Projekt 1.

Implementacja klasycznego algorytmu genetycznego.

Student: Daniel Terkała

Album: 121122

Kierunek: Data Science

Zadanie do wykonania

Podjęcie próby implementacji klasycznego algorytmu genetycznego służącemu optymalizacji wybranej funkcji celu (co najmniej dwóch zmiennych). W ramach rozwiązania należało zaimplementować podstawowe operatory genetyczne (krzyżowanie, mutacja oraz inwersja) oraz ich różne wariacje. Oprócz operatorów należało zakodować różne typy selekcji osobników poddawanych reprodukcji, a także niezbędne metody pozwalające na ewaluację znajdowanych rozwiązań oraz iteracyjny algorytm bazowy służący spięciu całego systemu służącemu minimalizacji funkcji. W implementacji skorzystano z paradygmatu obiektowego, a sama aplikacja wyposażona została w prosty, aczkolwiek (mam nadzieję) przejrzysty i zrozumiały interfejs graficzny.

Opis rozwiązania:

1. **Wykorzystane technologie oraz środowisko uruchomieniowe.**
   Do implementacji wykorzystany został język python 3.7 oraz jego następujące moduły:
   - Tkinter – wbudowana biblioteka umożliwiająca budowanie interfejsów graficznych użytkownika
   - Matplotlib – wbudowana biblioteka umożliwiająca generowanie wykresów
   - Math – wbudowana biblioteka udostępniająca gotowe do użycia rozmaite funkcje matematyczne
   - Timeit – biblioteka udostępniająca narzędzie umożliwiające pomiary czasów

Rozwiązanie zostało wykonane pod systemem Windows 10, docelowo zakłada się wspieranie tej właśnie platformy, choć przy zaopatrzeniu się w odpowiednią wersję interpretera inne systemy również powinny być obsługiwane.

2. Koncepcja rozwiązania:

Oś algorytmu stanowi pętla, której celem jest iterowanie po kolejnych epokach, w każdej z epok budowana jest kolejna generacja rozwiązań, złożona z chromosomów będących produktem reprodukcji (tzn. potomków) powstałych ze specjalnie wyselekcjonowanych w tym celu rozwiązań kandydujących (inaczej zwanych rodzicami). Rozwiązania kandydujące wybierane są z populacji wygenerowanej w iteracji bezpośrednio poprzedzającej aktualną. Reprodukcja natomiast polega na poddaniu wyselekcjonowanej pary rodziców procesowi umożliwiającemu wymianę poszczególnych genów, które je charakteryzują. Jeśli zajdą odpowiednie ku temu warunki (określone parametrem opisującym prawdopodobieństwo wystąpienia) następuje krzyżowania, które w zależności od typu polega na wymianie wybranych allei (wartości poszczególnych bitów) pomiędzy chromosomami. Tak utworzone potomstwo może zostać poddane mutacji, polegającej na odwróceniu wartości wybranych bitów oraz inwersji determinującej odwrócenie wartości wszystkich alleli. Zarówno mutacja, jak i inwersja zachodzą z pewnym, danym prawdopodobieństwem. Liczba iteracji pętli głównej algorytmu, zostaje określona przez specjalny parametr, jej działanie może zostać jednak zakończone wcześniej, w przypadku, gdy osiągnięta zostanie satysfakcjonująca dokładność znalezionego rozwiązania.

3. Wymagania funkcjonalne:

1.1. Parametry definiujące działanie głównego algorytmu takie jak: liczba epok, rozmiar populacji, pożądana dokładność szukanego rozwiązania oraz typ optymalizacji są definiowane przez użytkownika.

1.2. Parametry charakteryzujące operatory genetyczne, takie jak: typ krzyżowania oraz parametr pomocniczy krzyżowania, typ mutacji, prawdopodobieństwo wystąpienia mutacji, liczba bitów poddanych mutacji, prawdopodobieństwo zajścia inwersji, czy stosowanie strategii elitarnej, wraz z określeniem, jaka część najlepszych osobników populacji zostaje zachowana do kolejnej generacji, są definiowane przez użytkownika.

