Algorytmy ewolucyjne
Projekt 1
Sprawozdanie

Michał Ziober – 311612

1 Treść zadania

Napisać program umożliwiający znalezienie maksimum funkcji dopasowania jednej zmiennej określonej dla liczb całkowitych w zadanym zakresie przy pomocy elementarnego algorytmu genetycznego (reprodukcja z użyciem nieproporcjonalnej ruletki, krzyżowanie proste, mutacja równomierna). Program powinien umożliwiać użycie różnych funkcji dopasowania, populacji o różnej liczebności oraz różnych parametrów operacji genetycznych (krzyżowania i mutacji). Program powinien zapewnić wizualizację wyników w postaci wykresów średniego, maksymalnego i minimalnego przystosowania dla kolejnych populacji oraz wykresu funkcji w zadanym przedziale. Program przetestować dla funkcji f(x)= -0.4x2+4x+6 dla x= -1, 0, ... 21

2 Technologia

Projekt został wykonany w języku programowania Python w wersji 3. Użyte środowisko programistyczne to Visual Studio.

3 Instrukcja obsługi

Przede wszystkim projekt należy pobrać i wypakować. Następnie należy przejść do folderu projektu i uruchomić poleceniem python3 main.py. Projekt można także przetestować uruchamiając go poprzez środowisko Visual Studio. Konfiguracja parametrów odbywa się w pliku settings.json. Dostępna konfiguracja parametrów to:

- fitness_function - funkcja na jakiej bazuje algorytm
- population_size - rozmiar populacji
- generations - ilość generacji algorytmu
- mutation_probability - prawdopodobieństwo mutacji danego osobnika
- crossover_probability - prawdopodobieństwo skrzyżowania się danych osobników
- min_x - początek przedziału funkcji
- max_x - koniec przedziału funkcji

4 Opis wykonanych symulacji

Symulacja opiera się na podstawowym działaniu algorytmu genetycznego. W moim programie działa to następująco:

1. Tworzona jest początkowa populacja. Ilość osobników jest równa parametrowi "population_size". Od razu wyciągana jest minimalna, maksymalna i średnia wartość z populacji. Obliczana jest także wartość przystosowania dla poszczególnych osobników

2. Początkowa populacja jest dodawana do historii(zapisywanie minimalnej, maksymalnej oraz średniej wartości dla populacji)

3. Uruchamiana jest pętla, w której tworzona jest nowa populacja

   (a) Przy pomocy metody nieproporcjonalnej ruletki losowani są rodzice dla kolejnej populacji - największą szansę na wylosowanie mają osobniki z największa wartością funkcji przystosowania(przykładowo dla x = 2 wartość funkcji przystosowania to 12,4, natomiast dla x = 10 będzie to 6. Pierwszy osobnik ma więc większą szansę na bycie wylosowanym, bo ma większą wartość przystosowania)

   (b) Następuje następuje krzyżowanie proste pomiędzy wylosowanymi rodzicami. Po pierwsze, brana jest dwójka rodziców. Następnie losowany jest punkt krzyżowania i tworzona jest dwójka dzieci. To, czy trafią one do następnej generacji, zależy od parametru crossover_probability - losowana jest liczba z przedziału <0;1>. Jeśli mieści się ona w przedziale <0;crossover_probability>, to dzieci są dodawane do nowej generacji. Natomiast jeśli wylosowana liczba jest większa od crossover_probability, to do następnej generacji zostają dodani niezmienieni rodzice. Przykład: rodzice to 1010(10) oraz 1101(13). Wylosowany punkt krzyżowania to 2, czyli środek. Powstaną dzieci: 1001(9) oraz 1110(14)

   (c) Następuje mutacja. Pętla przechodzi po każdym osobniku w populacji(rodziców). To, czy osobnik ulegnie mutacji zależy od parametru mutation_probability. Losowana jest liczba z przedziału <0;1>. Jeśli mieści się ona w przedziale <0;mutation_probability>, to osobnik ulegnie mutacji. Jeśli liczba jest wyższa, to osobnik pozostanie niezmieniony. Losowany jest punkt mutacji(program operuje na binarnej reprezentacji). Wylosowane miejsce jest mutowane i zmieniana jest jego wartość - jeśli wcześniej było 0, to ustawiane będzie 1 i odwrotnie. Przykład: mutation_probability wynosi 0.001. Rozpatrywanym osobnikiem jest 1101(13). Wylosowano liczbę 0.0004, czyli osobnik zmutuje. Wylosowano punkt mutacji 2(iterujemy od zera), czyli zmieniona zostanie 3 pozycja - 0 zamieni się na 1. Zmutowany osobnik to 1111(15).

   (d) Po wykonaniu krzyżowania oraz mutacji zwracana jest nowa generacja, która dodawana jest do historii. Cykl się powtarza aż do osiągnięcia zadanej liczby generacji

   (e) Po przejściu przez wszystkie generacje, tworzona jest graficzna wizualizacja programu, która jest wyświetlana użytkownikowi, ale także jest zapisywana w folderze programu pod nazwą "output.png"

Natrafiono na kilka drobnych problemów podczas implementacji. Zostaną one objaśnione i przedstawione rozwiązania jakie zostały użyte.

- Podczas mutacji lub krzyżowania może powstać liczba spoza zakresu - mniejsza niż -1, lub większa niż 21, na przykład mutacja liczby 10100(20) po mutacji da 11100(28), czyli liczba poza zakresem. W tym przypadku mutacja jest po prostu powtarzana do skutku - aż powstały osobnik będzie się mieścić w zakresie, czyli na przykład powstanie liczba 10000(16). Przy krzyżowaniu jest podobnie, na przykład rodzice to 00111(7) i 10101(21). Po krzyżowaniu mogłyby wyjść przykładowe dzieci: 00101(5) oraz 10111(23). 23 jest większe od 21, więc po prostu dla tych samych rodziców krzyżowanie jest przeprowadzane do skutku, aż dzieci będą się mieścić w przedziale.

