Практическая работа №2

По курсу “**Генетические алгоритмы и нечёткая обработка данных**”

Исследование работы генетического алгоритма для поиска экстремумов

Работу выполнил студент группы ПМ-401

Машарипов Тимур Алишерович

Цель работы

Генетический алгоритм - это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе.

У самого алгоритма есть ряд его параметров (гиперпараметров). В данной работе предлагается исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от :

- числа особей в популяции
- вероятности кроссинговера
- вероятности мутации

Алгоритм будет находить максимум функции

$$f(x) = \ln(x) \cdot \cos(3x - 15)$$

на отрезке [1, 10]

Онлайн калькулятор WolframAlpha выдаёт следующий ответ:

max{log(x) cos(3 x - 15)}≈2.21828 at x≈9.19424

Это значение далее в исследовании будет рассматриваться, как истинное.

Технические особенности

Алгоритм реализован в пакете Matlab

Процессор: AMD Ryzen 7 1700 Eight-Core Processor 3.00 GHz

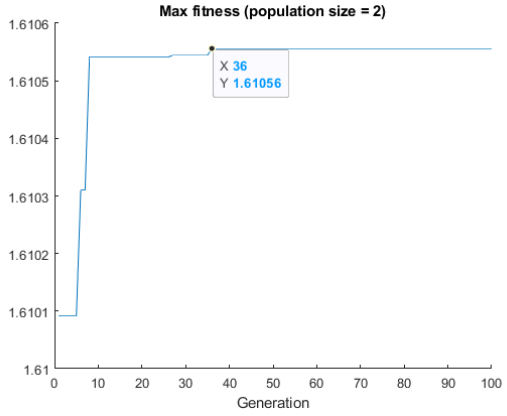
Видеокарта: Radeon RX 580 series

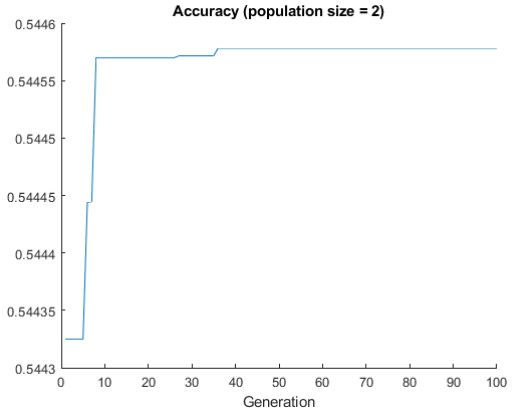
Зависимость от числа особей в популяции

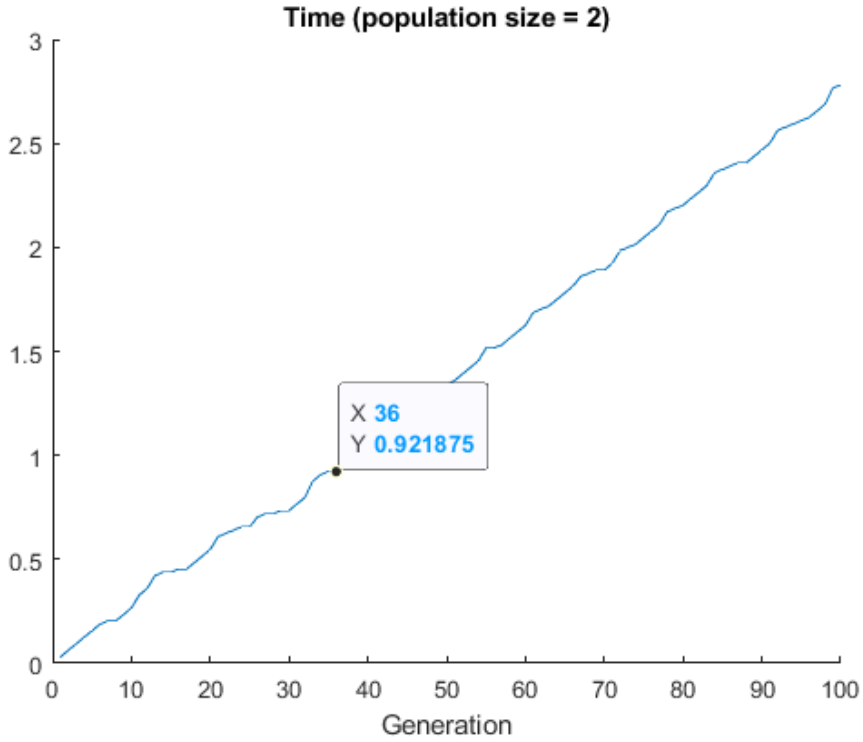
В теории генетических алгоритмов в каждом поколении создаётся популяция фиксированного размера. В этом разделе предлагается исследовать поведение алгоритма в зависимости от количества особей в популяции.

