Sprawozdanie nr 2

Obliczenia ewolucyjne

Projekt 1 – implementacja chromosomu rzeczywistego

Zespół: Dariusz Kotula, Jeremiasz Macura, Konrad Makselan

1. Informacje o wykorzystywanych technologiach do wykonania projektu

Projekt nr 2 zdecydowaliśmy się wykonać w postaci aplikacji internetowych. Część serwerowa jest odpowiedzialna za całą logikę algorytmu genetycznego i została napisana w języku Python z użyciem frameworka Django. Graficzny interfejs użytkownika oparliśmy na frameworku React.js i języku JavaScript. GUI pozwala na ustawienie wszystkich parametrów, z którymi algorytm wykonuje swoje działania, przy pomocy formularza. Zarówno aplikacja backendowa jak i frontendowa zostały skonteneryzowane przy pomocy narzędzia Docker. Biblioteki, które wykorzystaliśmy w celu napisania tych aplikacji, znajdują się w pliku requirements.txt oraz client/package.json.

2. Wymagania środowiska do uruchomienia aplikacji

Aplikacje można uruchomić na dwa sposoby. Pierwszym zalecanym sposobem jest stworzenie i uruchomienie kontenera poprzez wykonanie komendy “docker-compose up” w głównym katalogu. W celu skorzystania z tej opcji musimy posiadać zainstalowane i uruchomione narzędzie Docker na naszej maszynie hostowej. Zaletą tego rozwiązania jest między innymi to, że nie musimy się przejmować o żadne pozostałe zależności wymagane w celu uruchomienia aplikacji i wszystko dzieje się w sposób automatyczny.

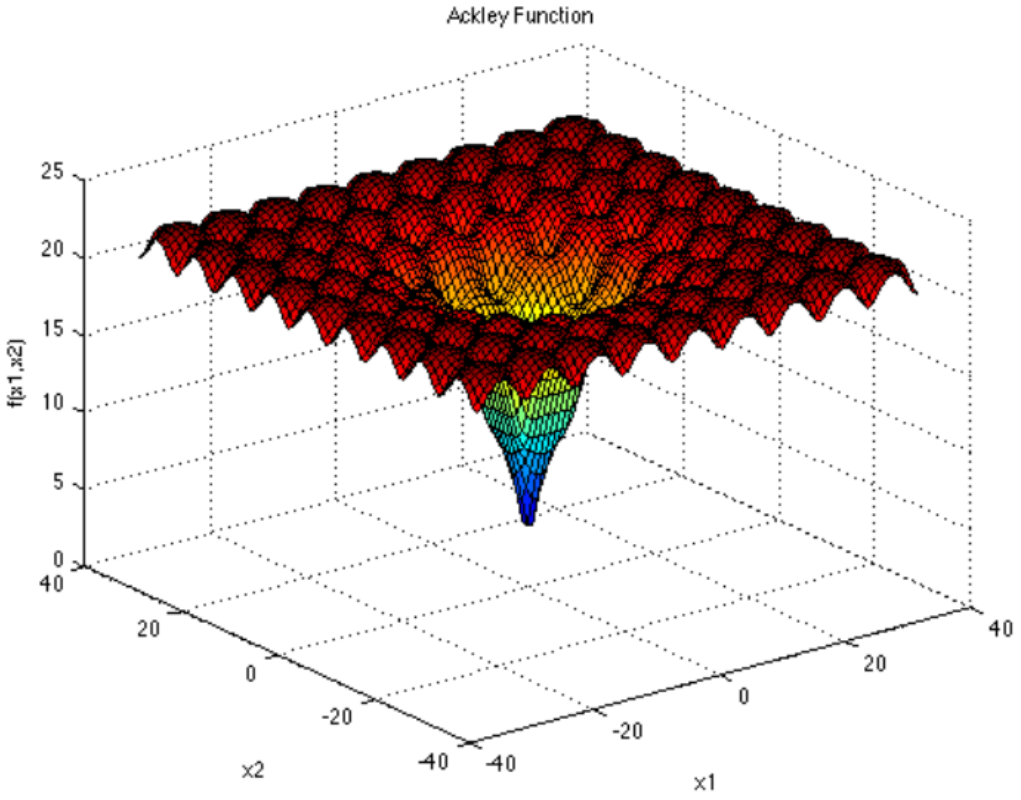
Drugim sposobem uruchomienia aplikacji jest zainstalowanie jej wszystkich zależności z wykorzystaniem dwóch komend i wcześniejszą instalacją Pythona oraz Node.js na naszej maszynie hostowej. Na poziomie pliku requirements.txt wykonujemy polecenie “python -m pip install -r requirements.txt”. Spowoduje to zainstalowanie następujących zależności: Django w wersji 4.0.3, numpy w wersji 1.22.3, django-cors-headers w wersji 3.11.0, matplotlib w wersji 3.5.1, pandas w wersji 1.4.2, openpyxl w wersji 3.0.9. Następnie musimy zainstalować zależności dla części frontendowej. Przechodzimy do katalogu klienta i wykonujemy polecenie “npm install”.

Dzięki powyższym wskazaniom będziemy w stanie uruchomić aplikacje zarówno kliencką jak i serwerową, na której znajduje się logika naszego algorytmu. Serwer uruchamiamy poleceniem “python -m manage runserver 8000” z poziomu pliku manage.py. Klient jest uruchamiany poleceniem “npm start” z poziomu folderu /client.

3. **Wykresy zależności wartości funkcji celu, średniej wartości funkcji celu oraz odchylenia standardowego w kolejnej iteracji i porównanie osiągniętych wyników przy różnych konfiguracjach algorytmu wraz z czasem obliczeń**

Wybrana przez nas funkcja celu to Ackley function w przypadku dwuwymiarowym.

1. ACKLEY FUNCTION



$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

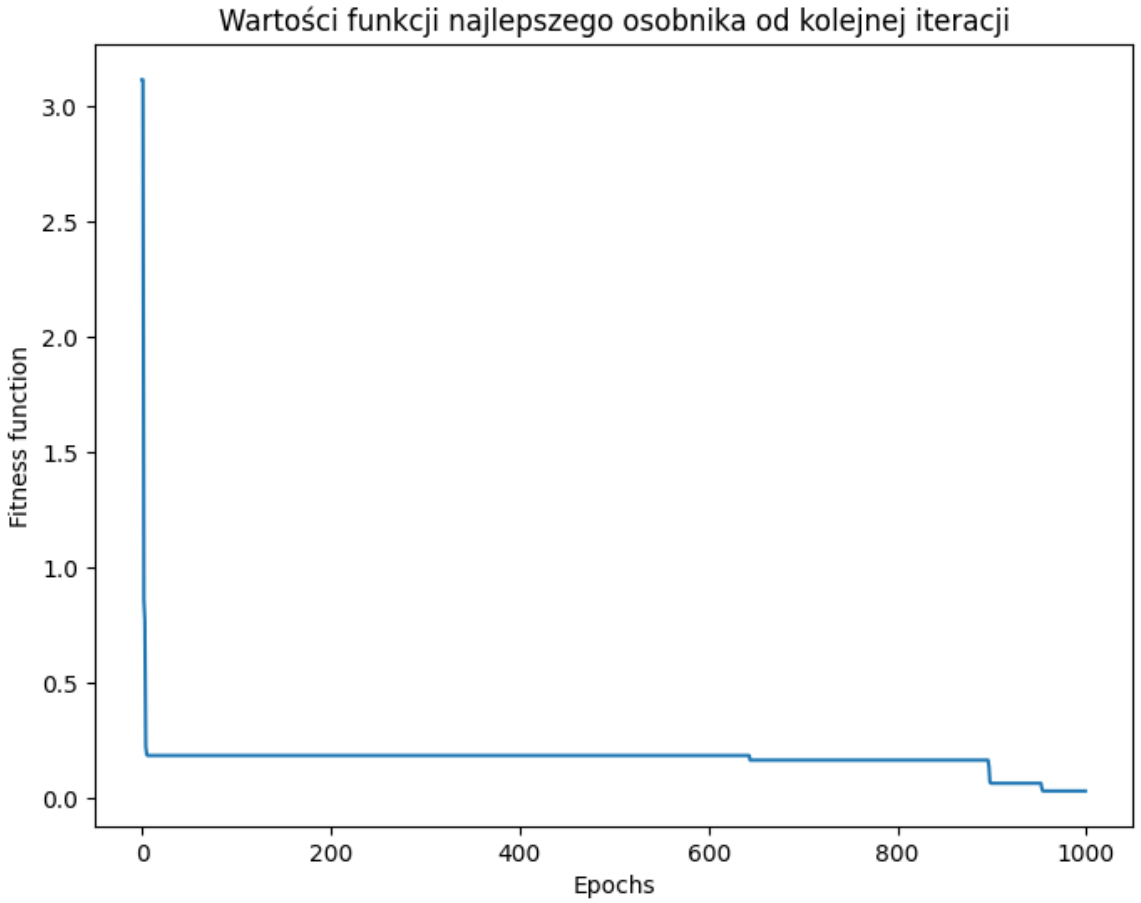
Parametry tej funkcji wynoszą: $a = 20,\ b = 0.2,\ c = 2\pi,\ d = 2$