1.3. Aplikacja ma posiadać interfejs graficzny umożliwiający:

- 1.3.1. Wprowadzanie powyższych parametrów.
- 1.3.2. Wyświetlenia znalezionego rozwiązania oraz czasu, w jakim zostało ono znalezione.

1.4. Program ma generować następujące wykresy:

- 1.4.1. Wartość funkcji celu od kolejnej iteracji.
- 1.4.2. Wartość odchylenia standardowego od kolejnej iteracji.
- 1.4.3. Średnia wartość funkcji od kolejnej iteracji.

1.5. Wyniki kolejnych iteracji powinny być zapisywane do repozytorium zewnętrznego.

4. **Architektura:**

Bazą algorytmu jest klasa Optimizer, jako parametry konstruktora przyjmuje:

- Funkcję poddawaną optymalizacji (adres definicji funkcji)
- Liczbę zmiennych niezależnych funkcji 9 (integer).
- Dziedziny zmiennych niezależnych (krotka)
- Precyzję rozwiązania (integer)
- Typ optymalizacji (integer/string ze zbioru {‘min’,’max}
- Oczekiwaną dokładność (float)
- Wartość definiująca czy wygenerowane mają zostać odpowiednie wykresy

Klasa Optimizer dziedziczy po klasach:

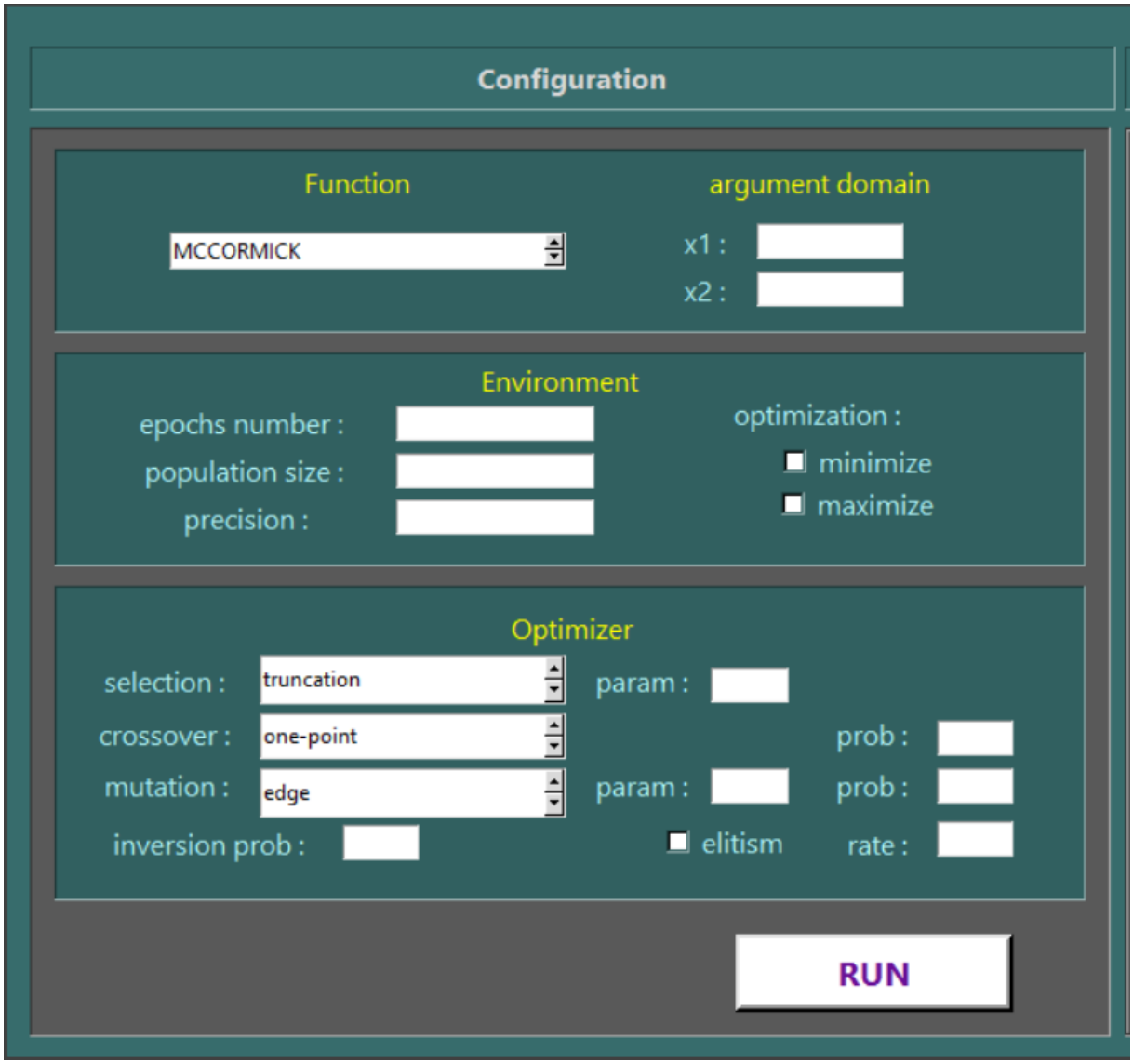
- Configuration – „statyczna” klasa przechowująca ustawienia konfiguracji algorytmu
- Selection - klasa implementująca metody selekcji
- Crossover – klasa implementująca metody krzyżowania
- Inversion – klasa implementująca inwersję
- Mutation – klasa implementująca metody mutacji