- Kolejną kwestią było obliczanie przystosowania. W zadanym przedziale funkcja osiąga również wartości mniejsze od 0. przystosowanie obliczane jest jako przystosowanie_pojedynczego_osobnika/suma_przystosowań_wszystkich_osobników. Problem jaki tu następuje jest w przypadku obliczania przystosowania dla ujemnych liczb - dawałoby ujemny procent szans na ruletce. Problemem jest także, gdyby wartość funkcji dla danego x wyniosłaby 0 - wtedy na ruletce dany rodzic miałby 0% szans na bycie wylosowanym. Zostało to rozwiązane następująco: najpierw następuje przejście po wszystkich osobnikach i szukana jest wartość minimalna. Po znalezieniu tej wartości, jest brana z niej wartość bezwzględna. Przy obliczaniu wartości przystosowania ta liczba jest dodawana do przystosowania każdego osobnika. Dodawana jest także jedynka do każdego przystosowania, aby żaden osobnik nie miał wartości "0". Operacje te powodują to, że żadne przystosowanie nie jest mniejsze lub równe 0 przy zachowaniu odpowiednich proporcji między osobnikami.

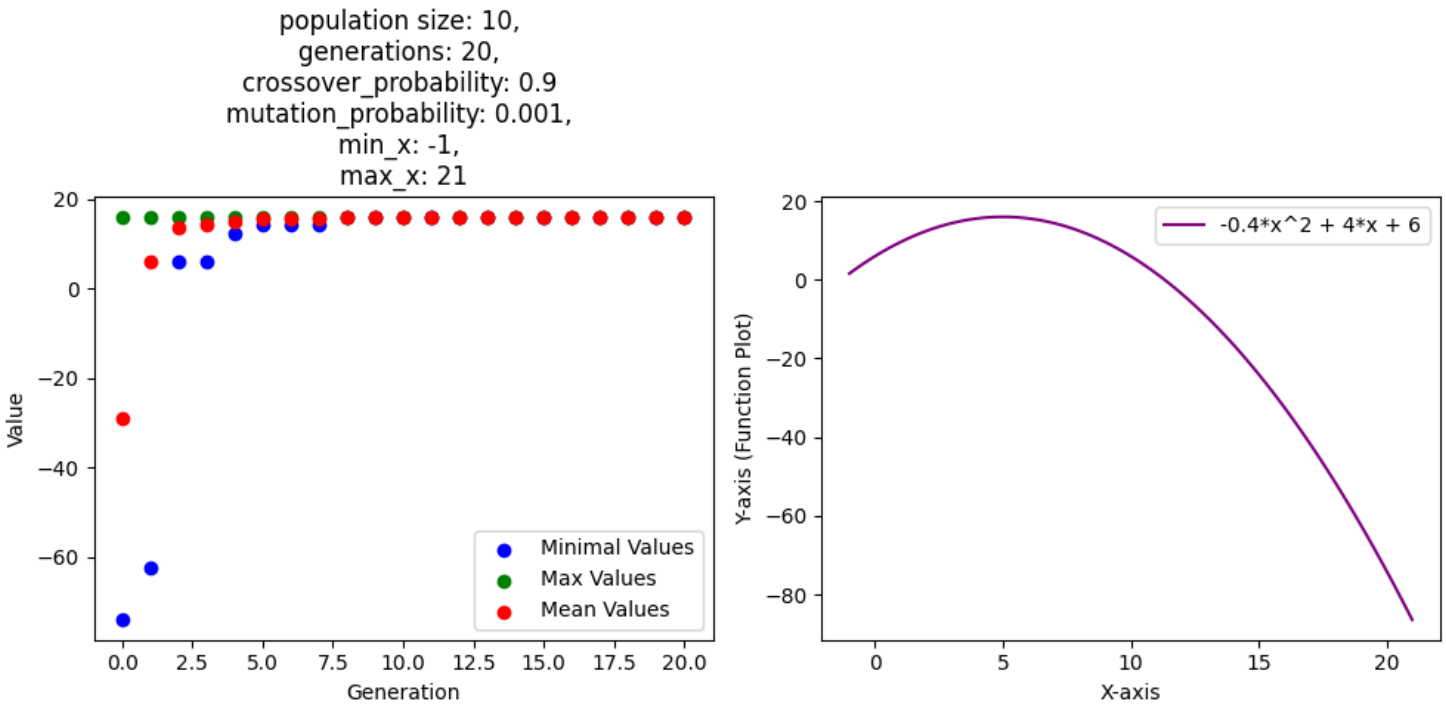
5 Testy dla różnych parametrów wejściowych

Maksymalna wartość w zadanym przedziale to 16 i odpowiada to x = 5.
Podstawowe parametry ustawiane w pliku settings.json, to:

```
{
  "fitness_function": "-0.4*x**2 + 4*x + 6",
  "population_size": 10,
  "generations": 20,
  "mutation_probability": 0.001,
  "crossover_probability": 0.9,
  "min_x": -1,
  "max_x": 21
}
```

