МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра систем автоматизированного проектирования

Отчет по лабораторной работе № 7–8

на тему: «Построение простого генетического алгоритма»

по курсу «Методы машинного обучения»

Выполнили:   
студенты гр. КТбо4-4

Батагов С.С.\_\_\_\_\_\_\_\_\_\_

Москаленко М.В.\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Проверил:

Доцент

Лебедев О.Б.\_\_\_\_\_\_\_\_\_

(подпись, дата)

Таганрог 2022

**СОДЕРЖАНИЕ**

[ЛАБОРАТОРНАЯ РАБОТА № 7 3](#_Toc119620081)

[1 ЦЕЛЬ РАБОТЫ 3](#_Toc119620082)

[2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 3](#_Toc119620083)

[3 ПРАКТИЧЕСКАЯ ЧАСТЬ 8](#_Toc119620087)

[4 ВЫВОД 18](#_Toc119620091)

[ЛАБОРАТОРНАЯ РАБОТА № 8 19](#_Toc119620092)

[1 ЦЕЛЬ РАБОТЫ 19](#_Toc119620093)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 19](#_Toc119620094)

[ПРИЛОЖЕНИЕ А 21](#_Toc119620097)

# ЛАБОРАТОРНАЯ РАБОТА № 7

## 1 ЦЕЛЬ РАБОТЫ

Ознакомиться с существующими простыми генетическими алгоритмами и стратегиями их выполнения. Изучить применение фундаментальной теоремы генетических алгоритмов. Научиться строить и разрабатывать генетические алгоритмы.

## 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Согласно Холланду генетические схемы поиска оптимальных решений включают следующие этапы процесса эволюции:

* Конструируется начальная популяция. Вводится начальная точка отсчета поколений t = 0. Вычисляются приспособленность хромосом популяции (целевая функция) и средняя приспособленность всей популяции.
* Устанавливается значение t = t+1. Выбираются два родителя (хромосомы) для кроссинговера. Выбор осуществляется случайным образом пропорционально жизнеспособности хромосом, которая характеризуется значениями целевой функции.
* Формируется генотип потомка. Для этого с заданной вероятностью над генотипами выбранных хромосом производится операция кроссинговера. Случайным образом выбирается один из потомков A(t), к которому с заданными вероятностями последовательно применяются операторы инверсии и мутации. Полученная хромосома сохраняется как A’(t).
* Обновление текущей популяции путем замены случайно выбранной хромосомы на A’(t).
* Определение приспособленности A’(t) и пересчет средней приспособленности популяции.
* Если t=t\*, где t\* – заданное число шагов, то переход к этапу 7, в противном случае – переход к этапу 2.
* Конец работы.

Основная идея эволюции, заложенная в различные конструкции генетических алгоритмов, проявляется в способности «лучших» хромосом оказывать большее влияние на состав новой популяции за счет длительного выживания и более многочисленного потомства.

Простой генетический алгоритм включает операцию случайной генерации начальной популяции хромосом и ряд операторов, обеспечивающих генерацию новых популяций на основе начальной. Этими операторами являются репродукция, кроссинговер и мутация.

Репродукцией называется процесс копирования хромосом с учетом значений целевой функции, т.е. хромосомы с «лучшими» Значениями целевой функции имеют большую вероятность попадания в следующую популяцию. Этот процесс является аналогией митозного деления клеток. Выбор клеток (хромосом) для репродукции проводится в соответствии принципом «выживания сильнейшего». Простейшим способом представления операции репродукции в алгоритмической форме является колесо рулетки, в котором каждая хромосома имеет поле, пропорциональное значению целевой функции.