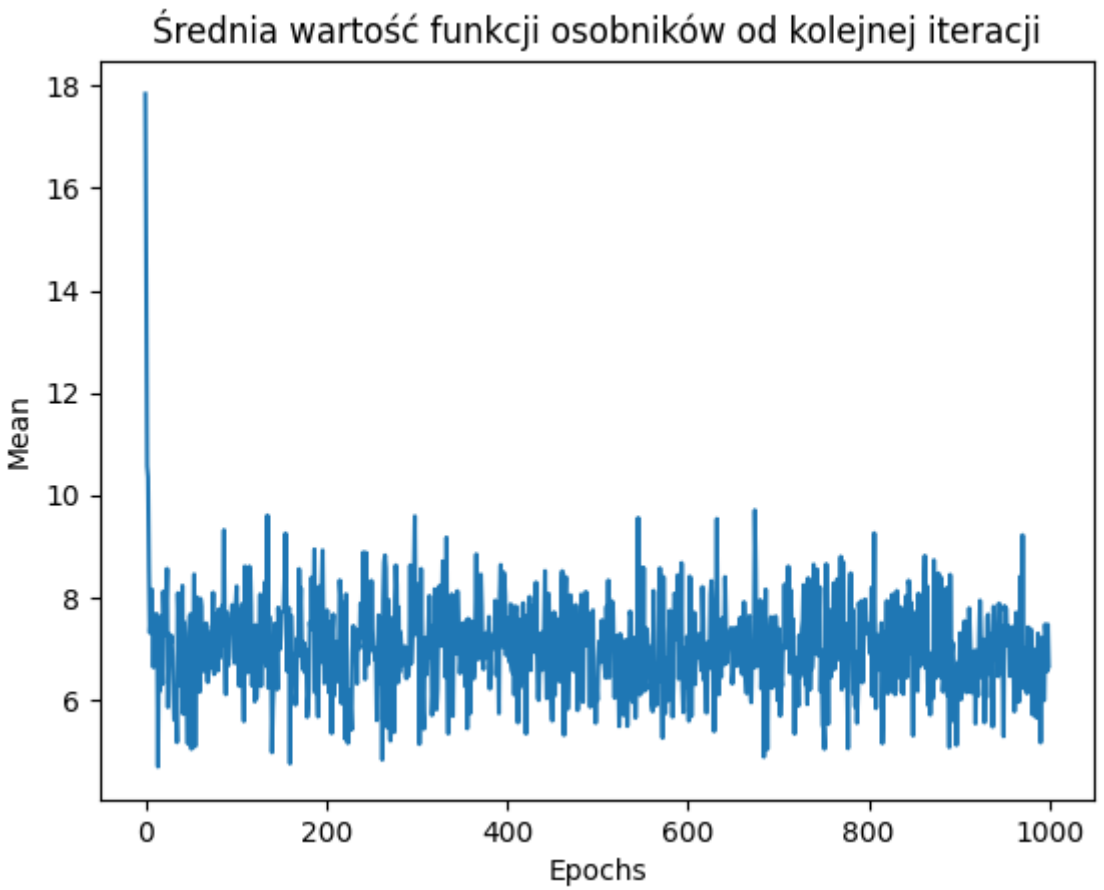
Dla każdego testu podane zostały:

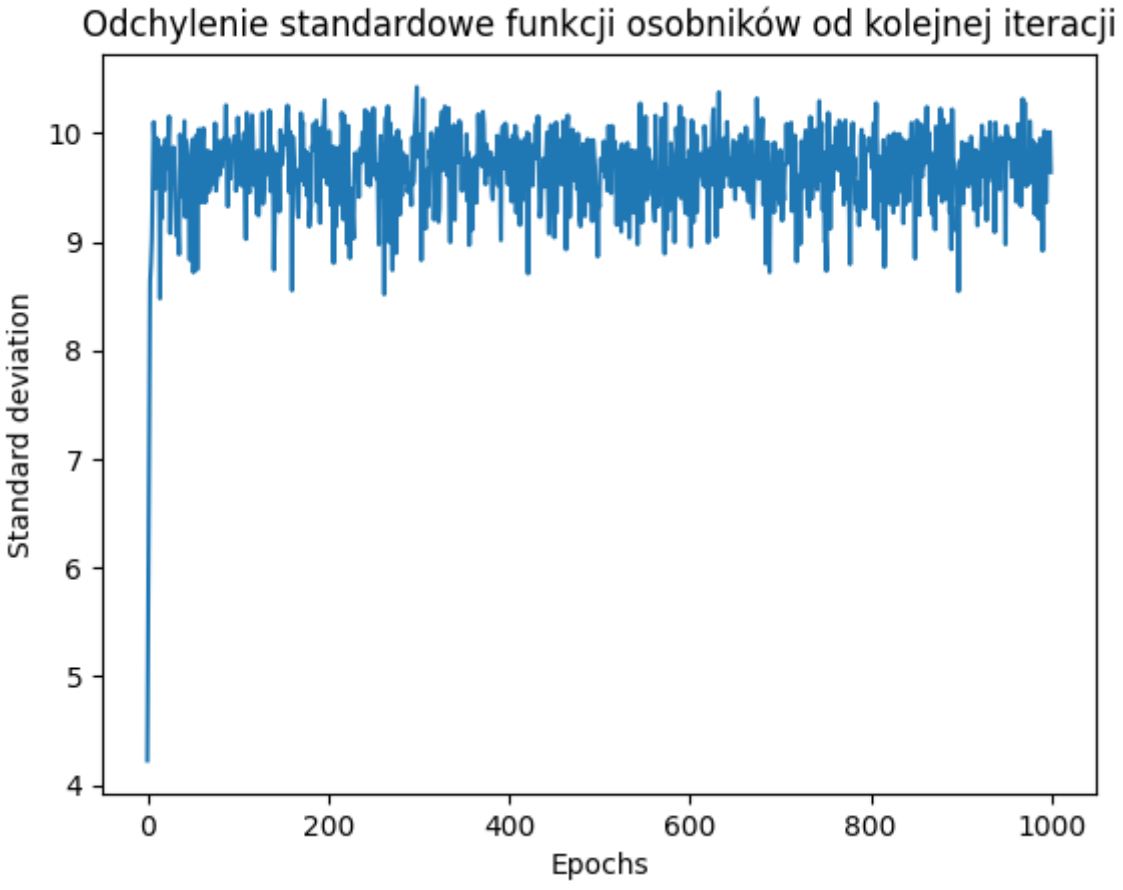
- parametry wejściowe,
- funkcja przedstawiająca wartość funkcji celu najlepszego osobnika w kolejnych epokach,
- funkcja przedstawiająca wartość średnią funkcji celu wszystkich osobników w kolejnych epokach
- funkcja przedstawiająca odchylenie standardowe funkcji celu wszystkich osobników w kolejnych epokach
- informację o powodzeniu algorytmu wraz z czasem potrzebnym na obliczenia.

a) Test nr 1

Number of population	**Elitist Strategy Percent**
100	0.1
Epochs	**Size of Tournament**
1000	4
Cross probability	**Selection Name**
0.6	Tournament
Mutation probability	**Crossover Name**
0.4	Arithmetic
Selection percent	**Mutation Name**
0.6	Uniform

