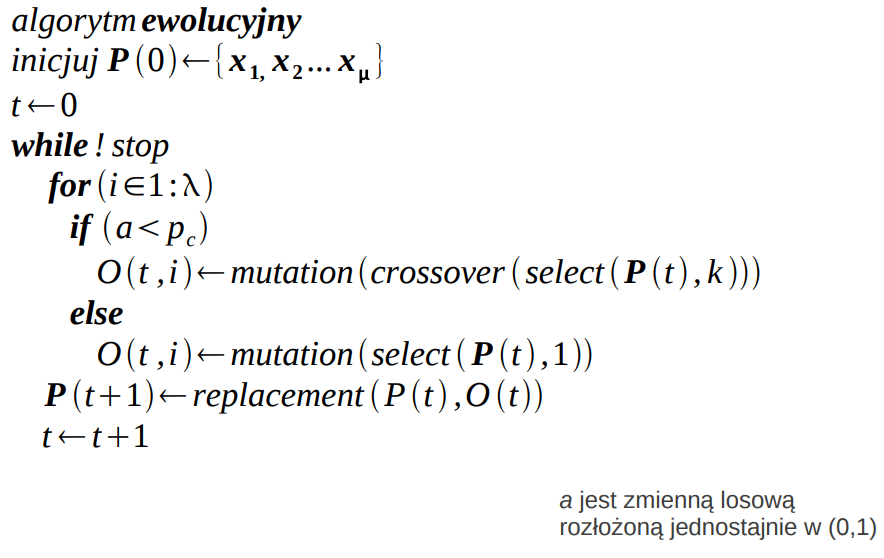
|  |  |
| --- | --- |
| **PSZT Projekt 1 Przeszukiwanie** | |
| **Tytuł projektu** | RB.S8 A może ślub? (bogata rodzina) |
| **Wykonujący projekt** | |
| Mateusz Bajdak | Nr albumu: 277125 |
| Tomasz Indeka | Nr albumu: 293457 |

# Definicja zadania.

Modyfikujemy klasyczny algorytm ewolucyjny losowo łącząc osobniki w pary (aż do śmierci). Niech funkcja oceny (podczas szukania minimum) dla każdego z osobników w parze zwraca minimum z funkcji oceny pary: .

Zadanie polega na porównaniu klasycznego algorytmu ewolucyjnego:



Rys. 1. Klasyczny algorytm ewolucyjny (pseudokod) (źródło: <https://staff.elka.pw.edu.pl/~jarabas/ALHE/wyklad6.pdf>)

ze zmodyfikowaną wersją opisaną na początku rozdziału 1 w kontekście zagadnienia minimalizacji wybranych funkcji testowych.

Zakładamy, że zmodyfikowany algorytm będzie znajdował optimum wolniej, ale finalnie osiągnie wynik co najmniej tak samo dobry jak standardowy algorytm ewolucyjny.

## Podział zadań w projekcie

Mateusz Bajdak:

* Implementacja parsera opcji wywołania
* Implementacja funkcji do testowania algorytmu
* Przeprowadzenie eksperymentów

Tomasz Indeka:

* Implementacja algorytmu ewolucyjnego
* Implementacja porównywania wyników

# Procedura testowania.

## Funkcja Ackleya

|  |
| --- |
| Obraz zawierający stół  Opis wygenerowany automatycznie |
|  |
| *Rys. 2. Dwuwymiarowa funkcja Ackleya (źródło:* [*http://infinity77.net/global\_optimization/test\_functions\_nd\_A.html*](http://infinity77.net/global_optimization/test_functions_nd_A.html)*)* |

## Alpine01

|  |
| --- |
|  |
|  |
| *Rys. 3. Dwuwymiarowa funkcja Alpine01 (źródło:* [*http://infinity77.net/global\_optimization/test\_functions\_nd\_A.html*](http://infinity77.net/global_optimization/test_functions_nd_A.html)*)* |

## Funkcja Bird

|  |
| --- |
|  |
|  |
| *Rys. 4. Dwuwymiarowa funkcja Bird (źródło:* [*http://infinity77.net/global\_optimization/test\_functions\_nd\_B.html*](http://infinity77.net/global_optimization/test_functions_nd_B.html)*)* |