В алгоритмических реализациях механизма воспроизводства хромосом следует придерживаться следующих правил:

* Выбор начальной популяции можно выполнять произвольным образом, например подбрасыванием монеты.
* Репродукция осуществляется на основе моделирования движения колеса рулетки.
* Оператор кроссинговера реализуется как взаимный обмен короткими фрагментами двоичных строк гомологичных хромосом.
* Вероятность оператора кроссинговера принимается равной Р(CO)<1.
* Вероятность оператора мутации принимается равной Р(МO)>0.001.
* Рассмотрим пример применения простого генетического алгоритма для максимизации функции f(x)=x2 на целочисленном интервале [0, 31].
* Значения аргумента функции изменяющегося в интервале от 0 до 31, можно представить пятиразрядными двоичными числами. Первоначальная популяция, состоящая из четырех пятиразрядных чисел, получена с помощью процедуры генерации случайных чисел.

## 3 ПРАКТИЧЕСКАЯ ЧАСТЬ

Полный код рассматриваемой программы представлен в приложении А.

OneMax (или One-Max) – это простая задача оптимизации, которую часто приводят в пример как аналог программы «Hello World» в мире генетических алгоритмов.

Задача OneMax состоит в том, чтобы найти двоичную строку заданной длины, для которой сумма составляющих ее цифр максимальна.

Очевидно, что решением всегда является строка, состоящая из одних единиц. Но генетический алгоритм не обладает таким знанием, поэтому должен слепо искать решение, пользуясь генетическими операторами. Если алгоритм справится с работой, то найдет решение (или приближение к нему) за разумное время (Рисунок 1).

Рисунок 1 — Блок-схема выполнения генетического алгоритма

Генерируемое в начале алгоритма популяция будет представлено в виде набора бинарный хромосом, то есть содержащих 1 и 0. Количество хромосом и количество генов будет устанавливаться с помощью констант, с целью сохранения гибкости на стройки и разработки ГА на языке программирования Python.

Для отбора подходящий для кроссинговера хромосом будут отбираться с помощью турнирного метода отбора, который рассматривался в предыдущих работах. На этапе кроссинговера будет использоваться метод одноточечного кроссинговера, так как данный метод прост в реализации, а также подходит для выполнения поставленной задачи. Мутация будет выполняться путём точечного инвертирования, то есть будет выбрано случайны ген хромосомы, далее будет заменён на противоположный, в данной задаче 1 будет заменена на 0, или 0 — на 1.

Для работы разрабатываемой программы необходимо иметь набор библиотек, в данной задаче будет нужно: *random*, для генерации псевдослучайных чисел, *matplplotlib*, библиотека с набором математических функция для представления информации в виде графиков и гистограмм (Рисунок 2).

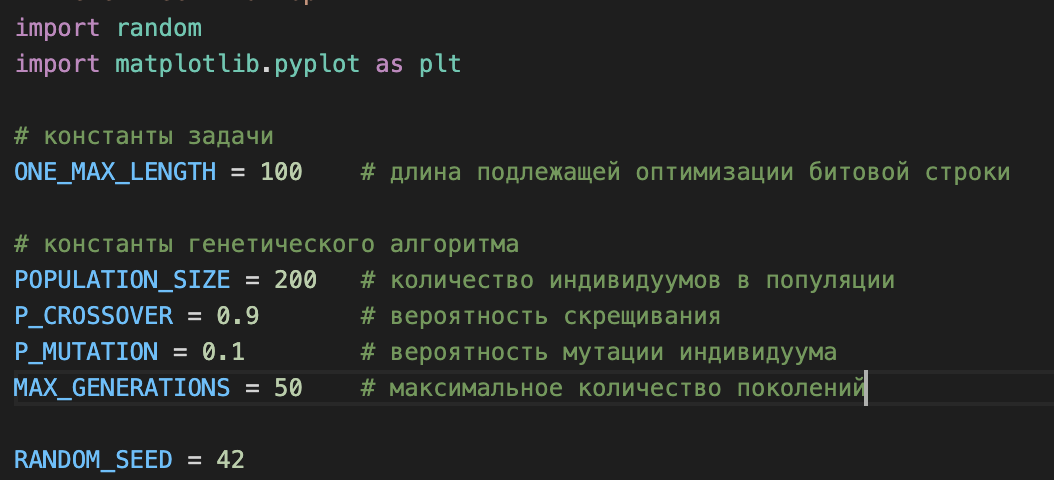


Рисунок 2 — Подключение библиотек и создание констант

Представленные константы являются глобальными для разрабатываемой программы, поэтому являются основными настройками ГА, для получения разного результата.