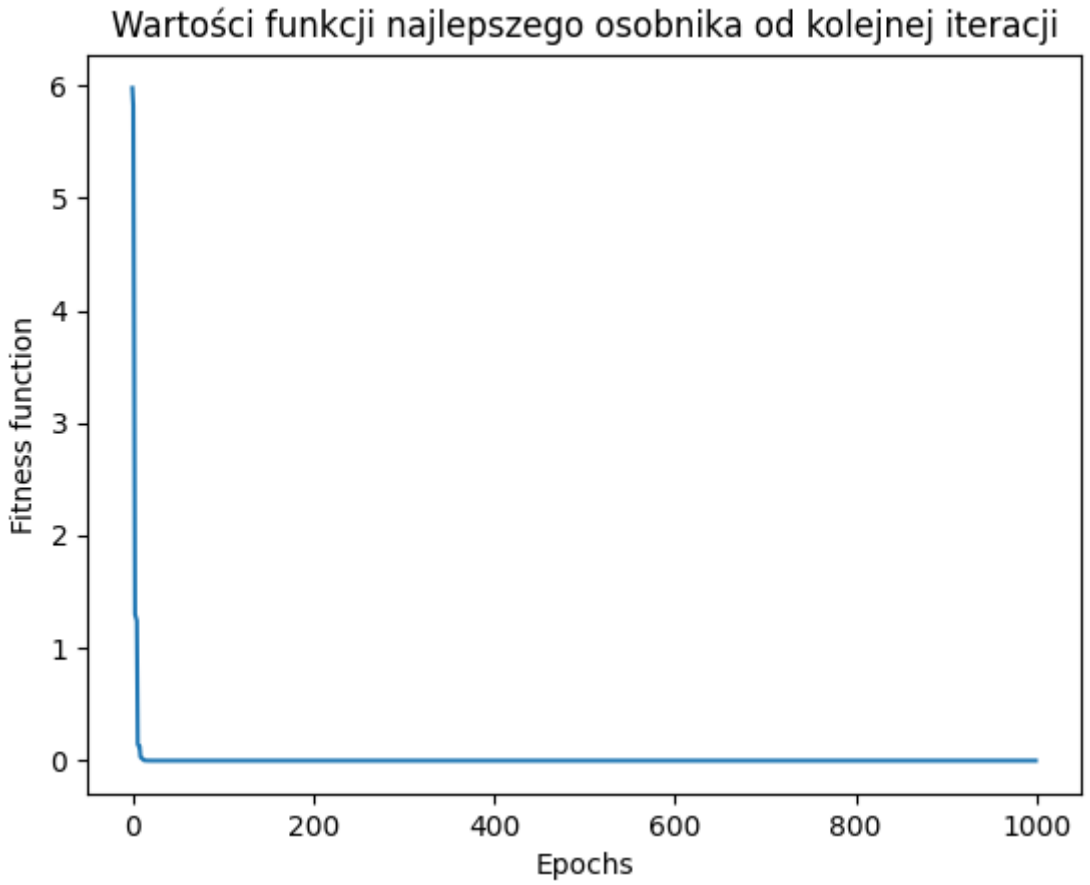


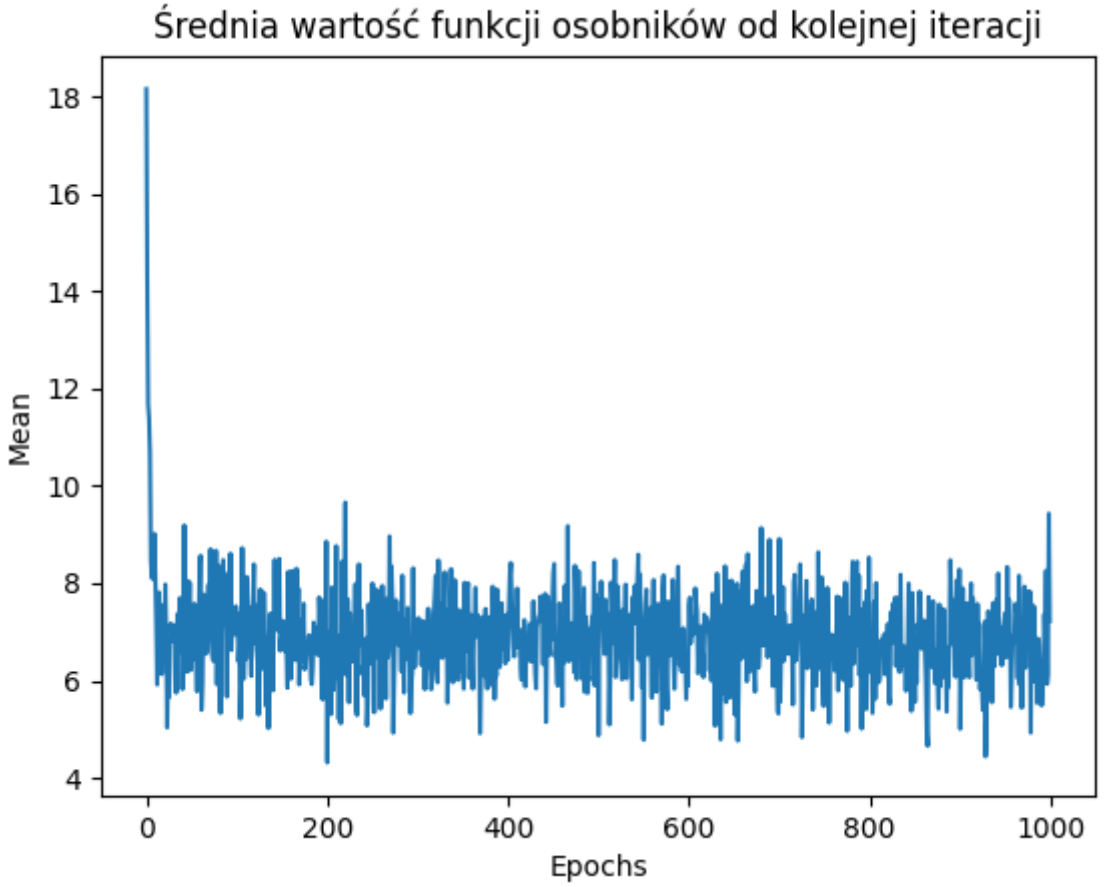

Średnia wartość funkcji osobników od kolejnej iteracji
Mean
Epochs
18
16
14
12
10
8
6
0
200
400
600
800
1000

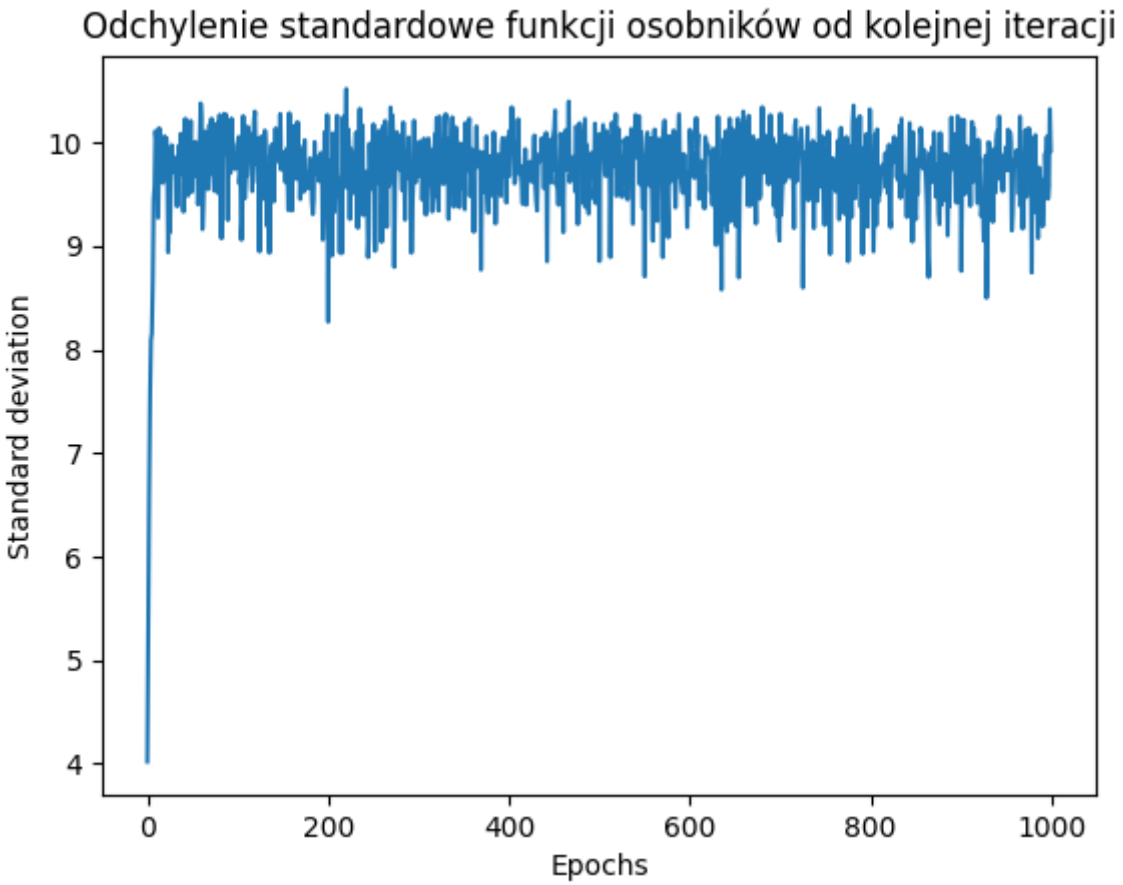

Odchylenie standardowe funkcji osobników od kolejnej iteracji
Standard deviation
Epochs
10
9
8
7
6
5
4
0
200
400
600
800
1000