## Funkcja Branin01

|  |
| --- |
|  |
|  |
| *Rys. 5. Dwuwymiarowa funkcja Branin01 (źródło:* [*http://infinity77.net/global\_optimization/test\_functions\_nd\_B.html*](http://infinity77.net/global_optimization/test_functions_nd_B.html)*)* |

## Funkcja Cigar

|  |
| --- |
|  |
|  |
| *Rys. 6. Dwuwymiarowa funkcja Cigar (źródło:* [*http://infinity77.net/global\_optimization/test\_functions\_nd\_C.html*](http://infinity77.net/global_optimization/test_functions_nd_C.html)*)* |

## Funkcja Griewanka

|  |
| --- |
|  |
|  |
| *Rys. 7. Dwuwymiarowa funkcja Griewanka (źródło:* [*http://infinity77.net/global\_optimization/test\_functions\_nd\_G.html*](http://infinity77.net/global_optimization/test_functions_nd_G.html)*)* |

## Parametry wywołania

Przewidziano następujące parametry wywołania algorytmu:

|  |  |
| --- | --- |
| **Iterations, i** | liczba wykonanych iteracji algorytmu ewolucyjnego |
| **dimensions, d** | liczba wymiarów optymalizowanej funkcji |
| **crossover\_p, C** | prawdopodobieństwo krzyżowania |
| **cardinality, n** | liczność populacji |
| **attempts, a** | liczba prób wykonania głównej pętli algorytmu |
| **x\_min, m** | ograniczenie od dołu wartości x pierwszej populacji |
| **x\_max, M** | ograniczenie od góry wartości x pierwszej populacji |
| **function, f** | rodzaj testowanej funkcji |
| **sel\_type, s** | typ selekcji |
| **rep\_type, r** | typ replikacji |
| **mut\_sigma, S** | odchylenie standardowe rozkładu normalnego wykorzystywanego przy mutacji |

## Dobór parametrów

Do celów testowania algorytmów ustaliliśmy wspólne niektóre parametry wywołania, aby móc porównywać ze sobą otrzymane wyniki:

* dimentions: 2, obliczenia wykonywane były na dokładnie takich funkcjach jak podane powyżej
* crossover\_p: 0.5, podczas testów okazała się być optymalna
* cardinality: 200, zapewnia nam rzetelną pulę testową
* attempts: 25, aby wyniki były miarodajne, ponieważ wykorzystujemy dużą ilość losowych operacji
* sel\_type: roulette, selekcja metodą ruletki
* rep\_type: generation+, zastępowanie metodą generacyjną µ+λ
* mut\_sigma: 5, podczas testów okazała się być optymalna

Warunkiem końcowym wykonania algorytmu była ilość iteracji, dobierana odpowiednio dla każdej badanej funkcji.

Pozostałe parametry były dobierane z uwzględnieniem cech poszczególnych funkcji.

## Wykorzystane narzędzia

Skrypt testujący algorytm ewolucyjny z parami został zaimplementowany w Pythonie.

Funkcje testujące zostały zaimplementowane przez nas z użyciem bibliotek dostępnych w Pythonie.

Do generacji wykresów użyliśmy biblioteki *matplotlib.*

## Wywołanie skryptu

Do wywołania skryptu należy mieć zainstalowane oprogramowanie Python i bibliotekę *matplotlib.*

Skrypt należy uruchomić z wiersza poleceń z folderu projektu wpisując komendę *python main\_app.py <opcja=wartość>*

Dostępne opcje widoczne są w tabeli w podpunkcie 2.7.