Najistotniejsze z metod klasy Optimizer.class:

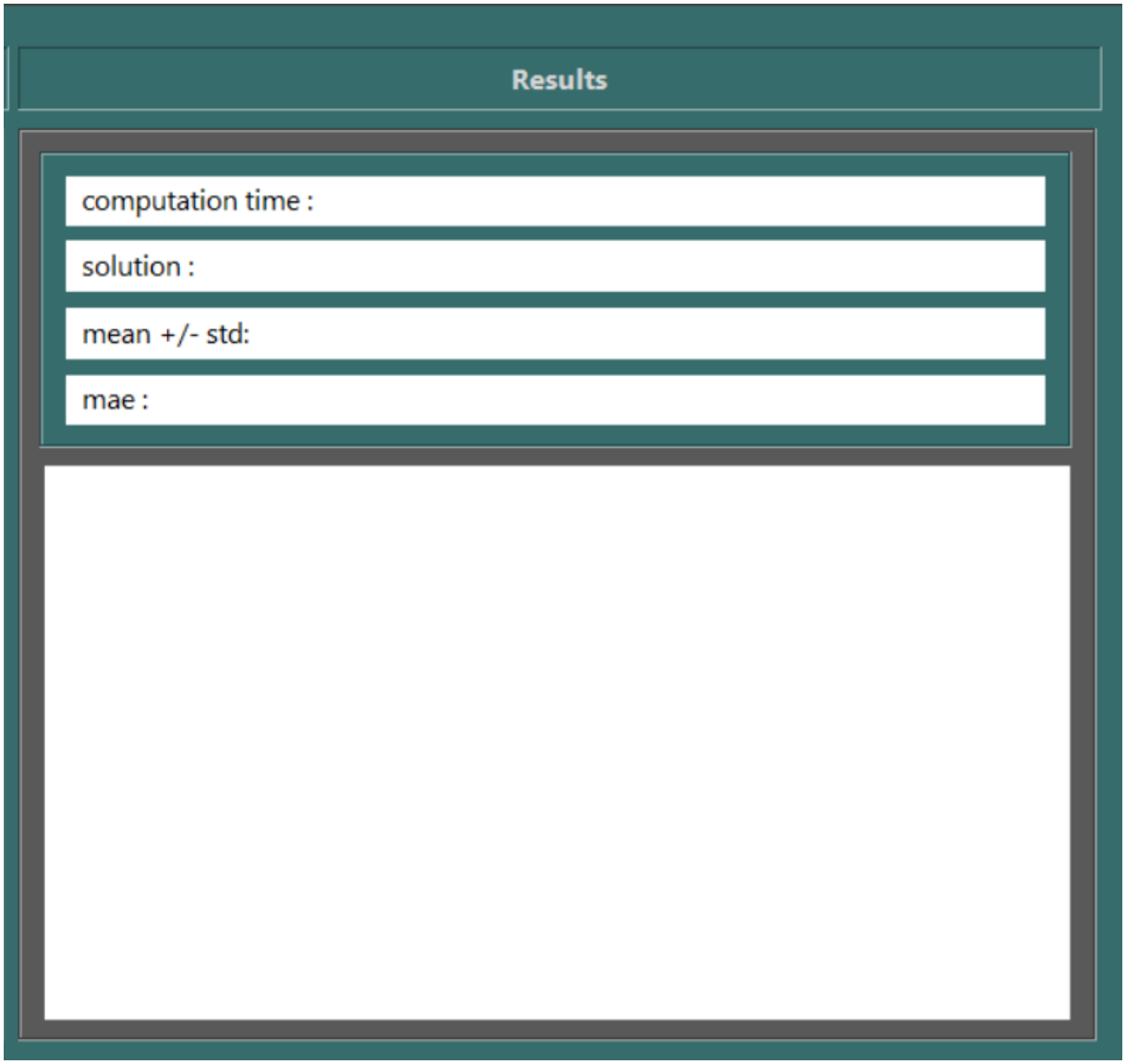
- initPopulation() – inicjacja populacji generacji „0”
- optimize(..) – metoda inicjująca algorytm genetyczny, argumenty :
  - selection – krotka z nazwą typy selekcji (string) oraz parametr pomocniczy selekcji (float/int)
  - crossover – krotka z nazwą typu krzyżowania (string) oraz wartość prawdopodobieństwa wystąpienia (float)
  - mutation – krotka z nazwą typu mutacji (string), liczba bitów poddanych mutacji (int), prawdopodobieństwo wystąpienia (float)
  - inversion_prob – prawdopodobieństwo wystąpienia (float)
  - elitism – liczba całkowita określająca czy ma zostać zastosowana strategia elitarna (1 – zastosuj strategię elitarną, N-{1} – nie stosuj strategii elitarnej) (int)
  - survival_rate – wycinek najlepszych osobników populacji przekazywanych bezpośrednio do kolejnej generacji
- run() – uruchamia algorytm, bezargumentowa metoda wywoływana w metodzie optimizer()