Rysunek 1: Początkowe parametry

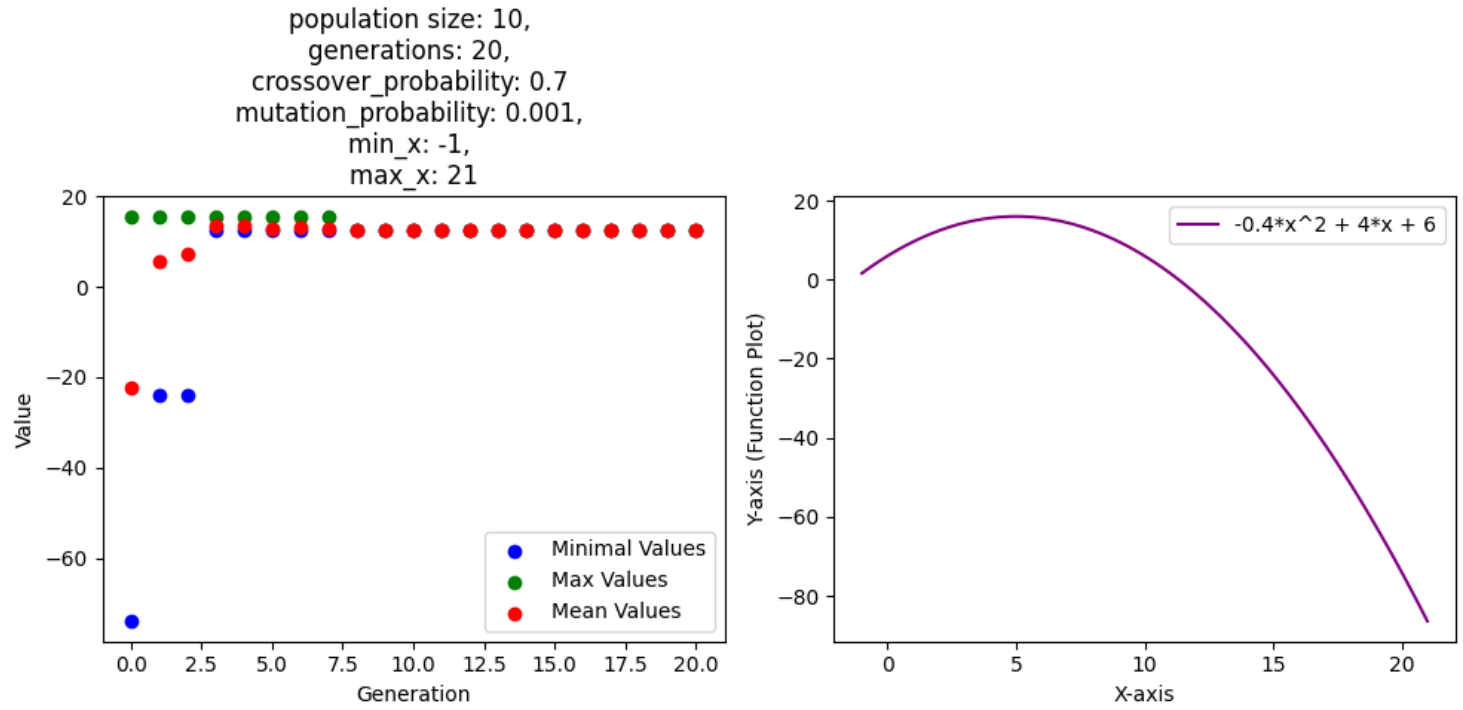
Graficzny efekt dla zadanych parametrów wygląda następująco:



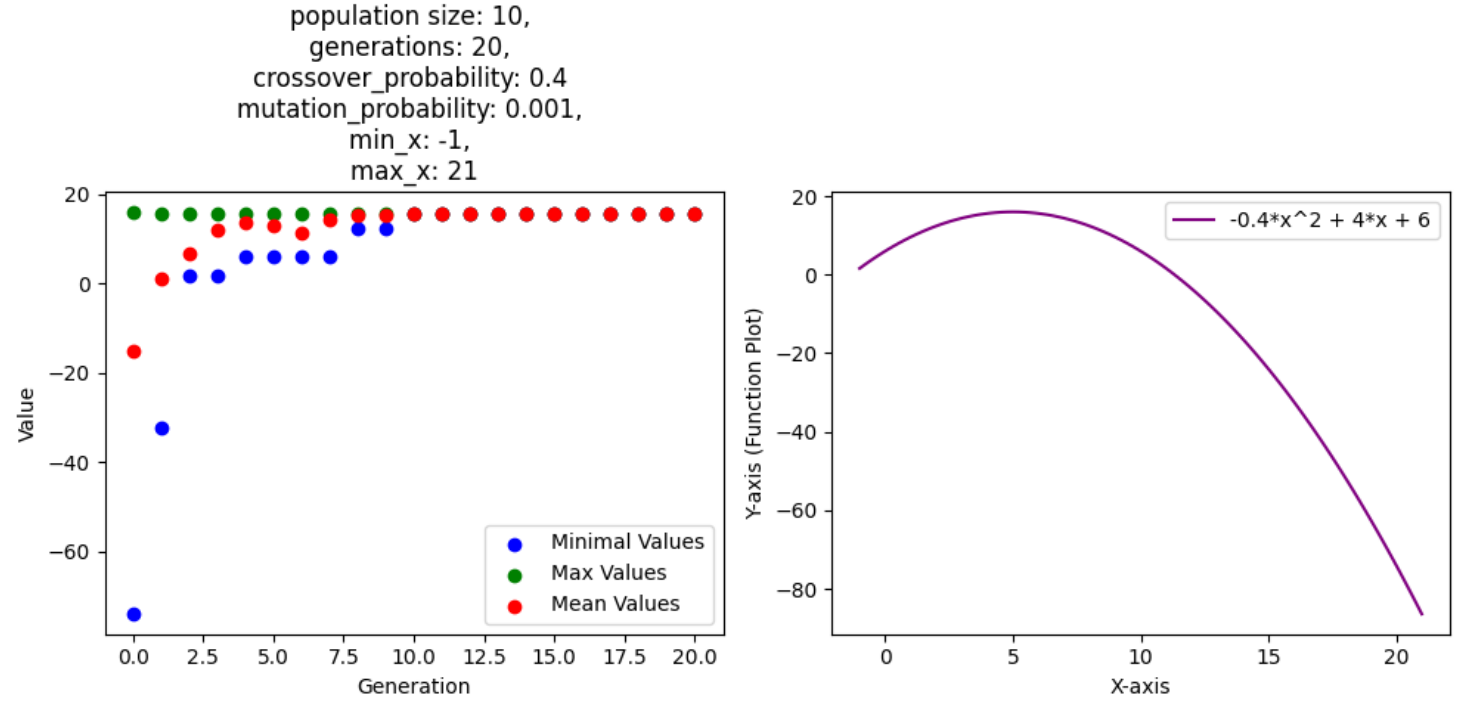
Rysunek 2: Graficzny efekt dla początkowych parametrów

Jak widać, dla zadanych parametrów algorytm już przy 8 generacji osiąga swoje maksimum - wszystkie osobniki stały się najlepszym rozwiązaniem(średnia, minimum oraz maksimum są sobie równe)
Przedstawię teraz efekt zmieniania różnych parametrów.