Пусть изначально populationSize=2

На выходе получаем следующие графики:





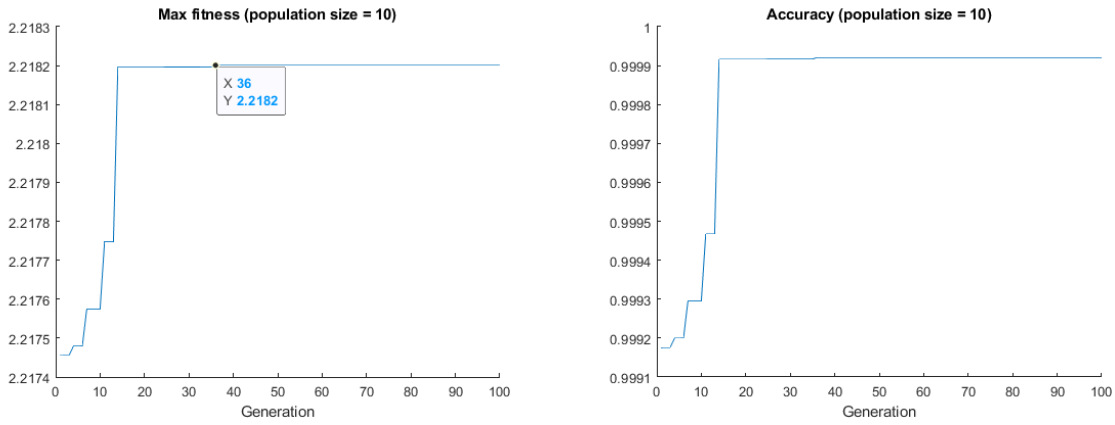


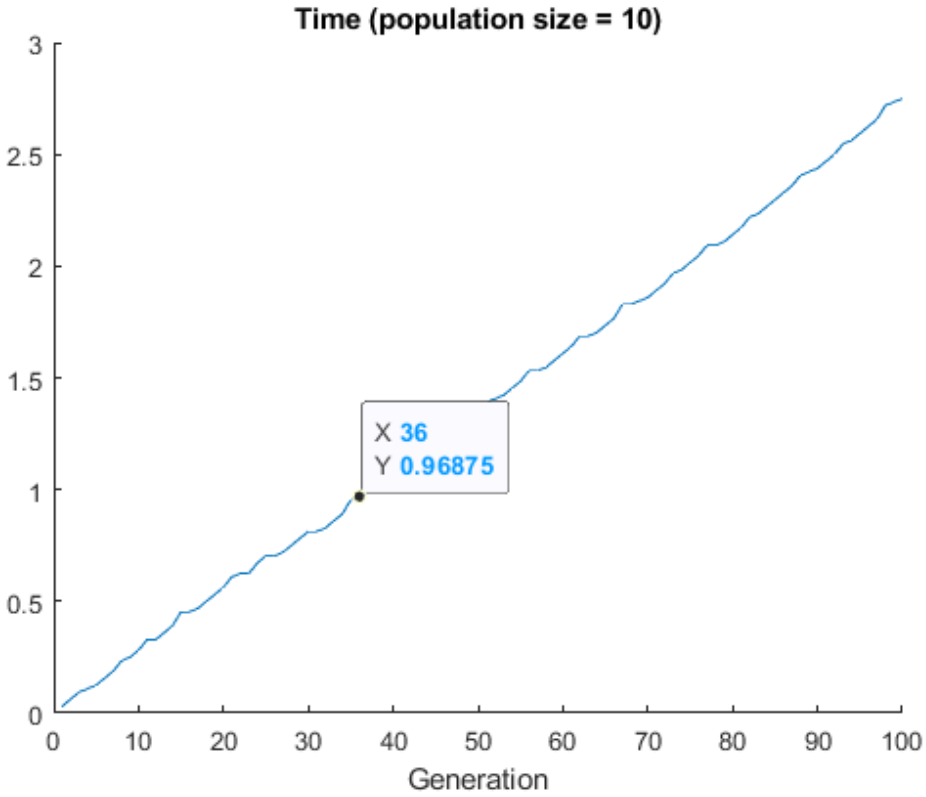
Левый верхний график отображает максимальное среди всей популяции значение функции приспособленности в каждом поколении,