- nextGen() – pośrednio odpowiada za budowanie kolejnej generacji, bezargumentowa metoda wywoływana w metodzie run()
- reptoduction() – implementacja logiki odpowiadającej za tworzenie nowych osobników, selekcję „rodziców”, ich krzyżowanie, mutację oraz inwersję potomków

- evaluate() – bezargumentowa metoda służąca ocenie rozwiązań osobników populacji utworzonych w generacji danej iteracji, wyznacza najlepszego osobnika na podstawie wartości funkcji docelowej, tworzy metryki:
  - średniej wartości funkcji docelowej
  - wariancji wartości funkcji docelowej
  - średniego odchylenia standardowego wartości funkcji docelowej
  - średniego błędu bezwzględnego wartości funkcji docelowej (MAE)

5. **Graficzny interfejs użytkownika:**
   - **Konfiguracja**

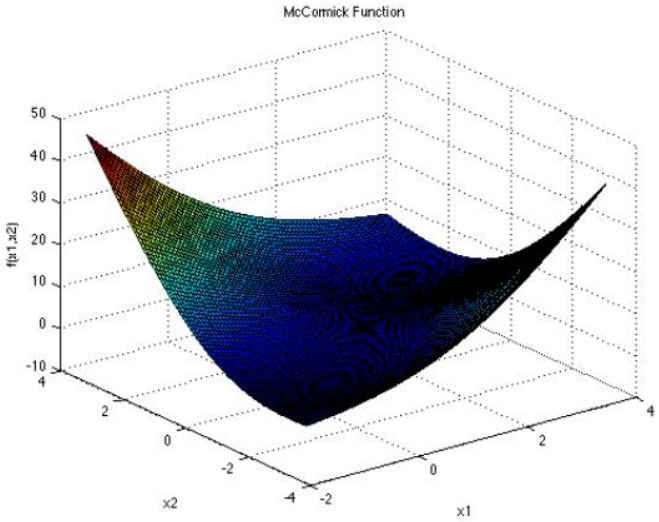


- **Prezentacja uzyskanych rezultatów**



6. **Przykładowy scenariusz uruchomieniowy:**
   Wybrana funkcja:

15. MCCORMICK FUNCTION



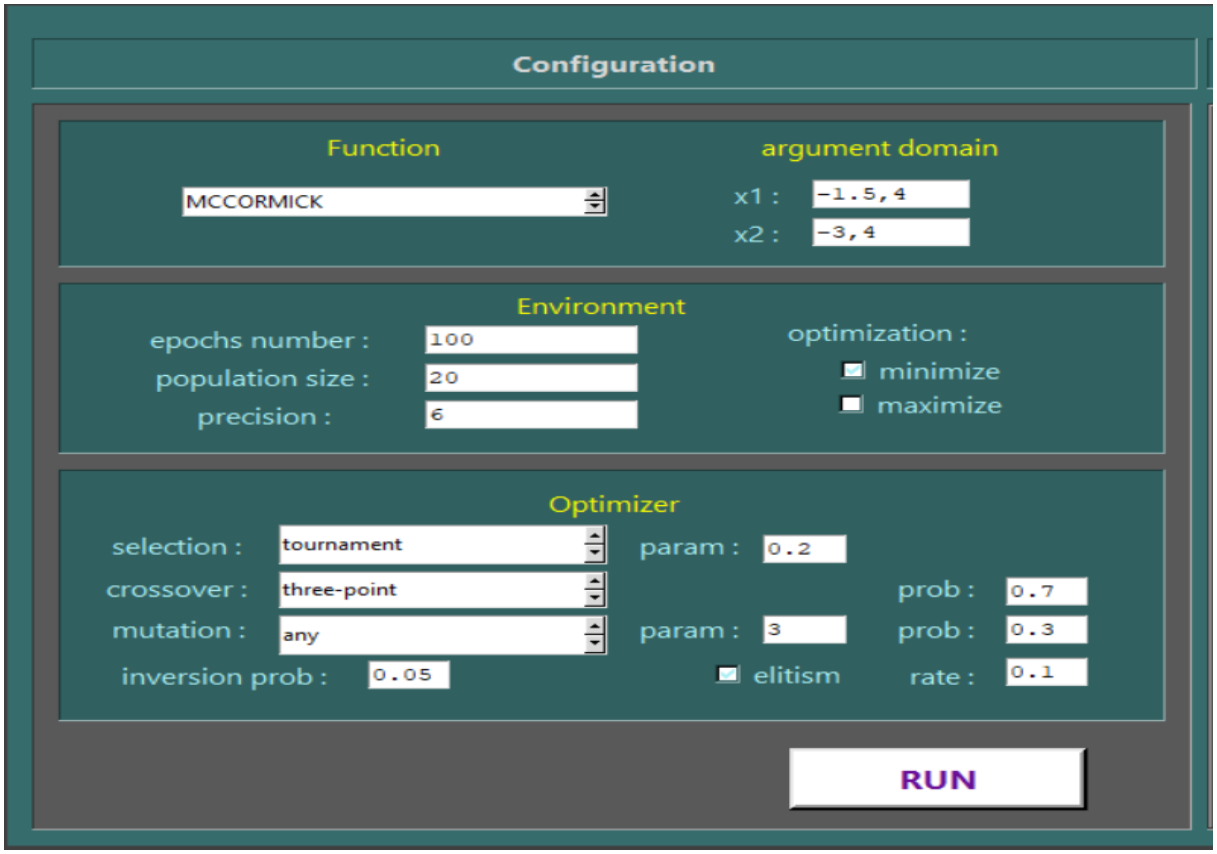
$$f(\mathbf{x}) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1$$