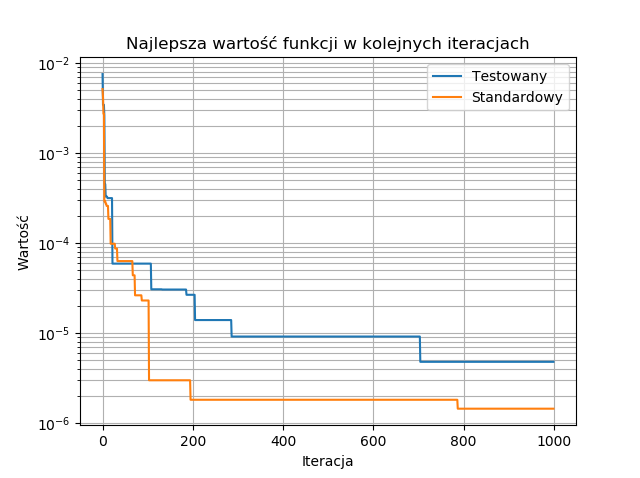
# Wyniki badań.

## Funkcja Ackleya (minimum: 0).

|  |
| --- |
| {      "iterations": 1000,      "dimensions": 2,      "crossover\_p": 0.5,      "cardinality": 200,      "attempts": 1,      "x\_min": -32.0,      "x\_max": 32.0,      "function": "ackley",      "sel\_type": "roulette",      "rep\_type": "generation+",      "mut\_sigma": 5  } |
| *Rys. 8. Parametry uruchomieniowe algorytmu dla funkcji Ackleya.* |

Minimum dla algorytmu tradycyjnego: **0.0014585412932004438**

Minimum dla algorytmu zmodyfikowanego: **0.004826041801653691**



Rys. 9. Wykres zależności minimalnej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Ackleya.

Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Rys. 10. Wykres zależności średniej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Ackleya.

Obraz zawierający mapa

Opis wygenerowany automatycznie

Rys. 11. Wykres zależności wariancji wartości funkcji optymalizowanej od numeru iteracji dla funkcji Ackleya.

## Alpine01 (minimum: 0).

|  |
| --- |
| {      "iterations": 1000,      "dimensions": 2,      "crossover\_p": 0.5,      "cardinality": 200,      "attempts": 1,      "x\_min": -10.0,      "x\_max": 10.0,      "function": "alpine-1",      "sel\_type": "roulette",      "rep\_type": "generation+",      "mut\_sigma": 5  } |
| *Rys. 12. Parametry uruchomieniowe algorytmu dla funkcji Alpine01.* |

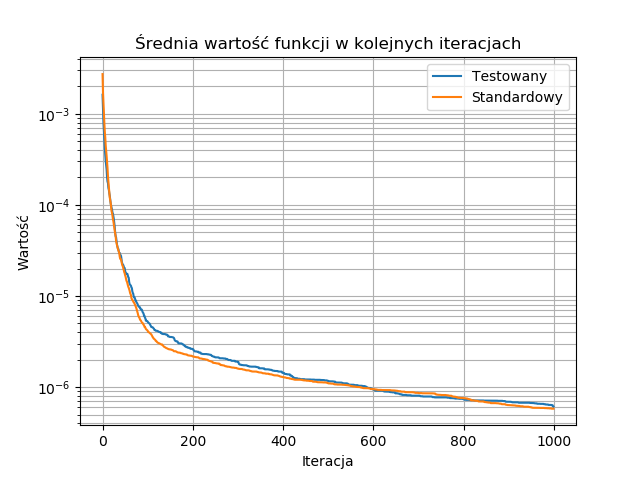
Minimum dla algorytmu tradycyjnego: **8.574258015428589e-05**

Minimum dla algorytmu zmodyfikowanego: **0.00012622519870459162**

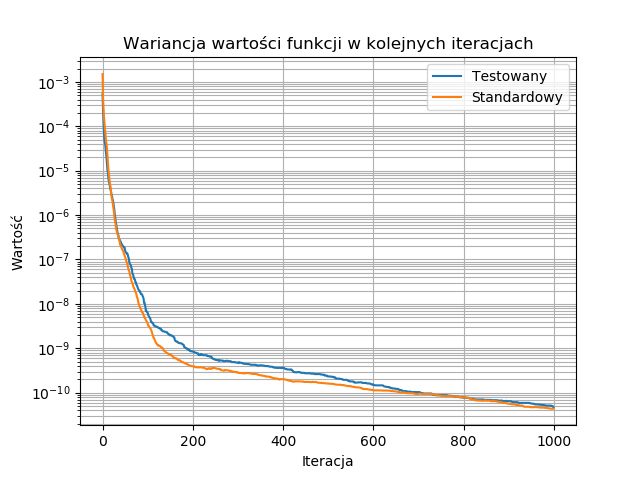
Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Rys. 13. Wykres zależności minimalnej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Alpine01.