правый верхний - точность функции приспособленности, рассчитываемую как exp(-ошибка), и, наконец, ***нижний*** график - время, прошедшее с запуска алгоритма (в секундах).

Как можно видеть, когда в популяции всего 2 особи, за 100 поколений так и не удаётся найти решения. На 36 итерации алгоритм зашёл в локальный оптимум и уже вряд ли оттуда выберется.

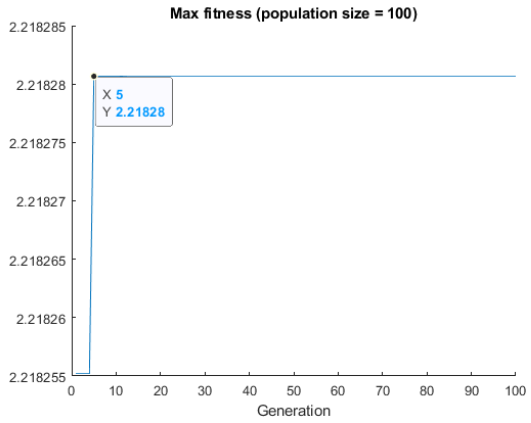
Рассмотрим populationSize=10

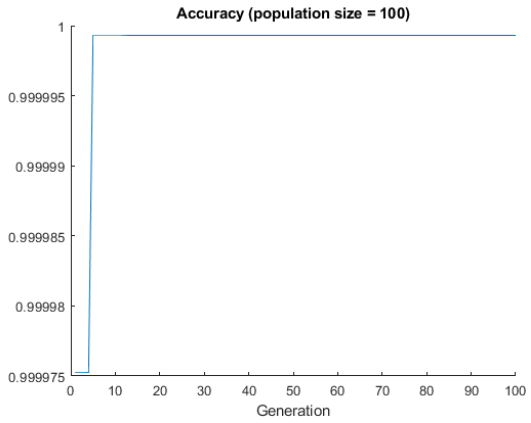


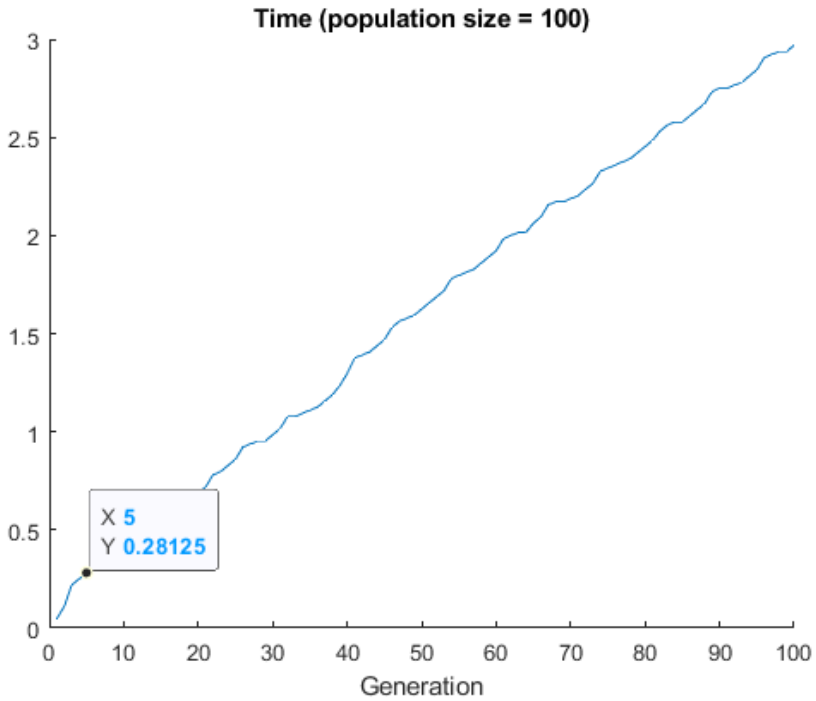