Description: *Dimensions: 2*

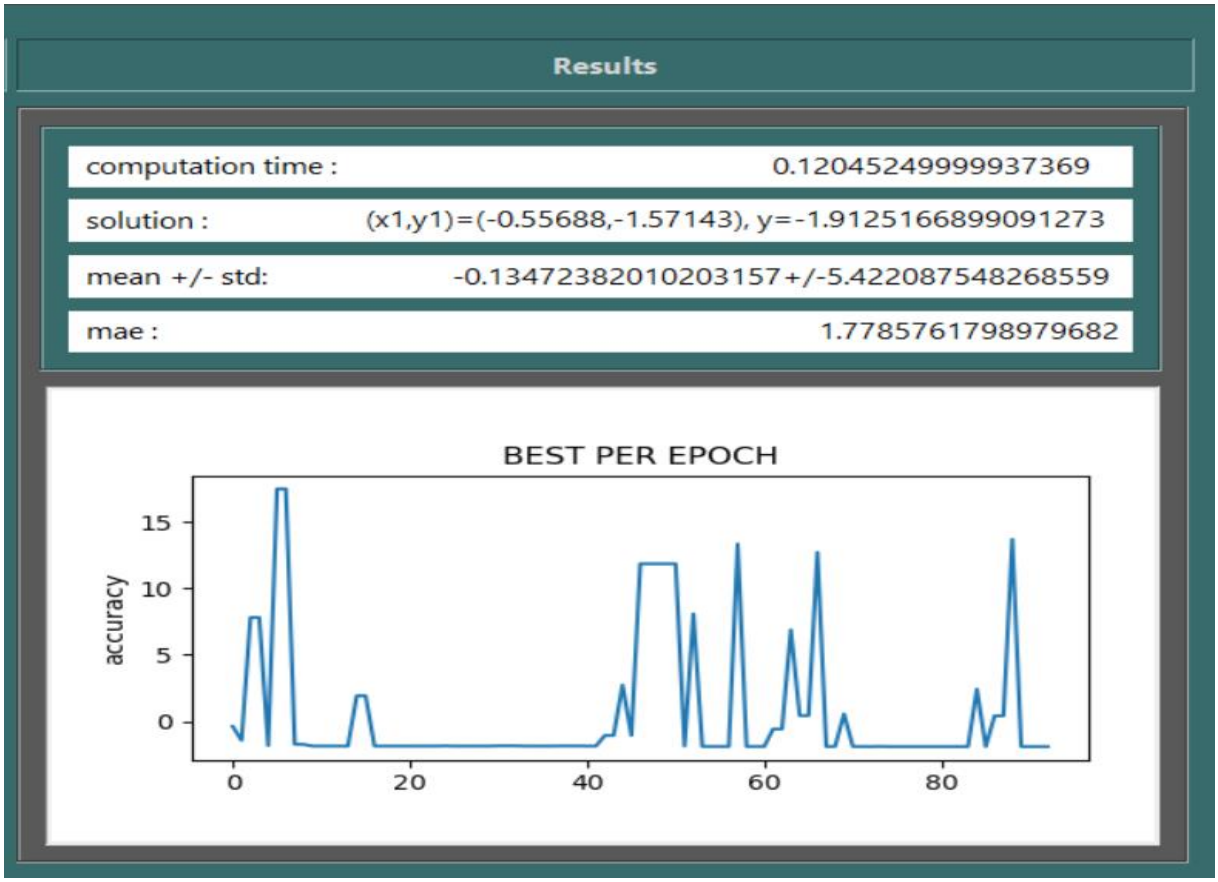
Input Domain: The function is usually evaluated on the rectangle $x_1 \in [-1.5, 4]$, $x_2 \in [-3, 4]$.