Rys. 14. Wykres zależności średniej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Alpine01.



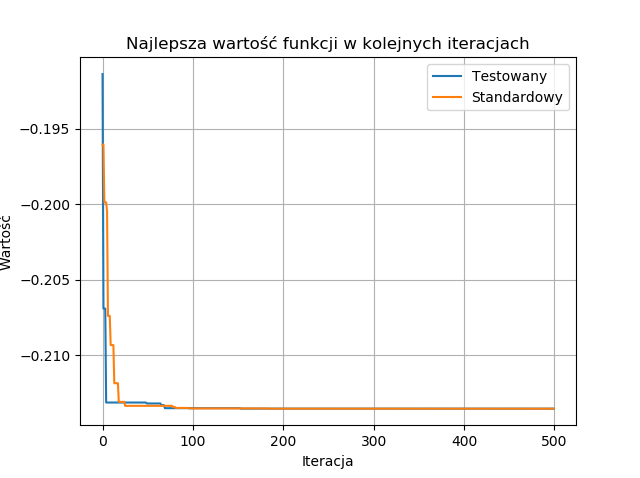
Rys. 15. Wykres zależności wariancji wartości funkcji optymalizowanej od numeru iteracji dla funkcji Alpine01.

## Funkcja Bird (minimum: -106,7645367198034).

|  |
| --- |
| {      "iterations": 500,      "dimensions": 2,      "crossover\_p": 0.5,      "cardinality": 200,      "attempts": 1,      "x\_min": -6.28,      "x\_max": 6.28,      "function": "bird",      "sel\_type": "roulette",      "rep\_type": "generation+",      "mut\_sigma": 5  } |
| *Rys. 16. Parametry uruchomieniowe algorytmu dla funkcji Bird.* |

Minimum dla algorytmu tradycyjnego: **-106.76449085940331**

Minimum dla algorytmu zmodyfikowanego: **-106.76371103219286**



Rys. 17. Wykres zależności minimalnej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Bird.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rys. 18. Wykres zależności średniej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Bird.

Obraz zawierający mapa

Opis wygenerowany automatycznie

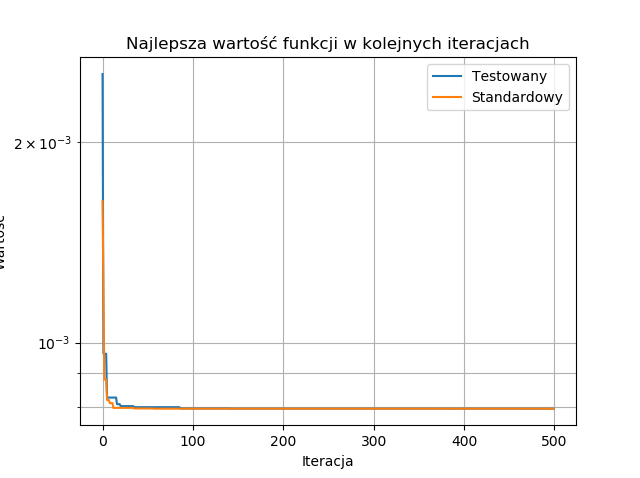
Rys. 19. Wykres zależności wariancji wartości funkcji optymalizowanej od numeru iteracji dla funkcji Bird.