При 10 особях в популяции алгоритму удаётся найти решение уже на 36ой итерации затратив при этом примерно 0.97 секунд. Более того, на 15ой итерации он был крайне близок к решению.

Рассмотрим populationSize=100





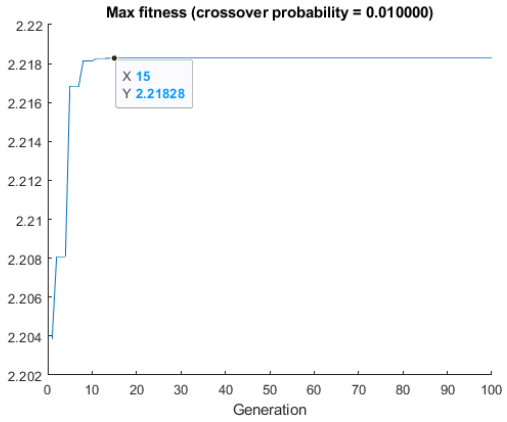


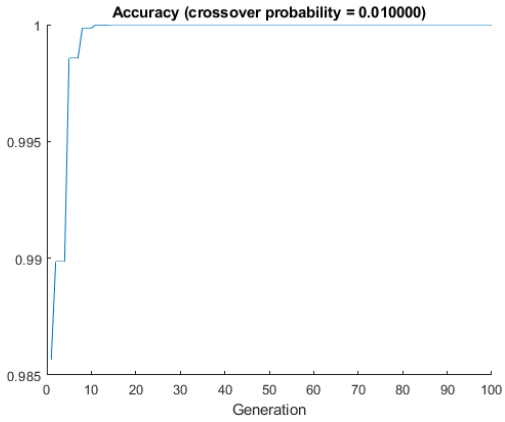
Ситуация меняется: решение находится всего за 5 поколений и 28 миллисекунд!

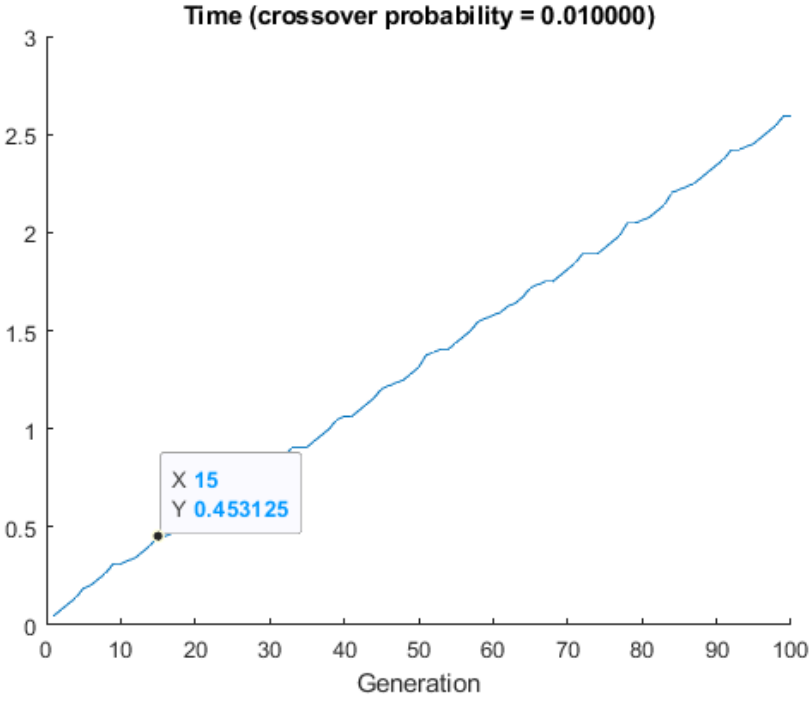
Зависимость от вероятности кроссинговера

Оператор кроссинговера позволяет скрещивать особи с заданной вероятностью, что реализует передачу полезной информации следующим поколениям.