Test został przeprowadzony dla populacji liczącej 100 osobników. Każdy z nich posiada po dwa chromosomy (jeden chromosom na jeden wymiar). Algorytm składał się elitarnej strategii (10%), selekcji turniejowej (60% + rozmiar turnieju 4), krzyżowania arytmetycznego (60%), mutacji jednorodnej (40%). Algorytm trwał 1000 epok. Na początku funkcja dopasowania najlepszego osobnika wynosiła około 3, ale bardzo szybko został on zastąpiony przez osobnika o wartości funkcji dopasowania 0,2. W okolicach 600, 900 i 950 epoki algorytm, aby ostatecznie funkcja przystosowania przybliżyła się do 0. Czas wykonania algorytmu wynosi 5,68 sekundy. W następnym teście została zmieniona metoda selekcji.

b) Test nr 2


Wartości funkcji najlepszego osobnika od kolejnej iteracji
Fitness function
Epochs
0
1
2
3
4
5
6
0
200
400
600
800
1000


Średnia wartość funkcji osobników od kolejnej iteracji
Mean
Epochs
4
6
8
10
12
14
16
18
0
200
400
600
800
1000

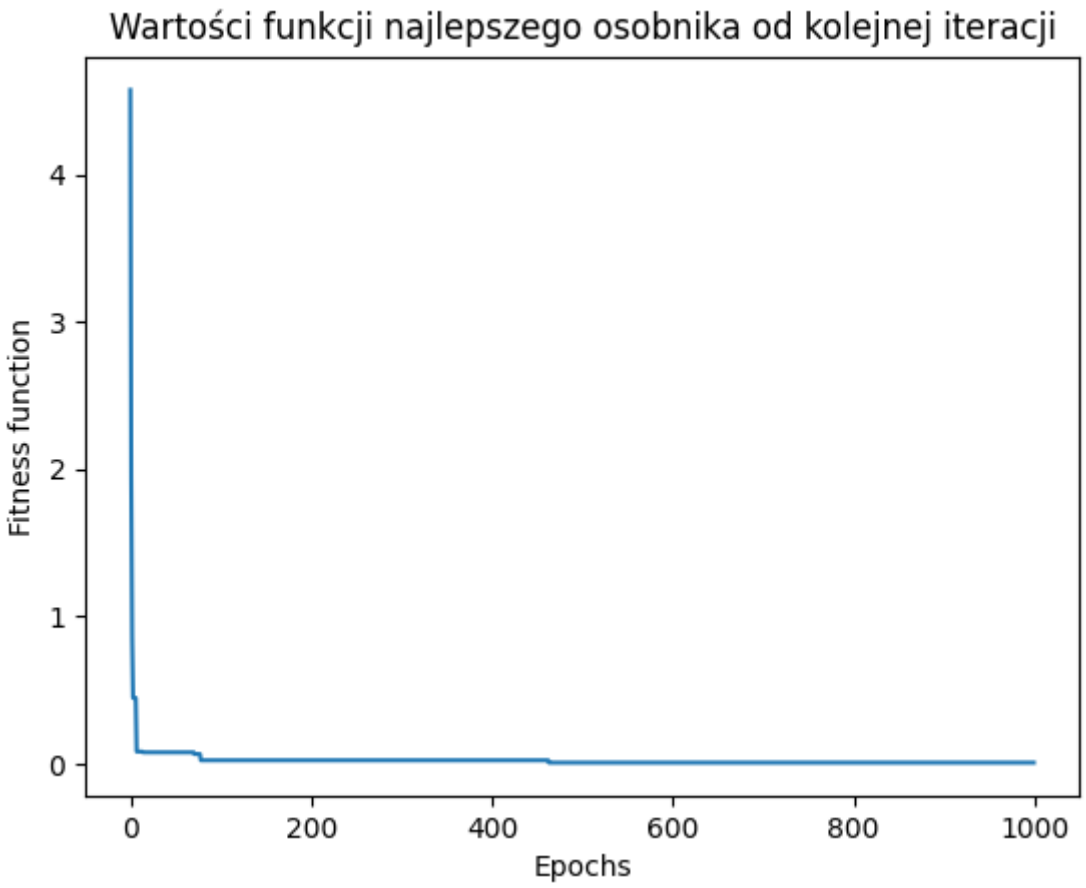


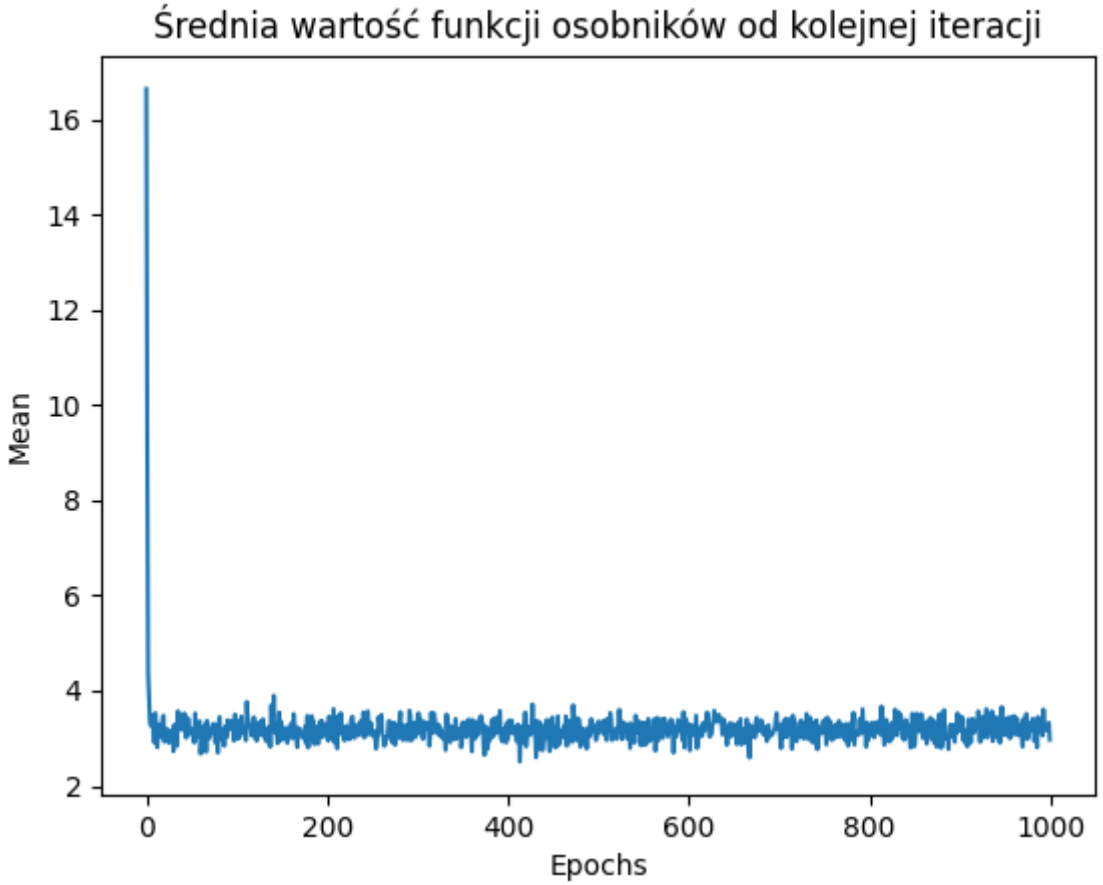
Status: OK
Execution time: 3.148015022277832 s

