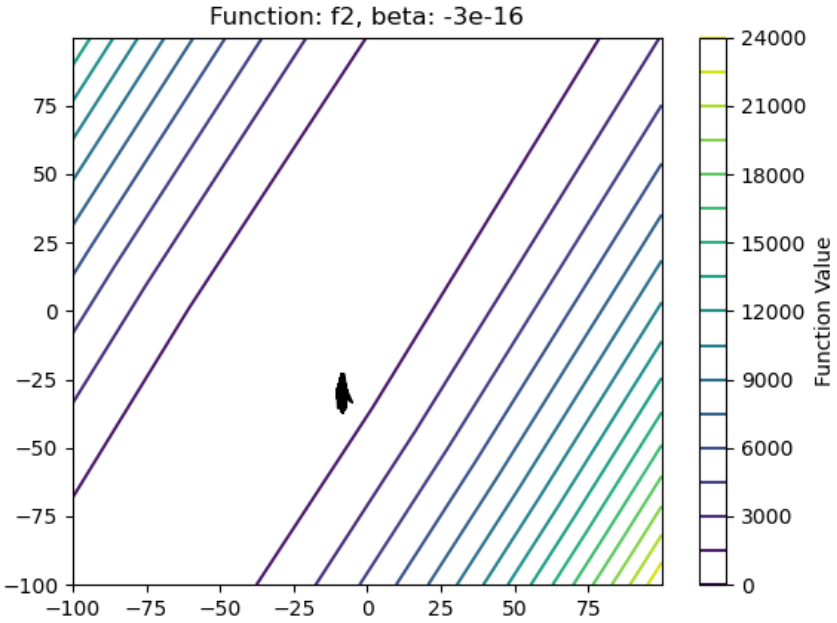



```
Function: f1, Iterations: 30002, beta = -1e-08
Initial point: [-54.17174702  65.59773176 -14.326706    96.04290561   0.6878485
 -47.72856138 -53.72885629 -78.40073405  26.35709733 -73.21539768]
Final point: [-73.65305805 -70.42955972 -29.61018187 -58.32676328  22.08960188
  59.93874989  11.69663814  18.55873627  76.68042093 -16.66998974]
Function value: 1033.986505
```

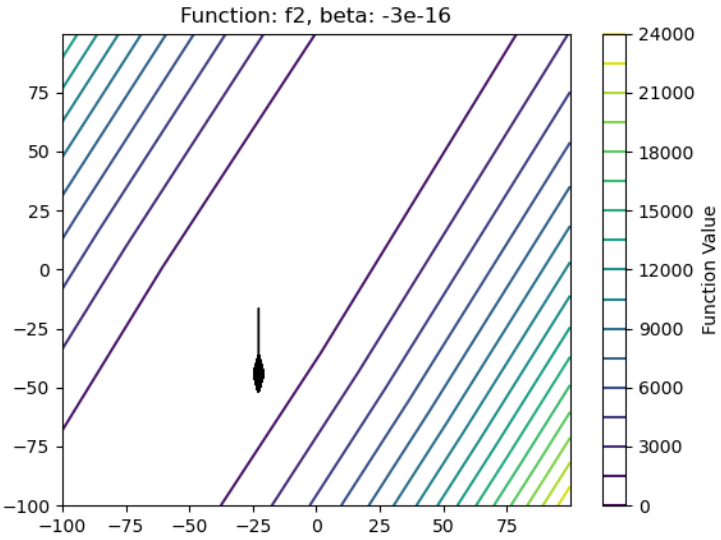



```
Function: f1, Iterations: 30002, beta = -1e-08
Initial point: [ -6.88024852 -20.40500677  -3.54479868  76.38713735  85.68008918
  55.70447555  77.9731237   71.85019773 -48.56380242  49.55832398]
Final point: [-45.09734572 -70.42955972 -29.61018187 -58.32676328  22.08960188
  59.93874989  41.02312527  18.55873627  76.68042093 -40.74844573]
Function value: 386.564372
```