Рассмотрим crossoverProbability=0.01



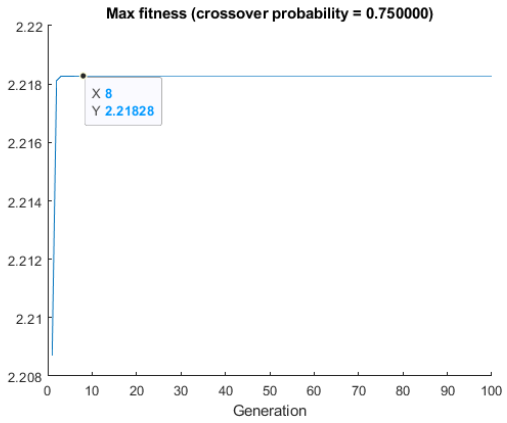


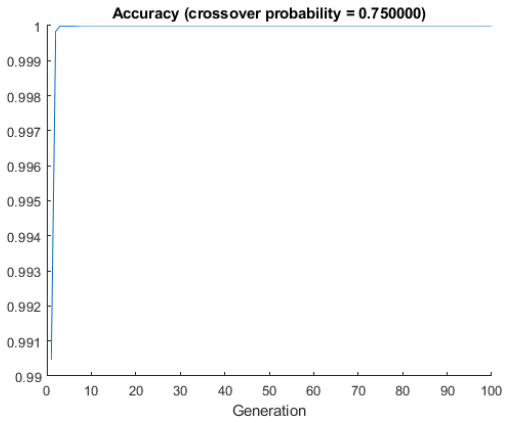


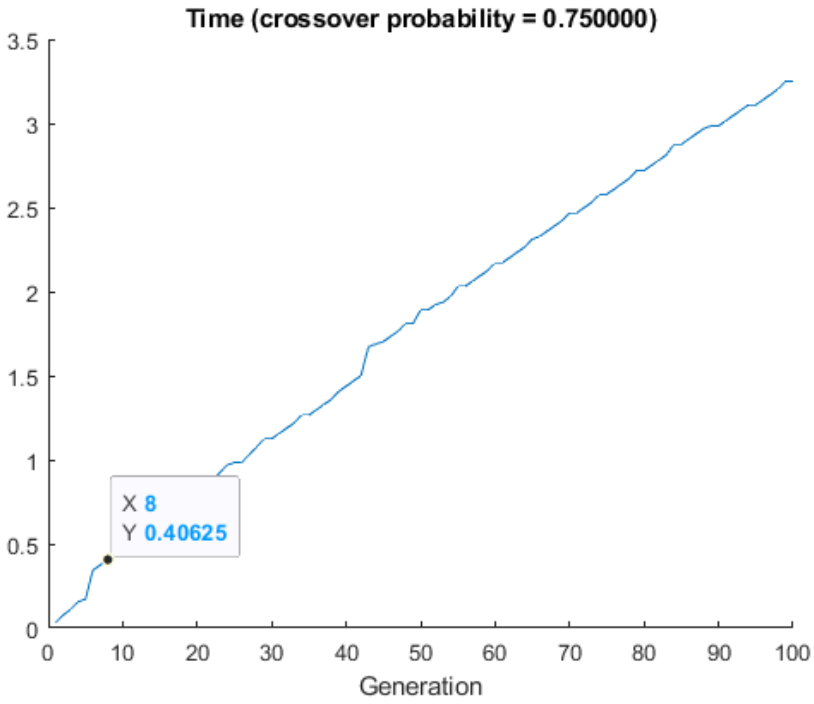
Несмотря на то, что вероятность скрещивания очень мала, уже на 15ом поколении решение найдено. Это связано с большим количеством особей в популяции, ведь некоторые из них уже могли оказаться около искомого оптимума.