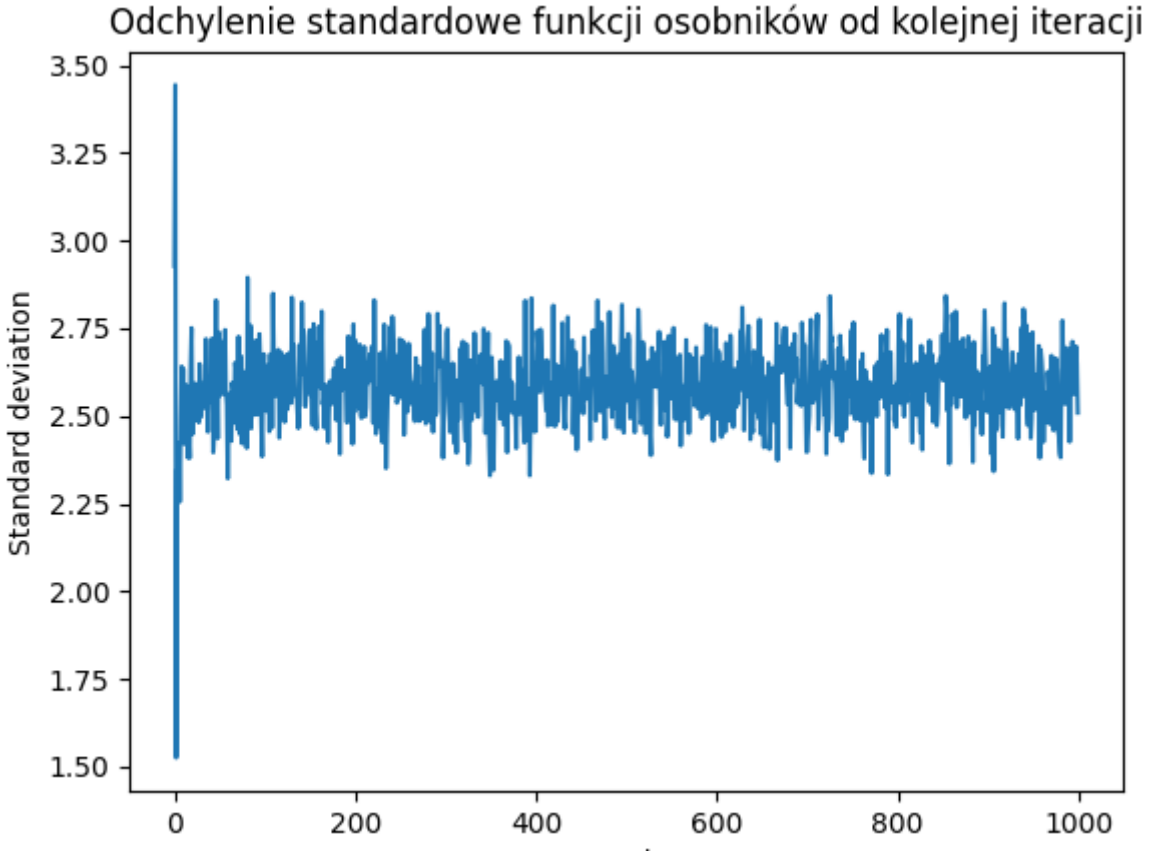
Parametry wejściowe testu nr 2 były prawie identyczne jak testu nr 1. Została zmieniona metoda selekcji z turniejowej na selekcję najlepszych. Metoda okazała się w tym teście bardziej efektywna na początku oraz szybsza - spadek z 5,68 sekundy do 3,15 sekundy. 1000 epok wydaje się zbyt dużym okresem, jeżeli weźmiemy pod uwagę liczbę zdarzeń, jakie mają miejsce. W związku z tym w następnym teście użyta zostanie mutacja Gaussa, dzięki której powinniśmy widzieć spadku w wartości funkcji celu w przeciągu trwania działania algorytmu..

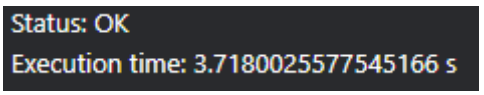
c) Test nr 3

Number of population	**Elitist Strategy Percent**
100	0.1
Epochs	**Size of Tournament**
1000	4
Cross probability	**Selection Name**
0.2	Roulette
Mutation probability	**Crossover Name**
0.8	Linear
Selection percent	**Mutation Name**
0.6	Gauss



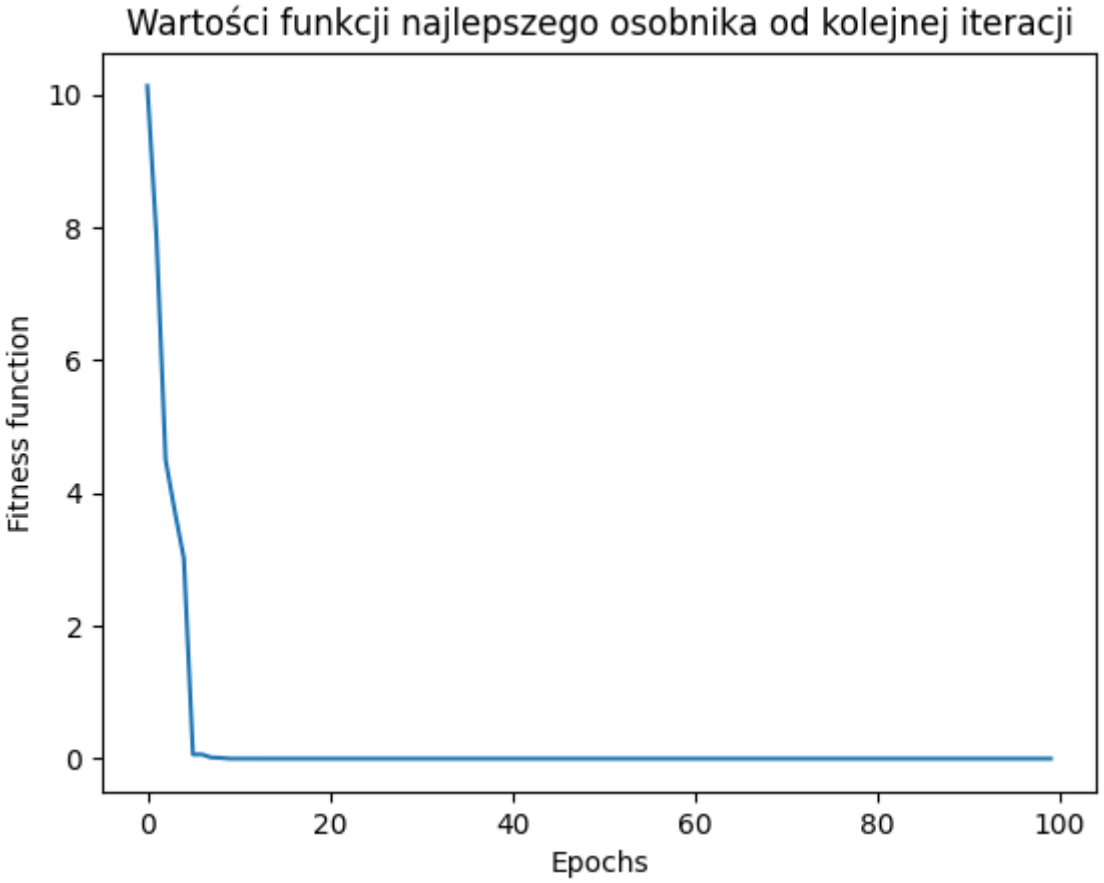

Średnia wartość funkcji osobników od kolejnej iteracji
Mean
Epochs
16
14
12
10
8
6
4
2
0
200
400
600
800
1000