Global Minimum: $f(\mathbf{x}^*) = -1.9133$, at $\mathbf{x}^* = (-0.54719, -1.54719)$

- **Konfiguracja:**

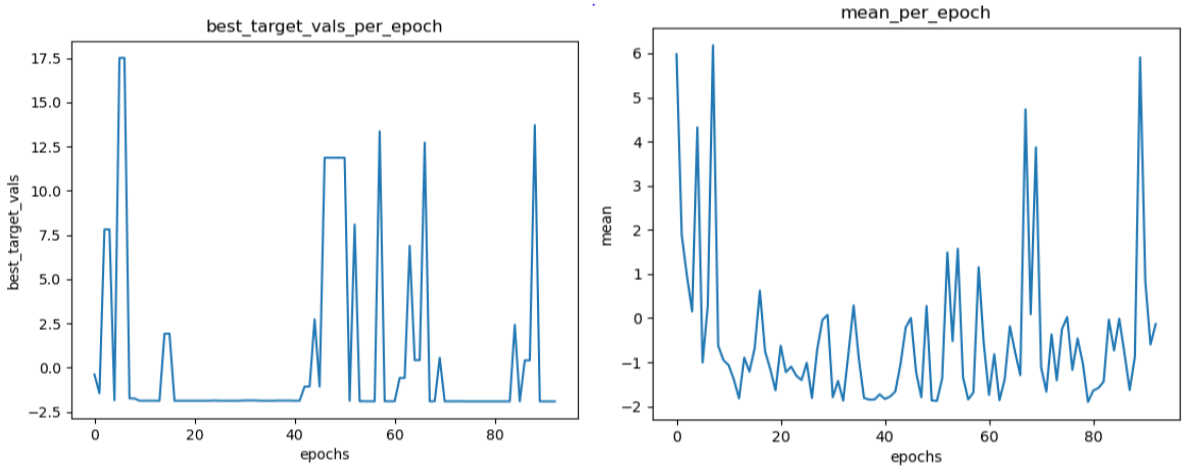


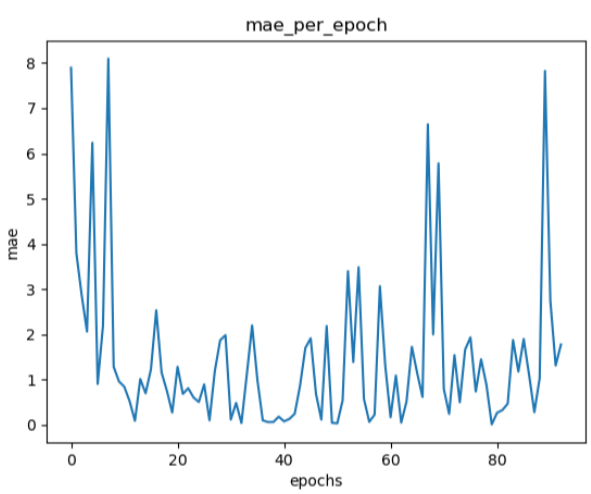
- **Uzyskany wynik:**

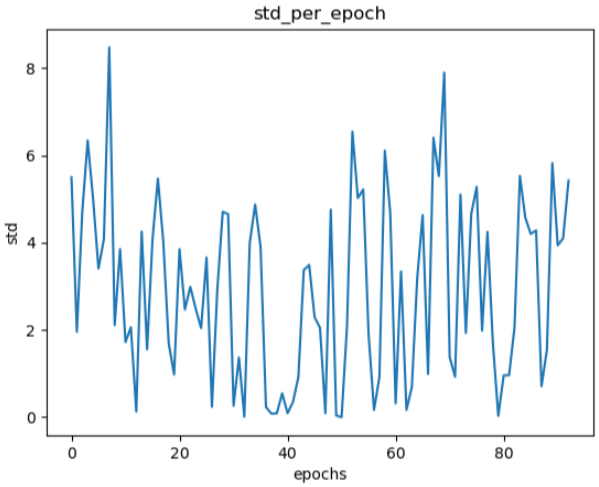


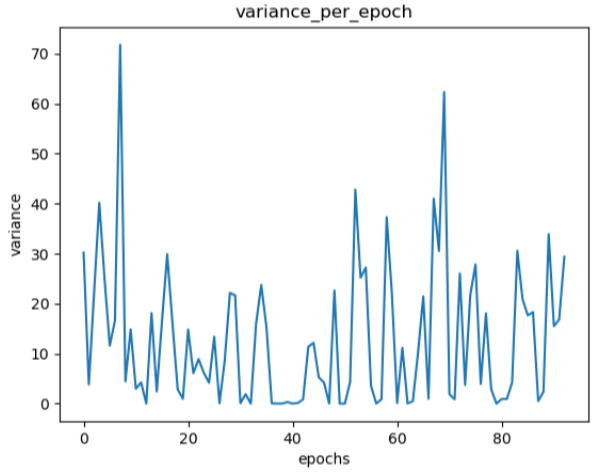
***czas mierzony jest w sekundach**

7. Wykresy uzyskane dla powyższej konfiguracj:





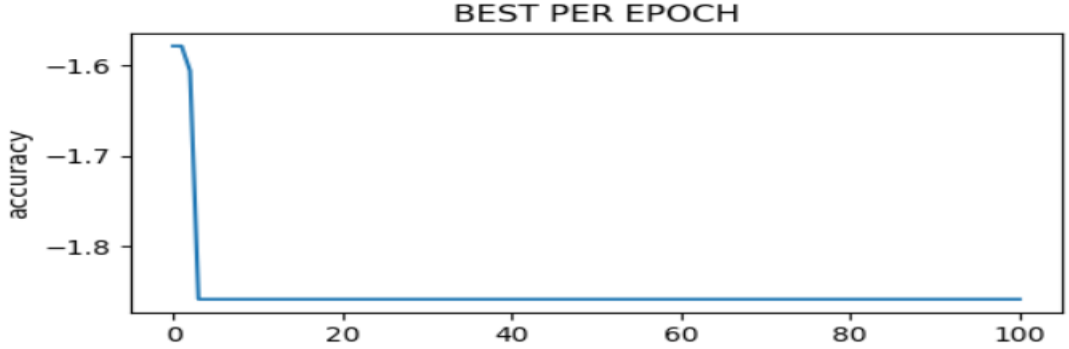




8. **Analiza otrzymanych wyników:**
   - Algorytm znalazł najlepsze rozwiązanie w 91 epoce, w związku z osiągnięciem satysfakcjonującej dokładności (do 3ech miejsc po przecinku) jego działanie zostało przerwane. Warto zwrócić uwagę, że stosunkowo bliskie rzeczywistemu minimum wyniku znajdowane są już na początku jego działania epoka nr 5 : -1.84, epoka nr 10 : -1.87, epoka nr 11: -1,87, epoka nr 12 = -1.87.

```
log : starting new epoch no. counter  91
log : process terminated due to expected accuracy being achieved in epoch :  91
log : plotting results..
```