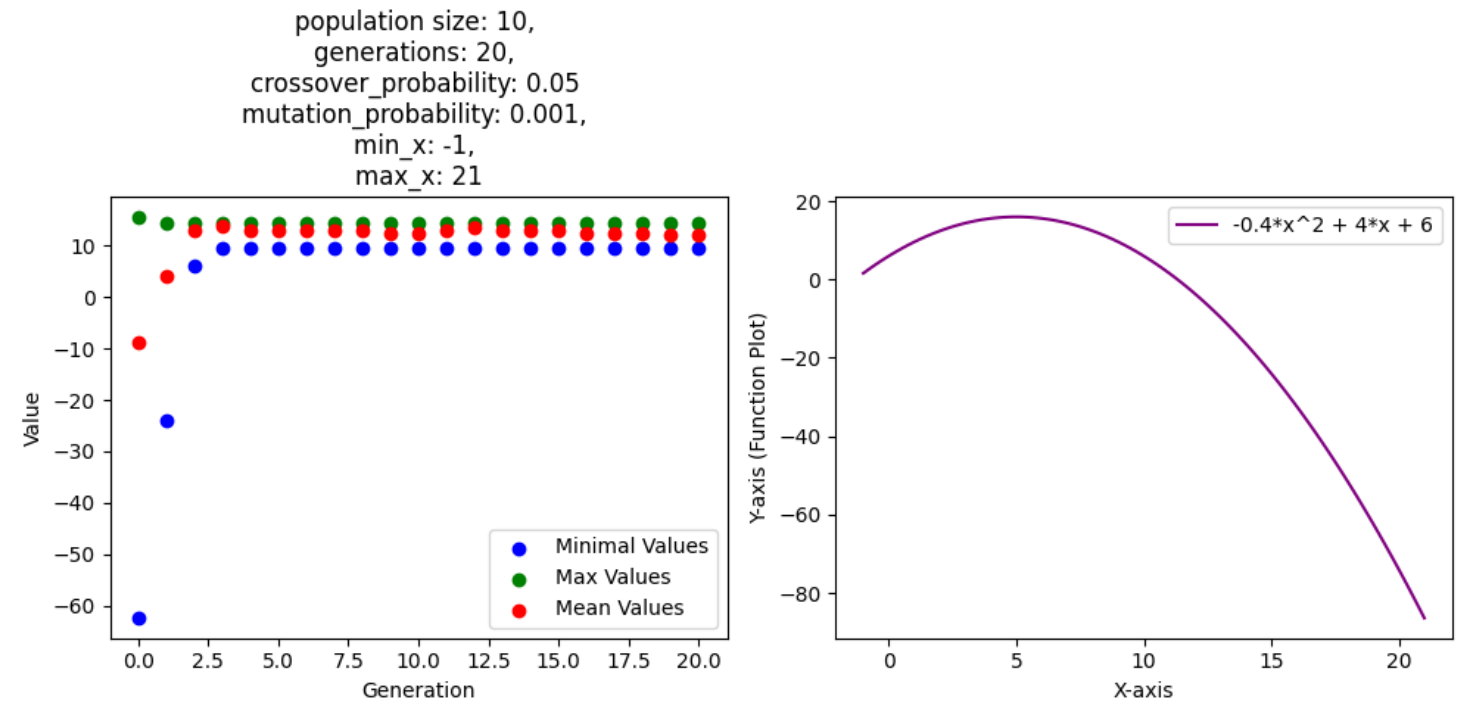
5.1 3 różne prawdopodobieństwa krzyżowania