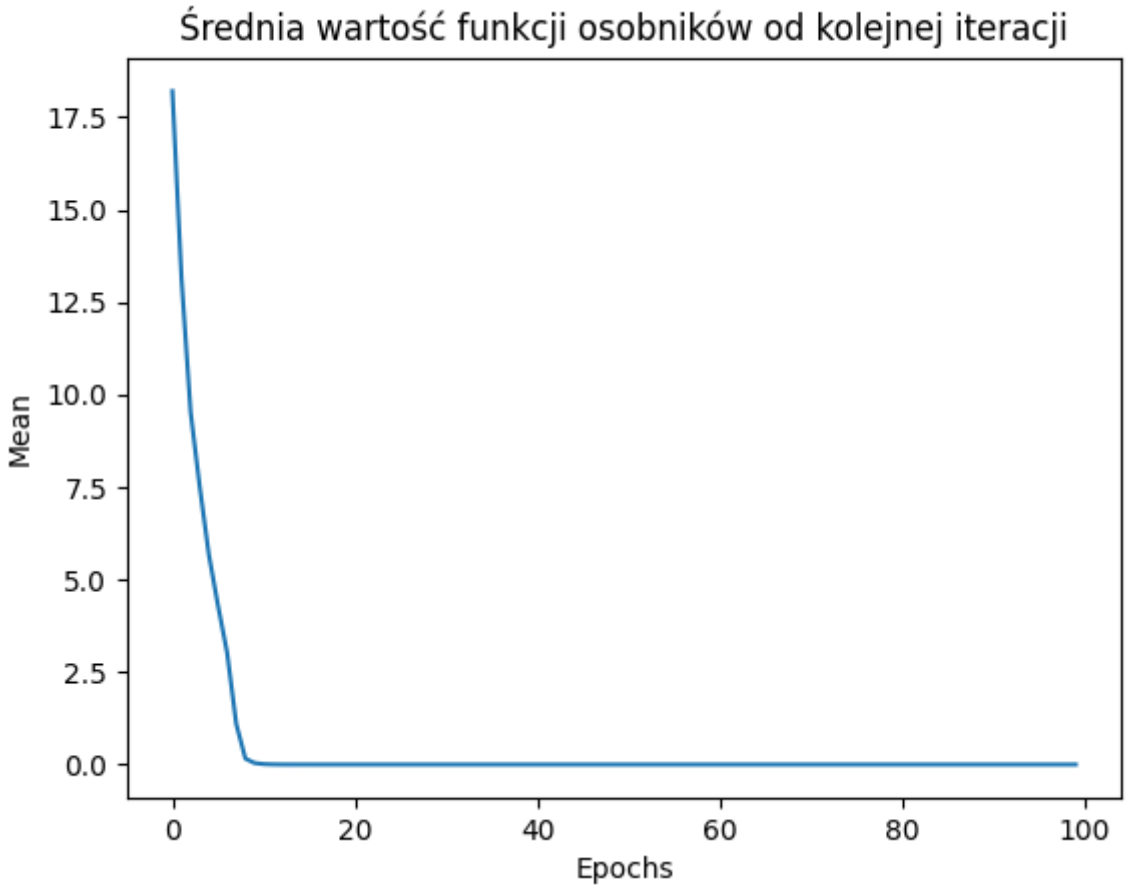

Odchylenie standardowe funkcji osobników od kolejnej iteracji
Standard deviation
3.50
3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
0
200
400
600
800
1000

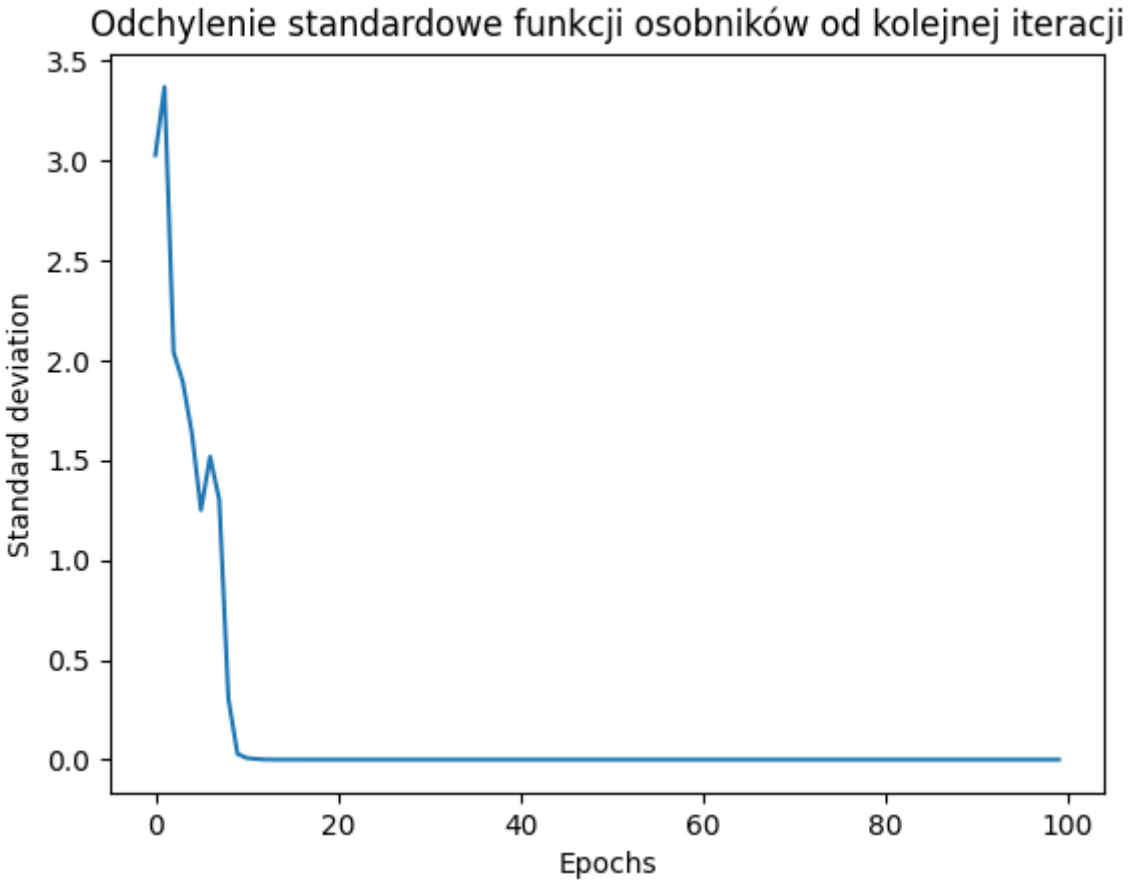


Głównym celem tej epoki jest sprawdzenie jak poradzi sobie algorytm, który przede wszystkim będzie przeprowadzał mutacje (80%), niż krzyżowania (20%). Tym razem algorytm korzysta z selekcji ruletki, krzyżowania liniowego i mutacji Gaussa. Wyniki wyszły dobre. Na wykresie można zauważyć niskie schodki, które zostały osiągnięte z użyciem odpowiedniej mutacji. Teraz sprawdzimy zachowanie algorytmu dla krzyżowania (100%) bez mutacji i inwersji.

d) Test nr 4


Wartości funkcji najlepszego osobnika od kolejnej iteracji
Fitness function
10
8
6
4
2
0
0
20
40
60
80
100
Epochs


Średnia wartość funkcji osobników od kolejnej iteracji
Mean
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80
100
Epochs