Тем не менее с увеличением этой вероятности, результаты закономерно улучшаются.

Рассмотрим crossoverProbability=0.75





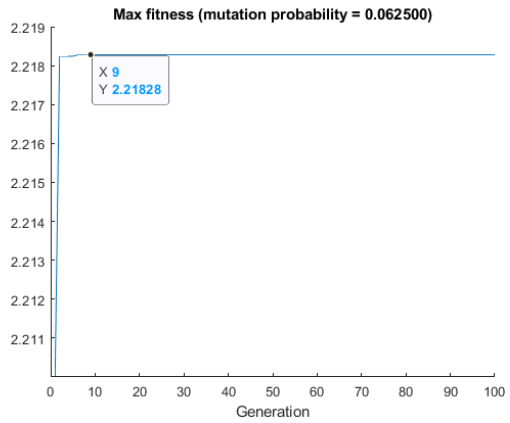


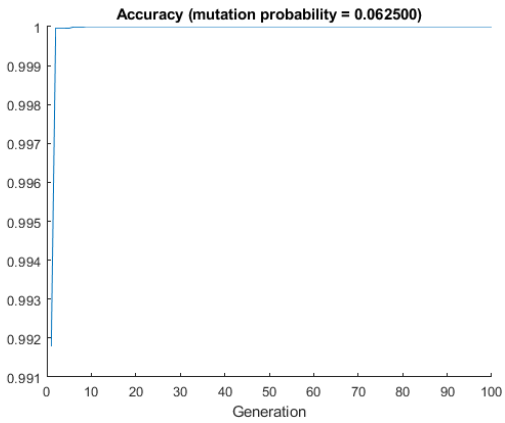
Как видно, решение найдено уже на 8мом поколении (в прошлом разделе вероятность скрещивания была 0.8 и решение нашлось за 5 поколений). Однако это не значит, что увеличение вероятности скрещивания обязательно повышает эффективность алгоритма.

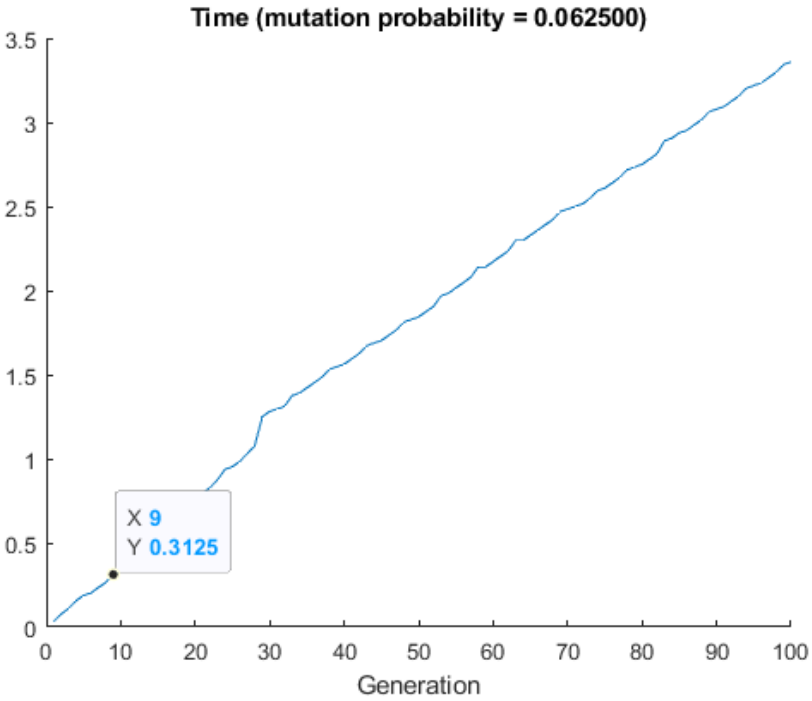
Зависимость от вероятности мутации

Оператор мутации позволяет очередному потомку родиться с некоторым дефектом. Это обеспечивает разнообразие внутри самой популяции.