Rysunek 3: Prawdopodobieństwo krzyżowanie 0.7

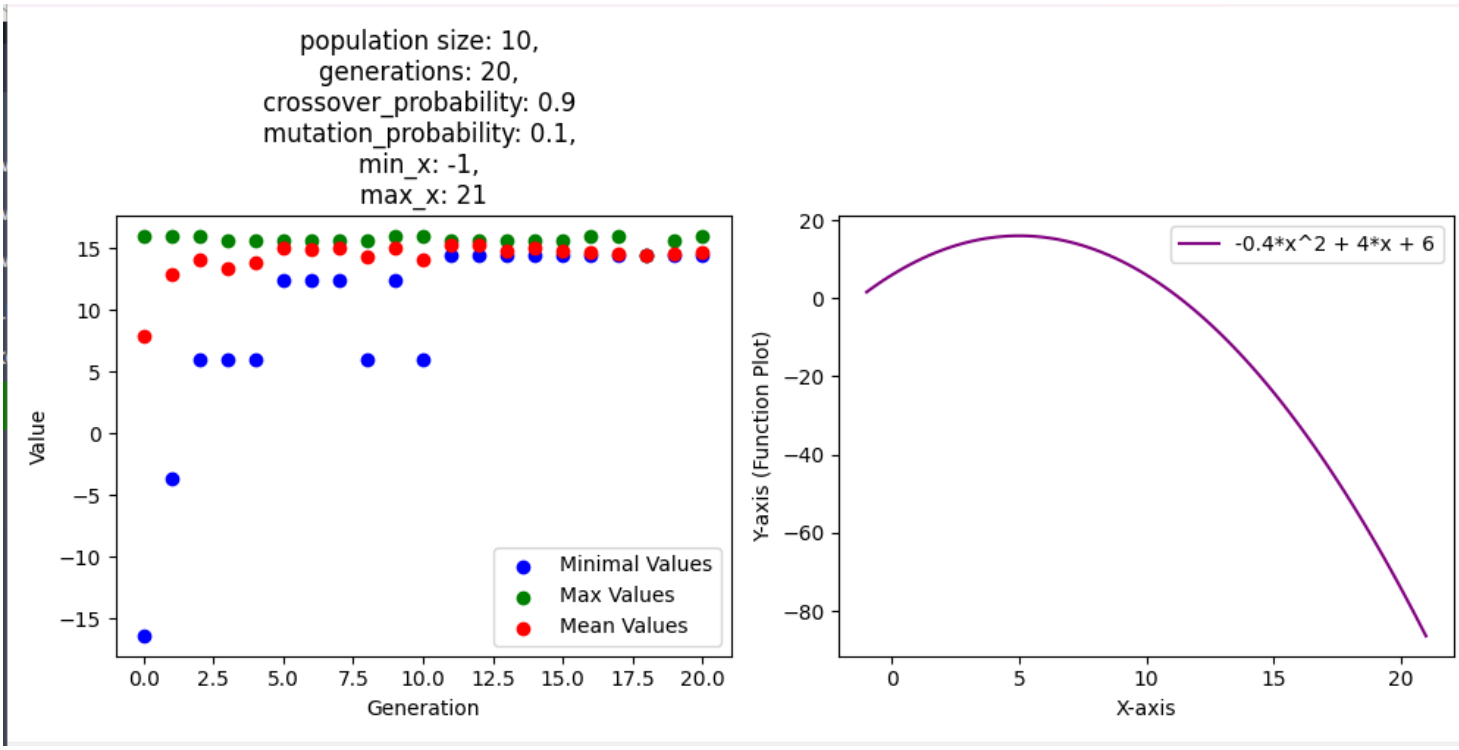


Rysunek 4: Prawdopodobieństwo krzyżowanie 0.4

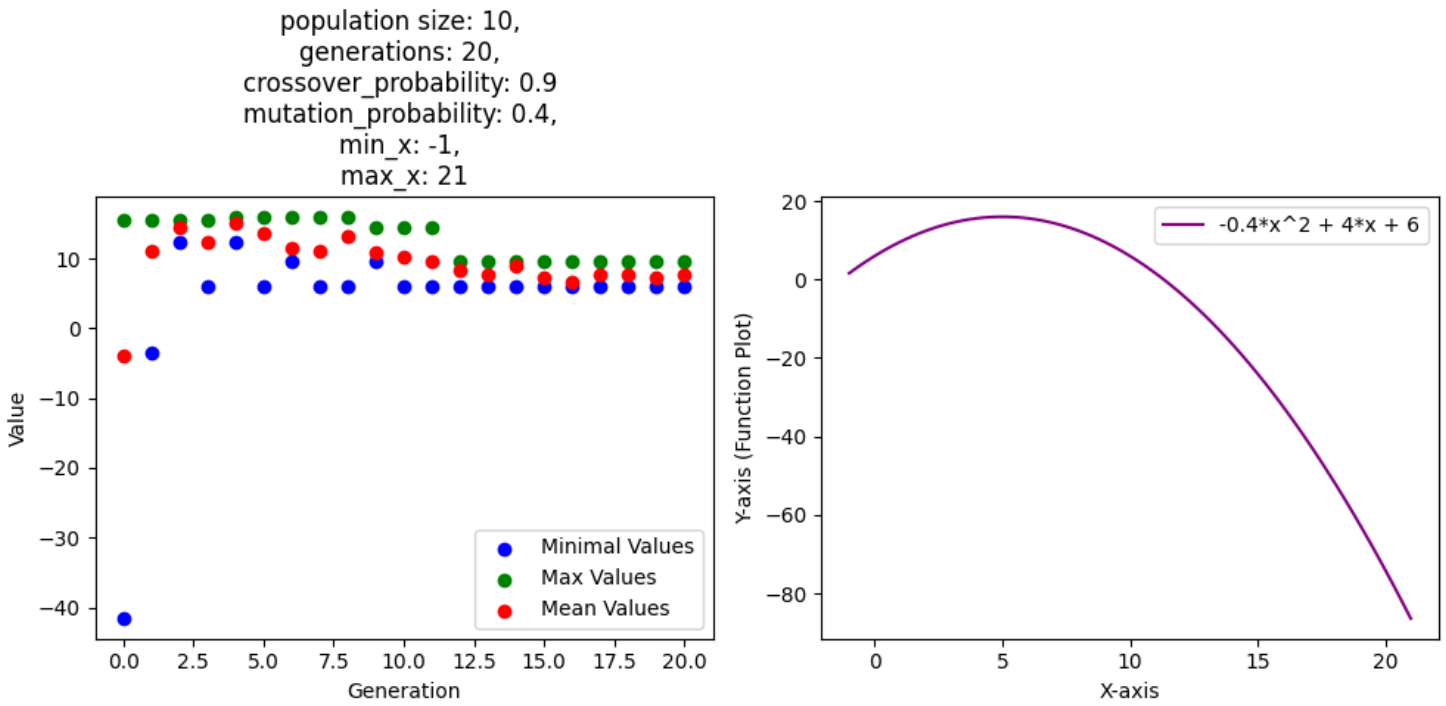


Rysunek 5: Prawdopodobieństwo krzyżowanie 0.05

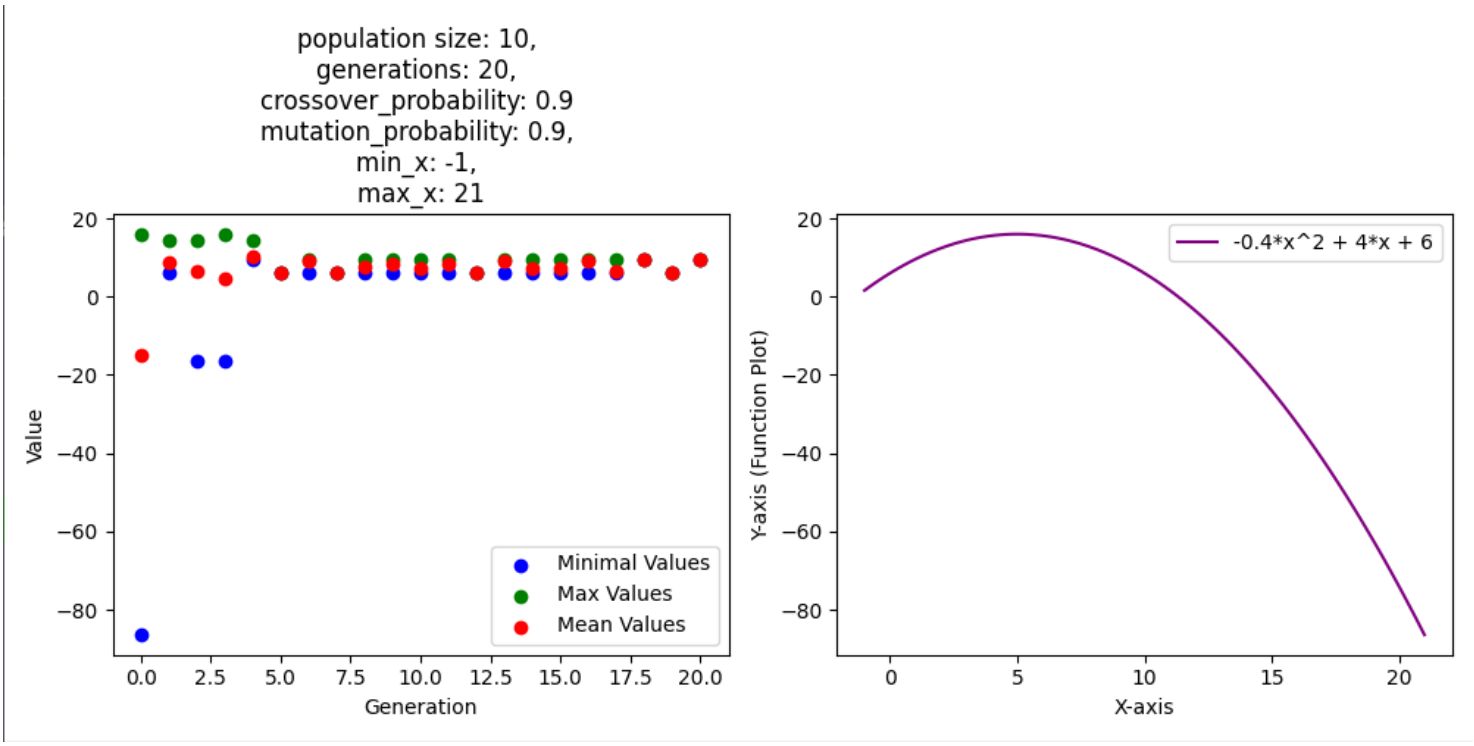
5.2 3 różne prawdopodobieństwa mutacji



Rysunek 6: Prawdopodobieństwo mutacji 0.1

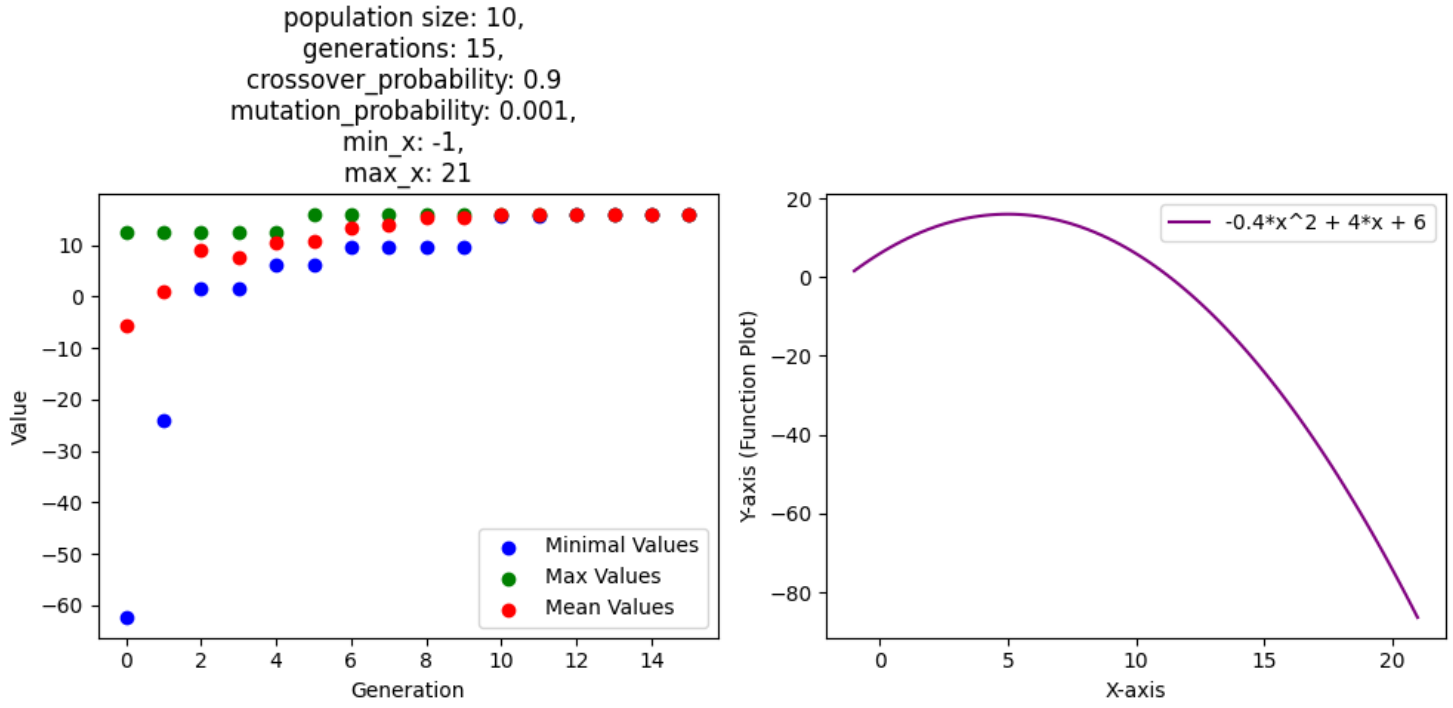


Rysunek 7: Prawdopodobieństwo mutacji 0.4

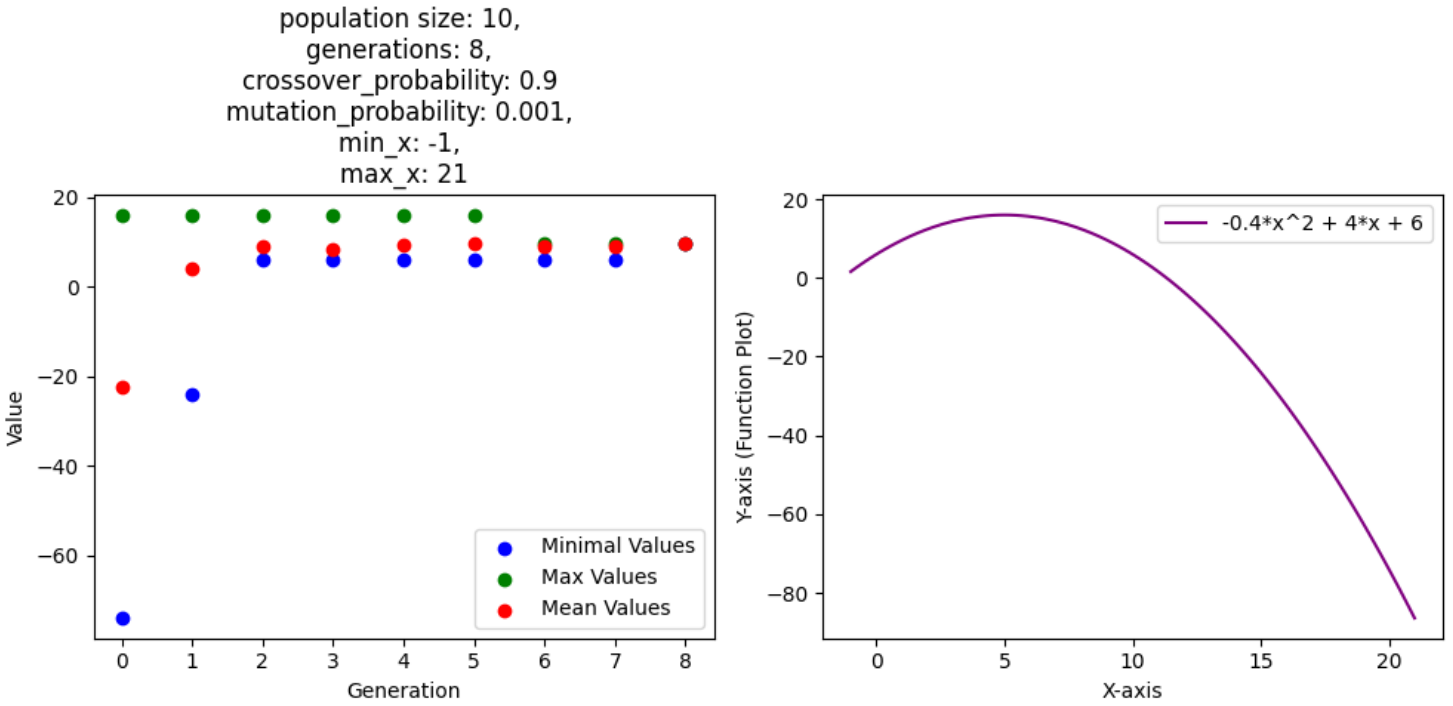


Rysunek 8: Prawdopodobieństwo mutacji 0.9

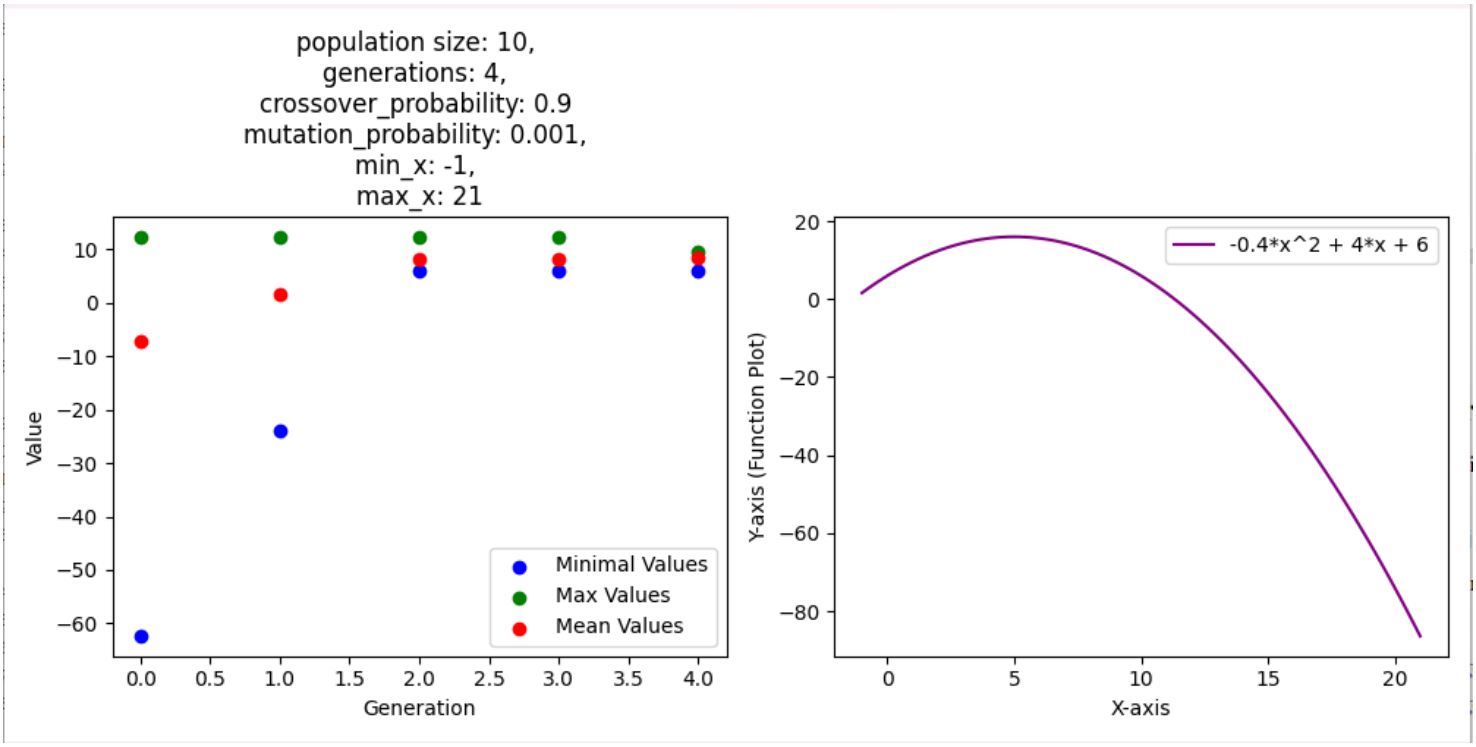
5.3 3 różne wartości maksymalnej liczby pokoleń



Rysunek 9: Ilość pokoleń: 15



Rysunek 10: Ilość pokoleń: 8



Rysunek 11: Ilość pokoleń: 4

6 Wnioski zmiany parametrów

- Prawdopodobieństwo krzyżowania powinno być dosyć wysokie - około 90%, aby jednoznacznie można było określić wartość najlepszego "x". W moich przykładach algorytm potrzebował średnio więcej pokoleń, aby jednoznacznie określić najlepsze przystosowanie, a dla skrajnie niskiej wartości tego współczynnika(0.05) algorytm podał błędny wynik.