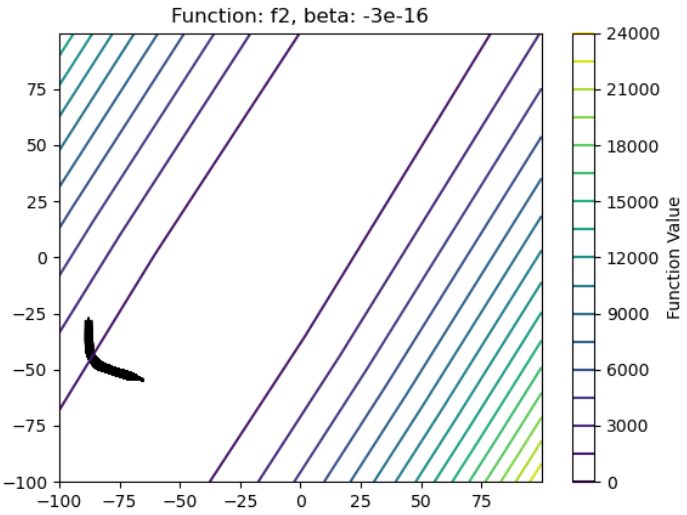
3. F2 Cec2017:




```
WARNING: f2 has been deprecated from the CEC 2017 benchmark suite
Function: f2, Iterations: 30002, beta = -3e-16
Initial point: [ -8.59895665 -36.21625537  24.41389597 -58.91084963  -9.13105637
   7.96367671  60.00400236  51.82401352 -76.72535733 -87.52311826]
Final point: [ -8.57738852 -31.3955042   24.41797633 -57.99765586  -3.99607175
   7.90505435  59.9912888   34.55428488 -79.38332609  27.95697939]
Function value: 3489255980133.824707
```

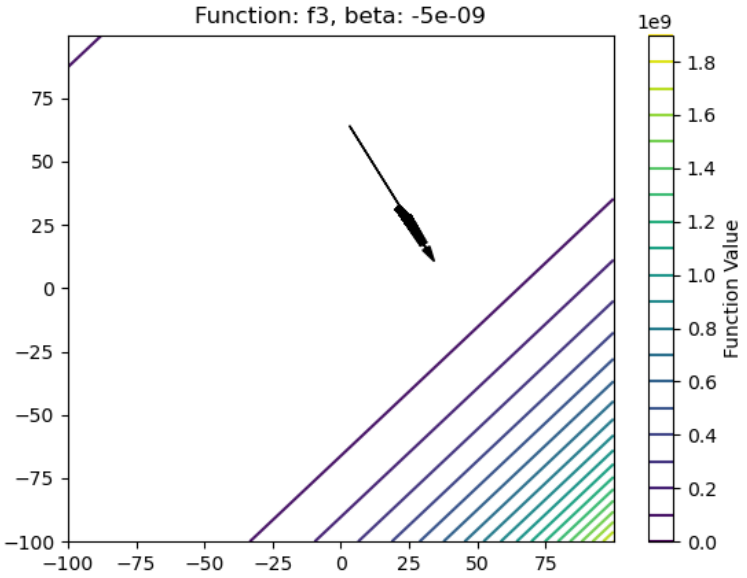



```
Function: f2, Iterations: 30002, beta = -3e-16
Initial point: [-22.7195333  -16.30440606 -64.7740222   14.77583361  14.74228882
 -51.36898744 -24.53543217   1.76809507  78.55087417 -58.47725264]
Final point: [-22.73036321 -45.87011082 -64.78219044  24.35096325  68.57255296
 -51.34575212 -24.472613     6.31575391  50.68486375  24.34382345]
Function value: 3547779104097.719238
```

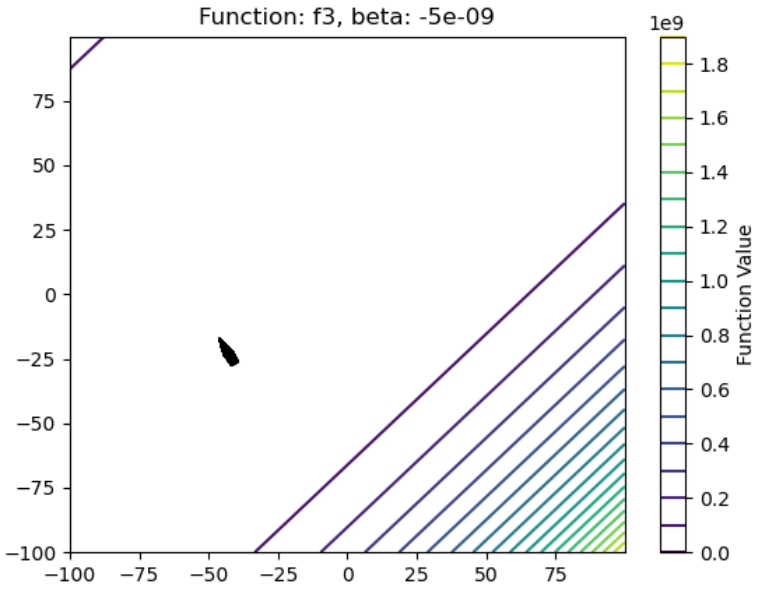



```
Function: f2, Iterations: 30002, beta = -3e-16
Initial point: [-87.80942405 -26.69477057  96.16909457   8.14725936 -60.70316093
  89.90978548  58.14593802 -41.56681837 -18.76421993  57.82595671]
Final point: [ -70.72731894  -52.88595481   99.40060036  100.          100.
   43.48037113   48.07822471  -28.97157774 -100.           64.34116063]
Function value: 1789206586471249084416.000000
```