Рассмотрим mutationProbability=0.0625



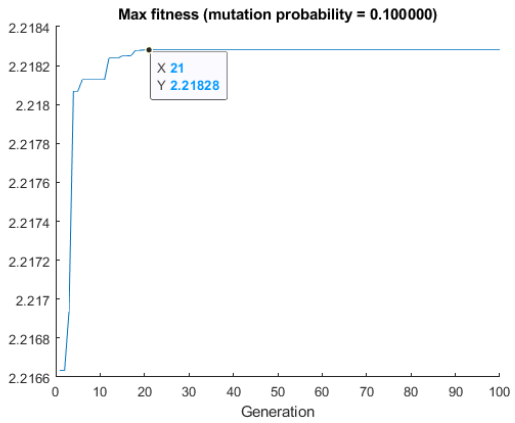


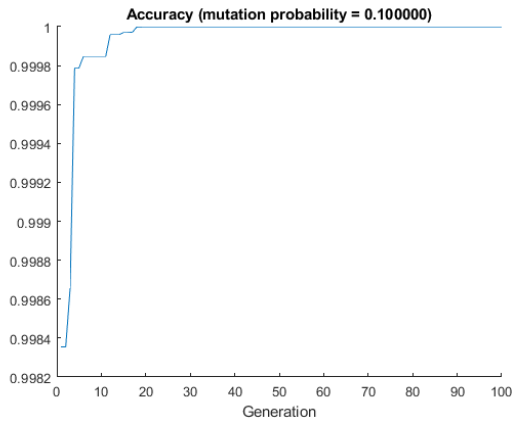


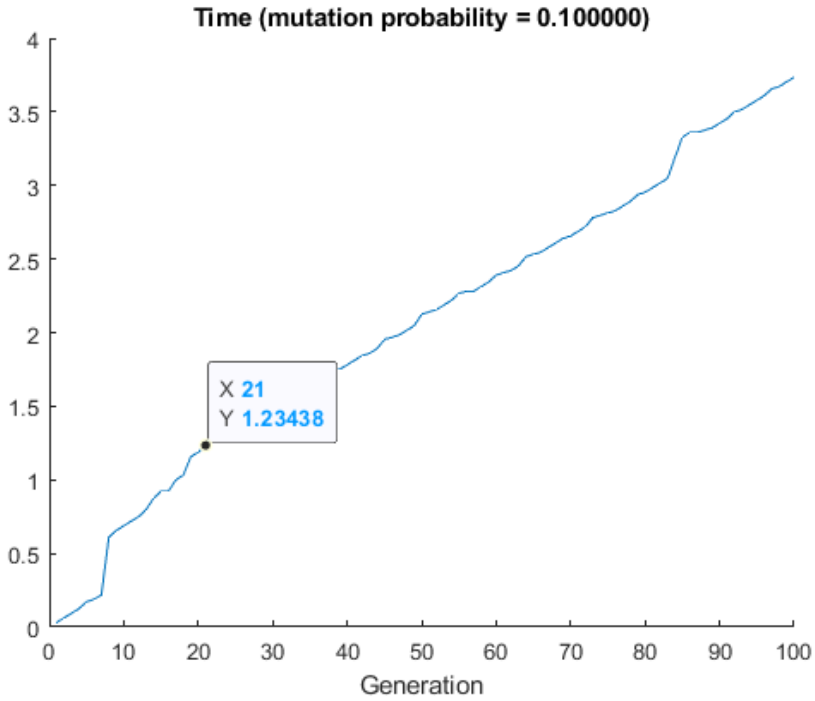
Как видно, алгоритм справился довольно неплохо и уже на 9ой итерации сошёлся к решению.

Однако...