Далее в целях тестирования и отладки программы будут использованы одни и те же данные, поэтому были использована команда random.seed().

Для представления каждого индивидуума в популяции был создан класс с именем Individual, как показано на рисунке 3, который будет наследоваться от базового класса list (это класс для представления списков в Python), то есть, хромосома особи будет представлена бинарным списком и содержать целые числа 0 или 1.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 — Класс для представления индивида\хромосомы в популяции

В конструкторе этого класса будет вызван конструктор базового класса, а также дополнительно определим локальное свойство fitness, через которое будет получать значение приспособленности данного индивидуума. Для этого дополнительно объявлен вспомогательный класс FitnessMax и при создании экземпляра этого класса в нем появляется свойство values, которое и определяет степень приспособленности особи.

Далее, определим функцию принадлежности, которая будет вычислять приспособленность отдельной особи, а также две функции для создания отдельного индивидуума и всей популяции (Рисунок 4).



Рисунок 4 — Создания популяции, индивида и определение приспособленности

После этого создаем начальную популяцию и определяем счетчик поколений, а также вычислим текущие значения приспособленности для каждой особи в начальной популяции, после сохраним полученные значения в свойстве values каждого индивидуума (Рисунок 5).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 — Создание популяции и вычисление приспособленности каждого индивида\хромосомы

Далее определим вспомогательные списки для хранения лучшей и средней приспособленности в каждом текущем поколении. После необходимо разработать функции для клонирования индивида, выполнения турнирного отбора, одноточечного скрещивания и мутации (Рисунок 6).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 — Создание списков и функций для каждого этапа алгоритма

Функция *clone*, клонирует рассматриваемую хромосому, используется на этапе отбора для формирования новой популяции. *SelTournament*, реализуется метод турнирного отбора. Функция *cxOnePoin*t, используется на этапе кроссинговера, а именно реализуется одноточечный метод. Метод одноточечной мутации представлена в функции *mutFlipBit*.

Перед главным циклом ГА происходит вычисления списка значений приспособленностей для всех хромосом в популяции (Рисунок 7).



Рисунок 7 — Создание списка приспособленности

Далее в условиях цикла указывается или достижение максимальной приспособленности, или достижение максимального числа поколений. Непосредственно в цикле увеличивается счетчик популяции, выполняется турнирный отбор для выбора родителей. После оператора отбора, необходимо создать копии выбранных особей, так как некоторые могут быть отобраны дважды и, фактически, получим две разные ссылки на один и тот же список, что может нарушить логику работы генетического алгоритма, поэтому необходимо выполнить клонирование данных.

Далее, последовательно выбираем четные и нечетные особи в качестве родителей и с вероятностью P\_CROSSOVER выполняем операцию одноточечного скрещивания.

Подобные действия выполняются и для мутации. Перебираются особи в формируемой популяции и с вероятностью P\_MUTATION, а после подвергаются случайному изменению. При этом, дополнительно определяется вероятность indpb для инвертирования конкретного бита (гена) в хромосоме.

После этого пересчитывается приспособленностью особей, далее обновляется свойство values, список популяции population и список fitnessValues.

В конце цикла выбираем максимальное и среднее значения приспособленности в текущей популяции, добавляется в соответствующие списки: maxFitnessValues и meanFitnessValues, после выводим текущую информацию в консоль.

Код описанного цикла представлен на рисунке 8.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 — Основной цикл генетического алгоритма

Для представления полученной информации в графическом виде для более наглядного представления воспользуемся встроенными функциями библиотеки matplotlib, и выведем график. На рисунке 9, представлен код, отвечающий за создания графиков.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 — Представление данных в виде графика

Запустим ранее написанную программу простого генетического алгоритма (Рисунок 10-11).