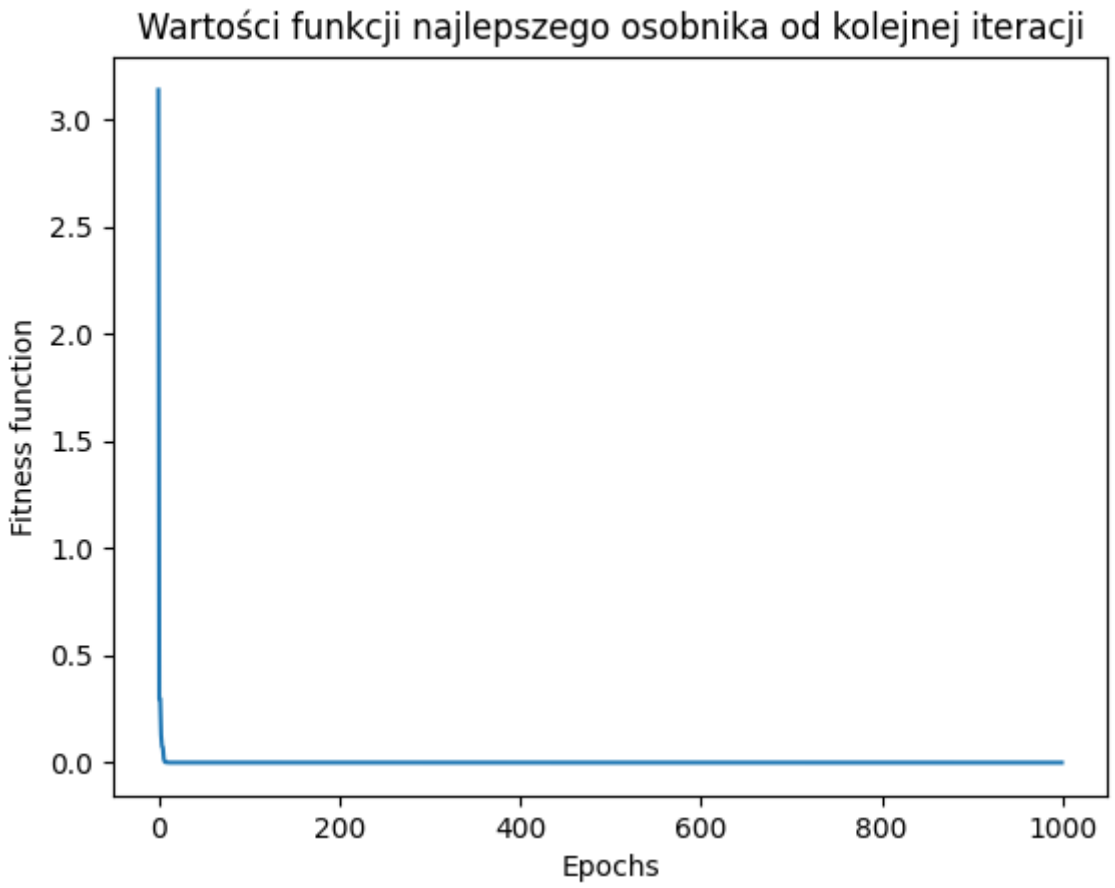


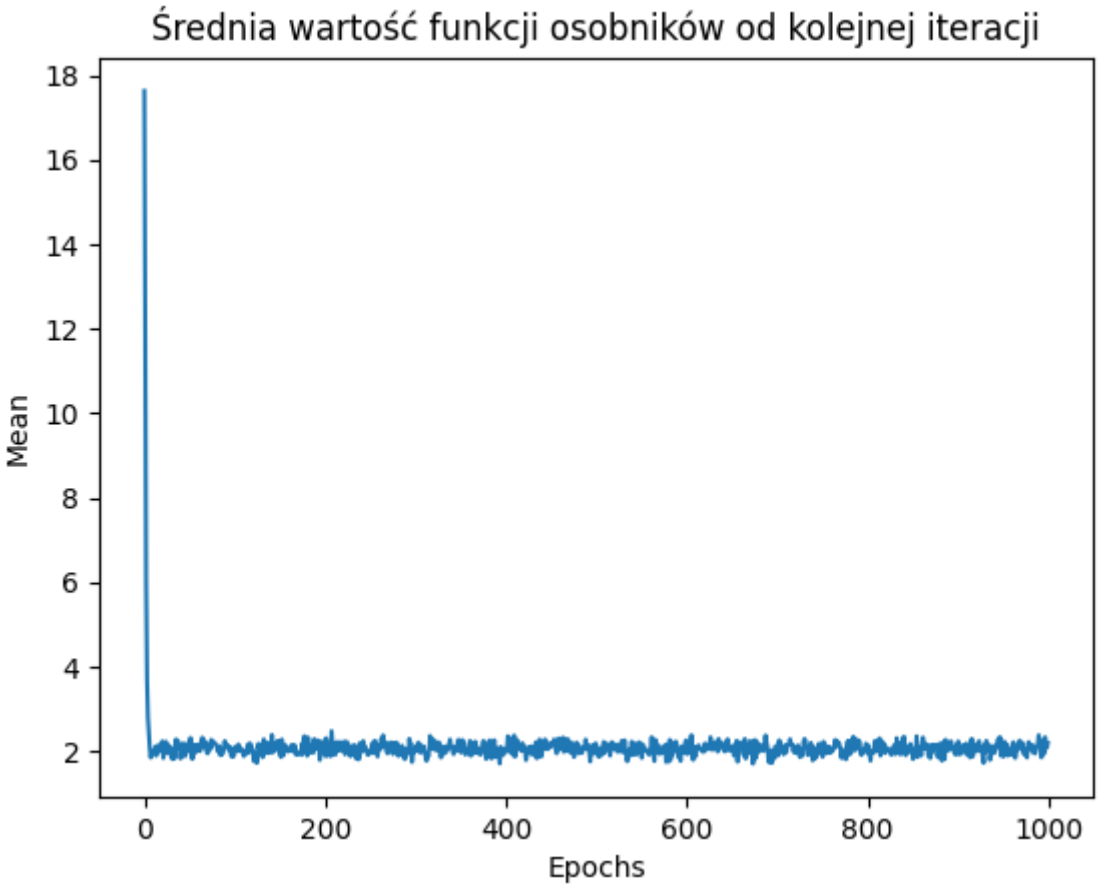

```
Status: OK
Execution time: 0.5559976100921631 s
```

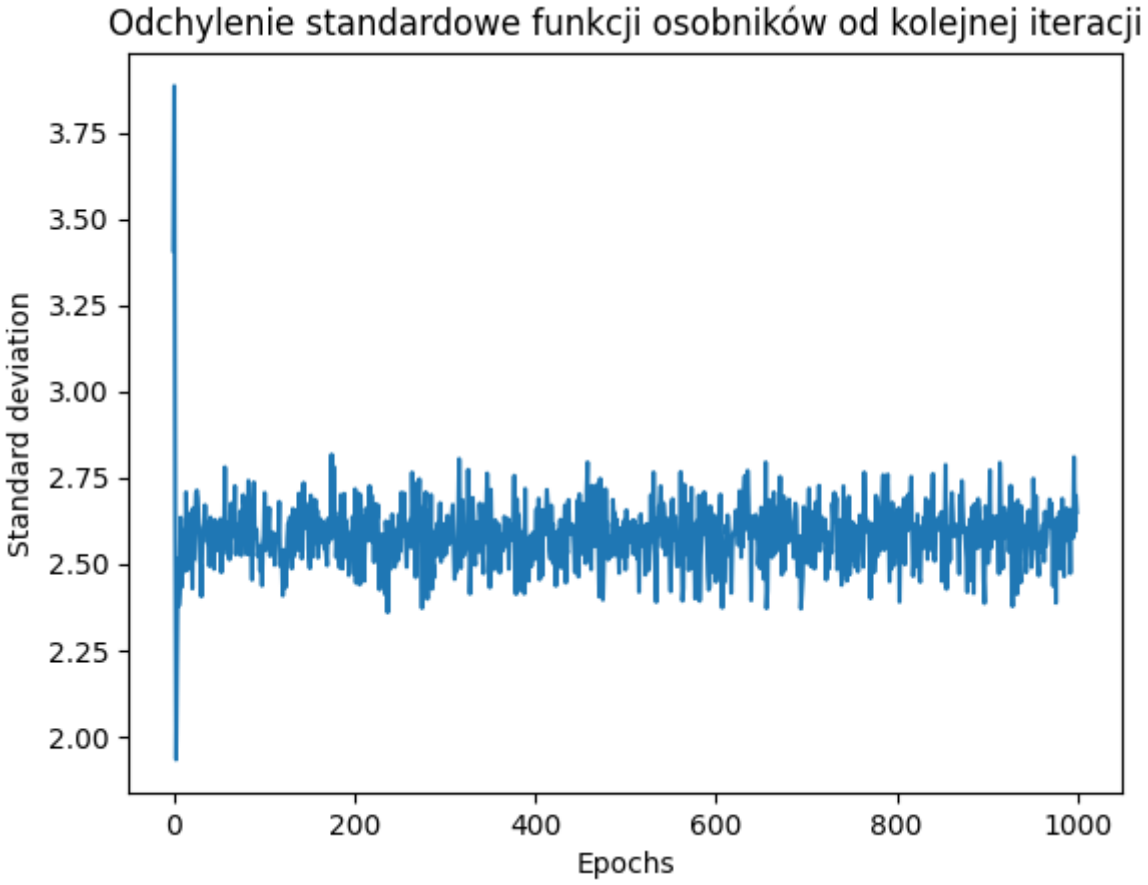
W algorytmie tym zostało użyte krzyżowanie mieszane alfa-beta (100%). Prawdopodobieństwo mutacji wynosi 0%. Wyniki wyszły zaskakująco dobrze, w kolejnych pięciu epokach wykres wartości funkcji najlepszego osobnika od kolejnej iteracji znacząco zbliżył się do zara. Osobniki krzyżowały się z samymi sobą, przez co w następnych epokach nie było żadnego progresu. W pierwszej epoce wartość funkcji celu dla najlepszego osobnika wynosiła 10, gdy w ostatniej epoce była to wartość bliska 0. W ostatnim teście spróbujemy uzyskać najlepszy wynik.

e) Test nr 5

Number of population	**Elitist Strategy Percent**
200	0.2
Epochs	**Size of Tournament**
1000	4
Cross probability	**Selection Name**
0.6	Tournament
Mutation probability	**Crossover Name**
0.6	Average
Selection percent	**Mutation Name**
0.6	Gauss




Średnia wartość funkcji osobników od kolejnej iteracji
Mean
Epochs
18
16
14
12
10
8
6
4
2
0
200
400
600
800
1000


Odchylenie standardowe funkcji osobników od kolejnej iteracji
Standard deviation
Epochs
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
0
200
400
600
800
1000