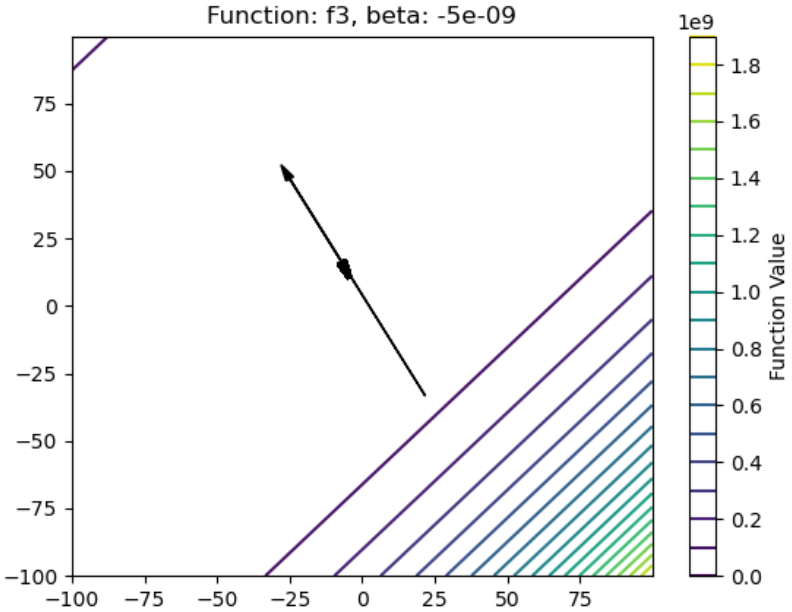
4. F3 Cec2017




```
Function: f3, Iterations: 30002, beta = -5e-09
Initial point: [  3.18504434  63.99163576 -28.78872894  -2.14219028  -0.28033372
  33.28343115 -63.07734291 -68.74359945  16.64016149 -16.86545605]
Final point: [ 24.47054929  27.29129293 -20.45440404   2.36094627  -5.51870447
  65.27843546 -21.70503973 -64.21653366  -0.72867026  -5.47447133]
Function value: 38551.198821
```




```
Function: f3, Iterations: 30002, beta = -5e-09
Initial point: [-40.44632319 -27.09401672  22.6613362   11.72929423  15.28163898
 -77.7034293   95.10161335 -64.50704534   2.20873974 -45.24820409]
Final point: [-43.14558017 -22.4446195   21.61336472  11.15906087  15.92362882
 -81.67847008  89.84404371 -65.05855196   4.38376885 -46.67886204]
Function value: 44334.954828
```




```
Function: f3, Iterations: 30002, beta = -5e-09
Initial point: [ 21.62030019 -33.19954207  88.49451627  90.45188826  -3.6557892
  86.19883443 -27.81059222 -54.30835125 -53.84800155  97.32048761]
Final point: [ -6.77847334  15.67218254  77.40300474  84.43331146   3.28964646
  43.48548718 -64.57573674 -60.29630674 -30.74625315  81.91579352]
Function value: 103038.650821
```

Pytania:

1. Im mniejsza wartość parametru Beta tym wolniej przesuwamy nasz aktualny punkt do optimum. Jeśli jednak wartość parametru Beta będzie za duża, może to doprowadzić do przeskoczenia optimum funkcji, a wręcz do uniemożliwienia znalezienia go.

Wartości bety:

- Funkcja Booth: -0,07
- F1: -0,00000001
- F2: -0.0000000000000003
- F3: -0.000000005

2. Zalety algorytmu:
   - Prosty w implementacji
   - Przełączanie się między szukaniem minimum a maximum jest bardzo proste
   - W miarę szybkie działanie dla funkcji w 2 wymiarach

   Wady:

   - Znajduje jedynie minimum lokalne
   - Przy większej ilości wymiarów potrzebna jest bardzo precyzyjna wartość parametru beta. W konsekwencji stoimy w punkcie, albo wychodzimy poza zakres
   - Długi czas wykonywania programu przy wielu wymiarach
3. Algorytm świetnie się nadaje do znalezienia optimum prostych dwuwymiarowych funkcji. Problemy rosną przy tych bardziej skomplikowanych. Dobranie dobrego parametru Beta jest bardzo trudne i wymaga wielu prób. Dynamiczne dobieranie parametru Beta powinno rozwiązać największą wadę tego algorytmu.