## Funkcja Branin01 (minimum: 0.39788735772973816).

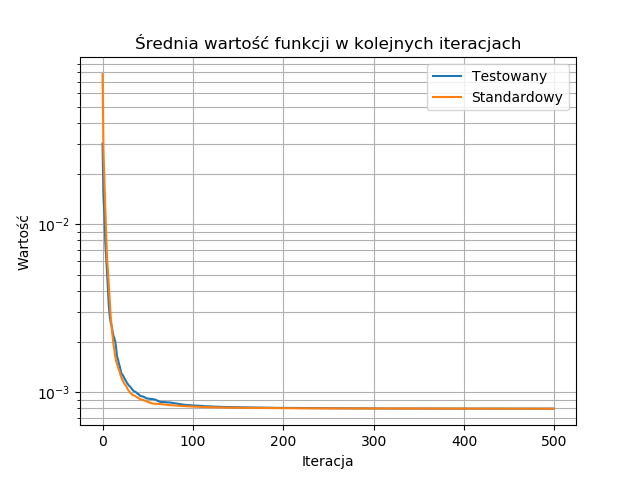
|  |
| --- |
| {      "iterations": 500,      "dimensions": 2,      "crossover\_p": 0.5,      "cardinality": 200,      "attempts": 1,      "x\_min": -15,      "x\_max": 15,      "function": "branin-1",      "sel\_type": "roulette",      "rep\_type": "generation+",      "mut\_sigma": 5  } |
| *Rys. 20. Parametry uruchomieniowe algorytmu dla funkcji Branin01.* |

Minimum dla algorytmu tradycyjnego: **0.3978907324735612**

Minimum dla algorytmu zmodyfikowanego: **0.397901875173968**



Rys. 21. Wykres zależności minimalnej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Branin01.



Rys. 22.Wykres zależności średniej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Branin01.

Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Rys. 23. Wykres zależności wariancji wartości funkcji optymalizowanej od numeru iteracji dla funkcji Branin01.

## Funkcja Cigar (minimum: 0).

|  |
| --- |
| {      "iterations": 1000,      "dimensions": 2,      "crossover\_p": 0.5,      "cardinality": 200,      "attempts": 1,      "x\_min": -100,      "x\_max": 100,      "function": "cigar",      "sel\_type": "roulette",      "rep\_type": "generation+",      "mut\_sigma": 5  } |
| *Rys. 24. Parametry uruchomieniowe algorytmu dla funkcji Cigar.* |

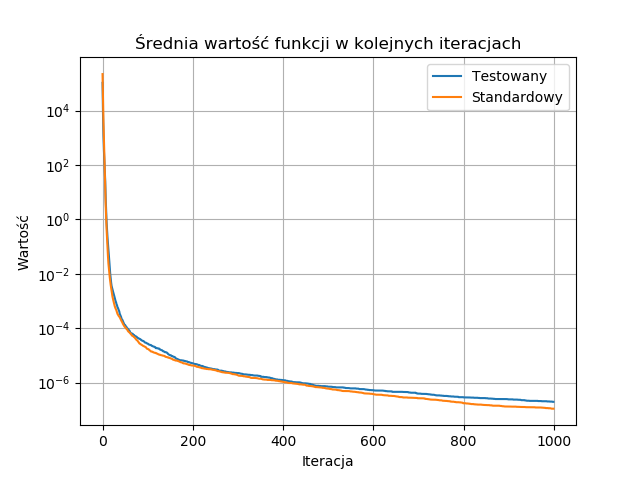
Minimum dla algorytmu tradycyjnego: **8.755544823133665e-07**

Minimum dla algorytmu zmodyfikowanego: **5.953699312814477e-06**

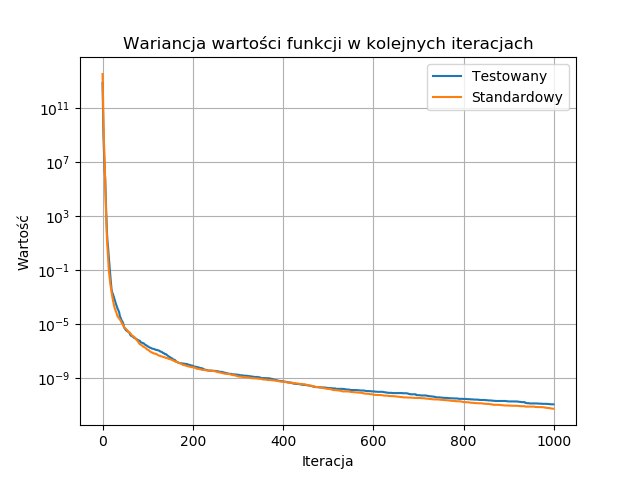
Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Rys. 25. Wykres zależności minimalnej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Cigar.