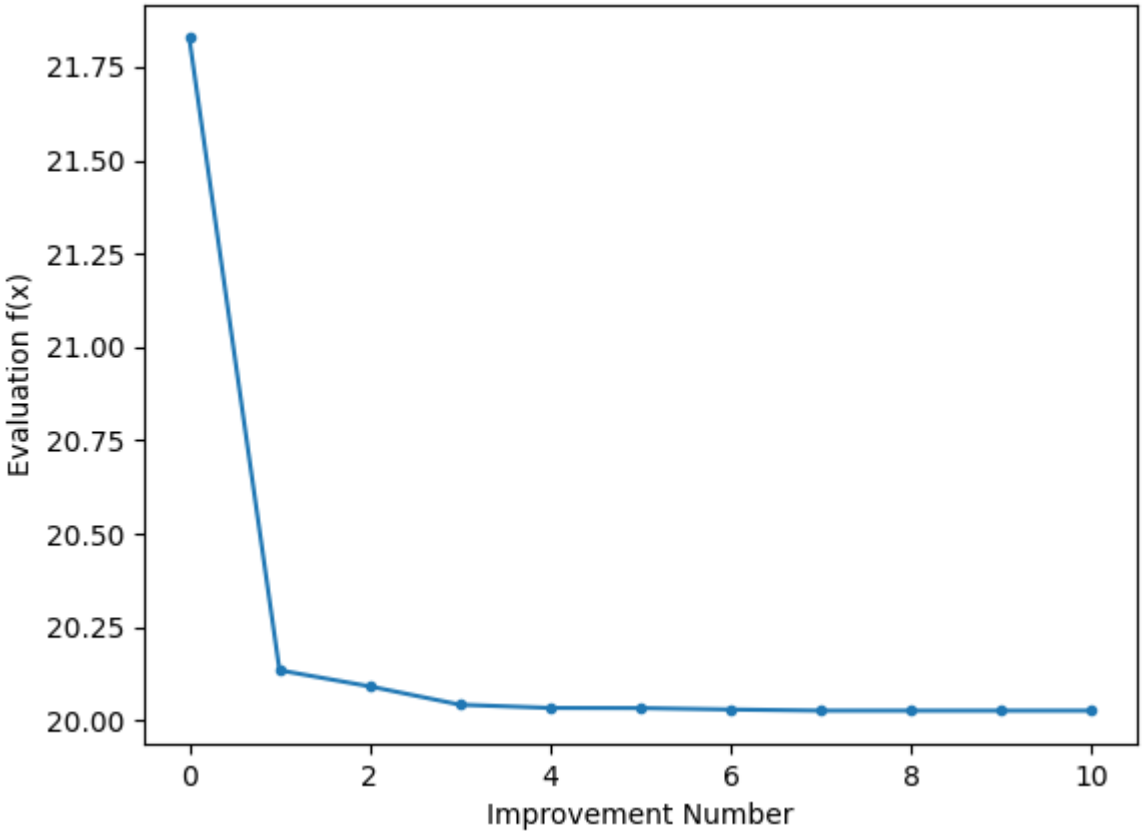
```
Status: OK
Execution time: 9.138001680374146 s
```

Ostatni test został przeprowadzony dla populacji liczącej 200 osobników. Algorytm składał się elitarnej strategii (20%), selekcji turniejowej (60% + rozmiar turnieju 4), krzyżowania uśredniającego (60%), mutacji Gaussa (60%). Algorytm trwał 1000 epok. Przez długi czas wydawało się, że wartość funkcji celu najlepszego osobnika nie zejdzie poniżej 0.5, ale około 820 epoki granica ta została przekroczona. Algorytm trwał około 9 sekund. Algorytm bardzo szybko zszedł do wartości bliskiej 0. Gdy zobaczymy na szczegółowe wyniki, możemy zobaczyć, że wartość funkcji celu najlepszego osobnika wynosi $6{,}66*10^{-7}$. Jest to bardzo dobry wynik.

```
--------------------------
|Fitness Function|
--------------------------
5.659435662330736e-07
```

4. Proste metaheurystyki

a) Algorytm wspinaczki:



Po uruchomieniu algorytmu wspinaczki dla funkcji Ackleya utknęliśmy w minimum lokalnym. Niestety wynik 20 nie jest zadowalający.

b) Algorytm Losowego Próbkowania

```
Result:
x: [0.10885984 0.47711506]
Fitness: 3.1573884100030374
```

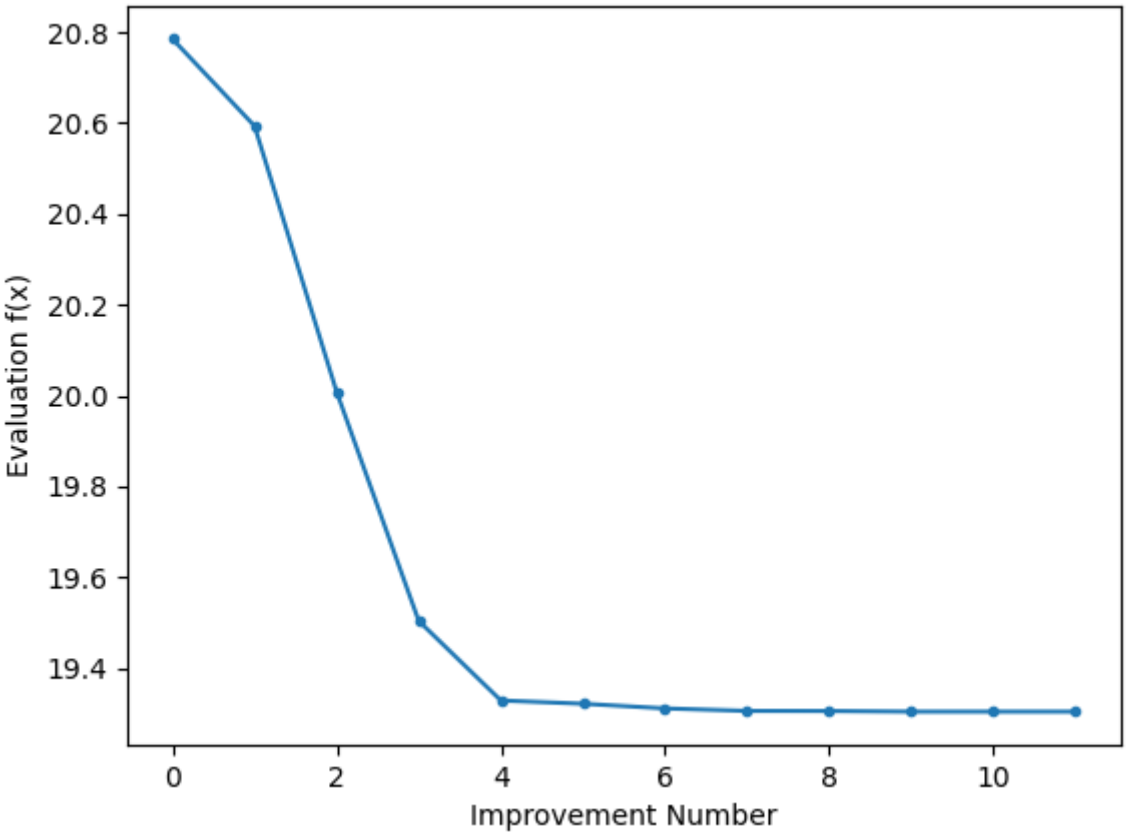
Algorytm mało praktyczny, ale w naszym wypadku, gdy funkcja posiada dużą liczbę minimum lokalnych, spełnia on swoją rolę. Po kilkukrotnym uruchomieniu algorytm podaje funkcję celu poniżej 1.

c) Algorytm Błądzenia Przypadkowego:

```
Result:
x: [ 289.40281702 -148.87924491]
Fitness: 21.763994406526525
```

Algorytm ten dał dotąd najgorsze wyniki. Bez sprawdzenia granic zmiennych algorytm wychodzi poza zakres (-40;40) dla funkcji celu.

d) Algorytm Symulowanego Wyżarzania:



Choć z początku algorytm szybko przybliża się do zera, to w pewnym momencie zdecydowanie zwalnia zatrzymując się na wartości 19,3 funkcji celu. Zwiększenie temperatury początkowej niestety nie pomaga w znalezieniu lepszego minimum.

5. Porównanie wyników

Porównując wyniki osiągnięte poprzez klasyczną reprezentację chromosomu, rzeczywistą reprezentację chromosomu oraz z użyciem prostych metaheurystyk możemy stwierdzić, że dla funkcji Ackleya najlepiej sprawuje się rzeczywista reprezentacja chromosomu. Korzystając z tej reprezentacji zeszliśmy do funkcji celu rzędu 10^{-7}, gdy w przypadku klasycznej reprezentacji było to 10^{-1}. Najgorzej wypadły proste metaheurystyki. Jeżeli chodzi o szybkość działania algorytmów, to reprezentacja rzeczywista lepiej wypada na tle reprezentacji klasycznej.