- Prawdopodobieństwo mutacji powinno być stosunkowo niskie - w projekcie przyjęte zostało jako 0.001. Takie prawdopodobieństwo pozwala "uciec"algorytmowi z błędnego rozwiązania, a przy okazji nie powoduje to losowości rozwiązania. Przy skrajnie wysokiej wartości tego współczynnika rozwiązanie algorytmu jest dosyć losowe. Nie ma tutaj zgodności rozwiązania(nie można jednoznacznie określić do jakiej wartości algorytm dąży)
- Ilość populacji - Zmiana tej wartości jest uzależniona od złożoności danego zadania oraz obserwacji zachowania algorytmu. Zadany przykład jest dosyć prostym problemem i wystarczające jest tutaj około 8-10 generacji aby poznać rozwiązanie(zbieganie się wykresów minimalnej, maksymalnej oraz średniej wartości)

7 Struktura programu

Program jest podzielony na kilka plików, poniżej przedstawiony zostanie opis każdego z nich wraz z potencjalnymi szczegółami:

- AE_proj1.sln - plik projektu, stworzony przez środowisko Visual Studio
- Entity.py - klasa reprezentująca danego osobnika

  Parametry:

  - x - wartość x danego osobnika
  - y - wartość y danego osobnika(uwzględniając zadaną funkcję)

  - binary - wartość x przekształcona na binarny odpowiednik. Czyli na przykład x = 7, to binary = 111
  - percent_score - wartość przystosowania wyrażona w procentach całości

- Generation.py - klasa reprezentująca konkretną generację, czyli zbiór osobników

  Parametry:

  - population - populacja, czyli zbiór obiektów typu Entity
  - minimal_value - minimalna wartość "y"spośród wszystkich osobników danej generacji
  - max_value - maksymalna wartość "y"spośród wszystkich osobników danej generacji
  - mean_value - średnia wartość "y"wszystkich osobników

  Metody:

  - initialize_population - metoda inicjalizująca początkową populację
  - calculate_fitness - oblicza przystosowanie dla każdego osobnika w populacji
  - select_parents - wybór rodziców z aktualnej generacji dla kolejnej generacji
  - crossover - metoda odpowiedzialna za krzyżowanie w danej generacji
  - mutate - metoda odpiwadająca za mutację w danej generacji
  - next_generation - metoda tworząca kolejną generację bazując na aktualnej. Wykorzystuje wcześniejsze metody select_parents, crossover, mutate
  - get_minimal_value - zwraca najmniejsze "y"w danej popualcji
  - get_max_value - zwraca największe "y"w danej popualacji
  - get_mean_value - zwraca wartość średniego "y"z danej popualcji

- History.py - klasa odpowiadająca za zbieranie danych dotyczących generacji - wartości najmniejsze, największe i średnie

  Parametry

  - minimal_values[] - minimalne wartości z kolejnych generacji
  - max_values[] - maksymalne wartości z kolejnych generacji
  - mean_values[] - średnie wartości z kolejnych generacji

  Metody:

  - add_generation_to_history - dodanie generacji do historii

- settings.json - plik z konfiguracją paremetrów programu
- settings.py - plik odczytujący konfiurację z pliku settings.json
- output.png - plik graficzny, będący wynikiem działania programu. Jest w nim zawarty wykres funkcji oraz punkty odpowiadająca wartości maksymalnej, minimalnej oraz średniej z każdej kolejnej generacji. Dodatkowo są wypiswane parametry programu