Rys. 26. Wykres zależności średniej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Cigar.



Rys. 27. Wykres zależności wariancji wartości funkcji optymalizowanej od numeru iteracji dla funkcji Cigar.

## Funkcja Griewanka (minimum: 0).

|  |
| --- |
| {      "iterations": 5000,      "dimensions": 2,      "crossover\_p": 0.5,      "cardinality": 200,      "attempts": 1,      "x\_min": -10,      "x\_max": 10,      "function": "griewank",      "sel\_type": "roulette",      "rep\_type": "generation+",      "mut\_sigma": 5  } |
| *Rys. 28. Parametry uruchomieniowe algorytmu dla funkcji Griewanka.* |

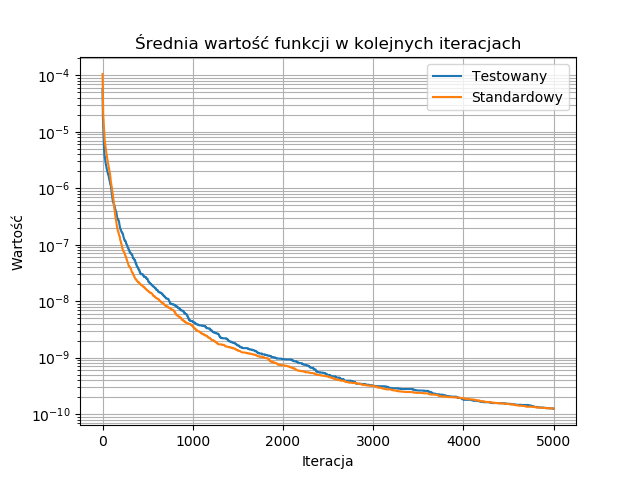
Minimum dla algorytmu tradycyjnego: **3.800689429844795e-09**

Minimum dla algorytmu zmodyfikowanego: **2.8388669193191163e-10**

Obraz zawierający mapa

Opis wygenerowany automatycznie

Rys. 29. Wykres zależności minimalnej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Griewanka.



Rys. 30. Wykres zależności średniej wartości funkcji optymalizowanej od numeru iteracji dla funkcji Griewanka.

Obraz zawierający tekst, mapa

Opis wygenerowany automatycznie

Rys. 31. Wykres zależności wariancji wartości funkcji optymalizowanej od numeru iteracji dla funkcji Griewanka.

# Wnioski i obserwacje.

Jak możemy zaobserwować wyniki otrzymane za pomocą testowanego algorytmu są bardzo zbliżone do tych otrzymanych za pomocą standardowego algorytmu. Oba pomyślnie znajdują minimum zadanej funkcji, choć algorytm standardowy zdaje się to robić szybciej. Jest to spowodowane przez utrzymywanie przez testowany algorytm informacji o niektórych punktach z niekorzystnym położeniem w przestrzeni, ale posiadających partnera o dobrym dopasowaniu. Standardowy algorytm nie przechowuje informacji o źle dopasowanych osobnikach przez co dzięki mutacji w okolicy optimum funkcji jest w stanie szybciej znaleźć rozwiązanie.

Przechowywanie informacji o źle dopasowanych osobnikach może prowadzić do znajdowania minimów znajdujących się za dużymi słabymi obszarami funkcji celu. Z naszych obserwacji wynika jednak, że informacja o źle dopasowanych osobnikach była często tracona w wyniku krzyżowania i znajdowania lepszych rozwiązań, przez co para ze słabym osobnikiem nie była już tak dobrze dopasowana. Niemniej jednak na wykresach można zauważyć, że w przypadku testowanego algorytmu wariancja otrzymanych funkcji celu jest większa co może świadczyć o przeszukiwaniu większej przestrzeni.