   - Nagłe, wręcz biegunowe zmiany wartości znajdowanych rozwiązań prawdopodobnie są mutacji z prawdopodobieństwem ustalonym na 30% (oraz inwersji z 5% szans na wystąpienie) – zabiegi te pozwalają na utrzymanie różnorodności w populacji, przykład z wytłumioną mutacją oraz inwersją :



- Jednak kluczowe wydaje się być samo krzyżowanie, przy odpowiednio dużym prawdopodobieństwie (0.5-0.8) oraz mutacji na poziomie (0.2-0.3) rozwiązania o satysfakcjonującej dokładności (0.001) krzywa najlepszych rozwiązań per populacja wyraźnie wskazuje na stosunkowo szybkie skierowanie optymalizatora w odpowiednim kierunku osi zmiennych niezależnych.
- Ciekawym spostrzeżeniem wydaje się fakt, iż bez zastosowania strategii elitarnej rozwiązanie znajdowane jest najczęściej między 20, a 60 epoką, natomiast ze strategią elitarną nawet w ciągu pierwszych 5 epok.

9. Podsumowanie

Zadanie jakim jest optymalizacja funkcji przy wykorzystaniu algorytmu choć z pozoru wydaje się prostym problemem (intuicyjnie), ideowo jak i jeśli chodzi o realizację stanowiła spore wyzwanie, nie mniej jednak poznanie technik algorytmów genetycznych czerpiących przecież bezpośrednio ze świata biologii okazało się szalenie zajmujące oraz fascynujące na swój sposób.