Рассмотрим mutationProbability=0.1



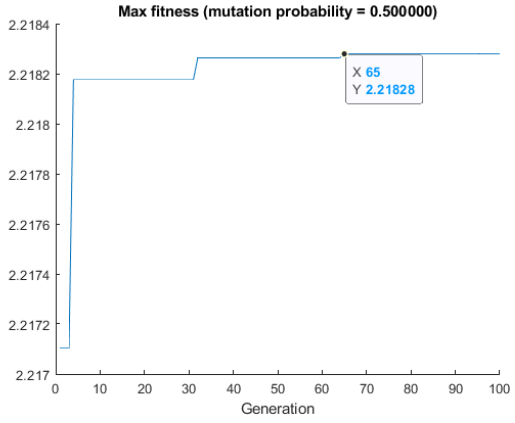


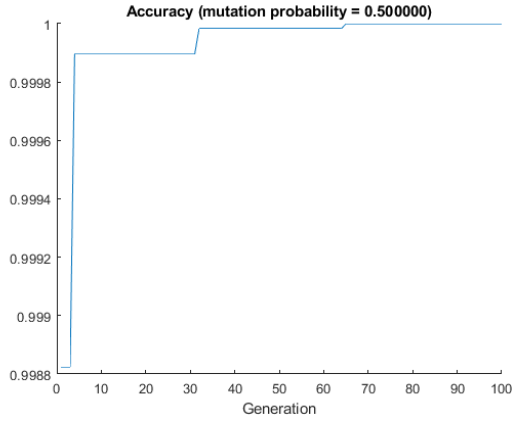


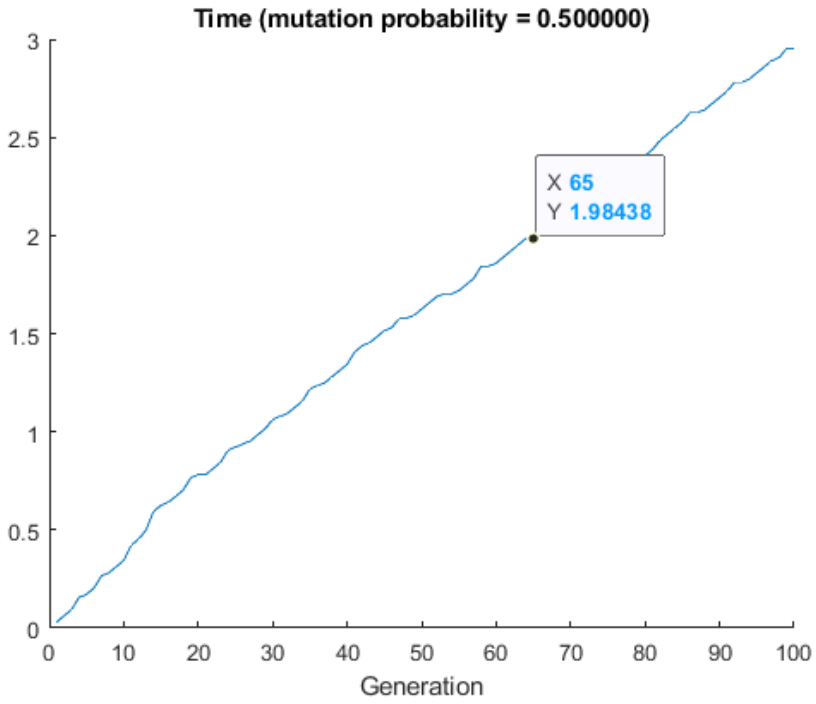
Стоило сделать очередную популяцию более “разнообразной”, поиск оптимального решения затянулся вплоть до 21го поколения!

Более того...

Рассмотрим mutationProbability=0.5







Ситуация стала плачевной: оптимум хоть и нашёлся, но ценой 65ти поколений, да и на поиск ушло почти 2 секунды!

Выводы

В данной работе я познакомился с пакетом Matlab и реализовал на нём простой генетический алгоритм.

Познакомился с основными средствами языка для построения графиков.

Узнал, как меняется эффективность поиска максимума на отрезке в зависимости от основных параметров алгоритма.

Установил, что размер популяции повышает скорость нахождения решения, однако ценой бОльшего использования ресурсов компьютера.

Определил, что малая вероятность скрещивания даёт результаты хуже, чем более высокая. Однако слишком высокая вероятность способствует более жадному поиску экстремумов. Из-за этого плотность популяции стягивается к тем областям, где она и так велика в том предположении, что "нас много там, где хорошо".

Заметил, что при больших значения вероятности мутации, генетический алгоритм теряет свои преимущества и становится более схож со случайным поиском.

Таким образом, алгоритм явился мне крайне эффективным при правильном подборе гиперпараметров.