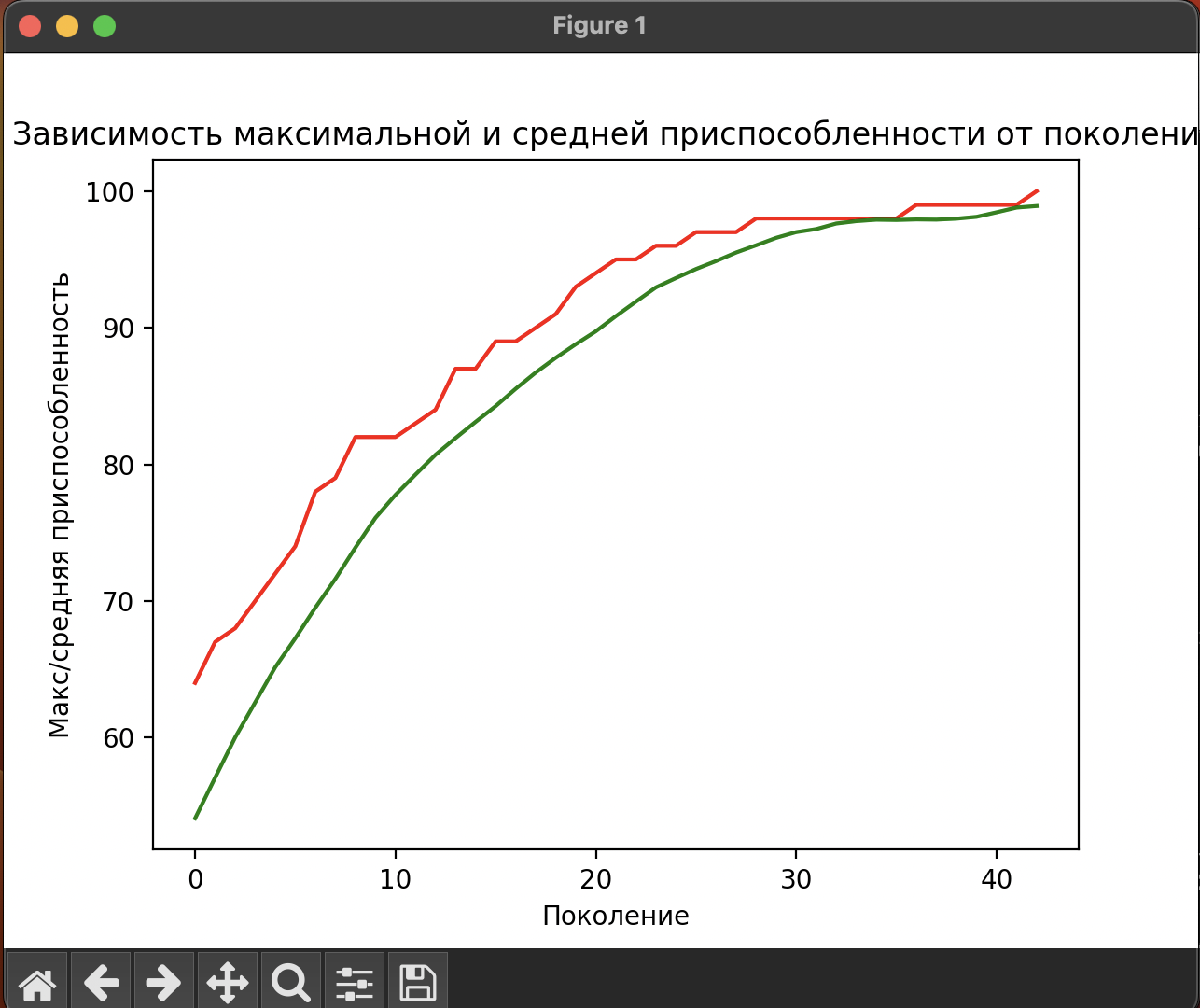


Рисунок 10 — Графическое представление данных

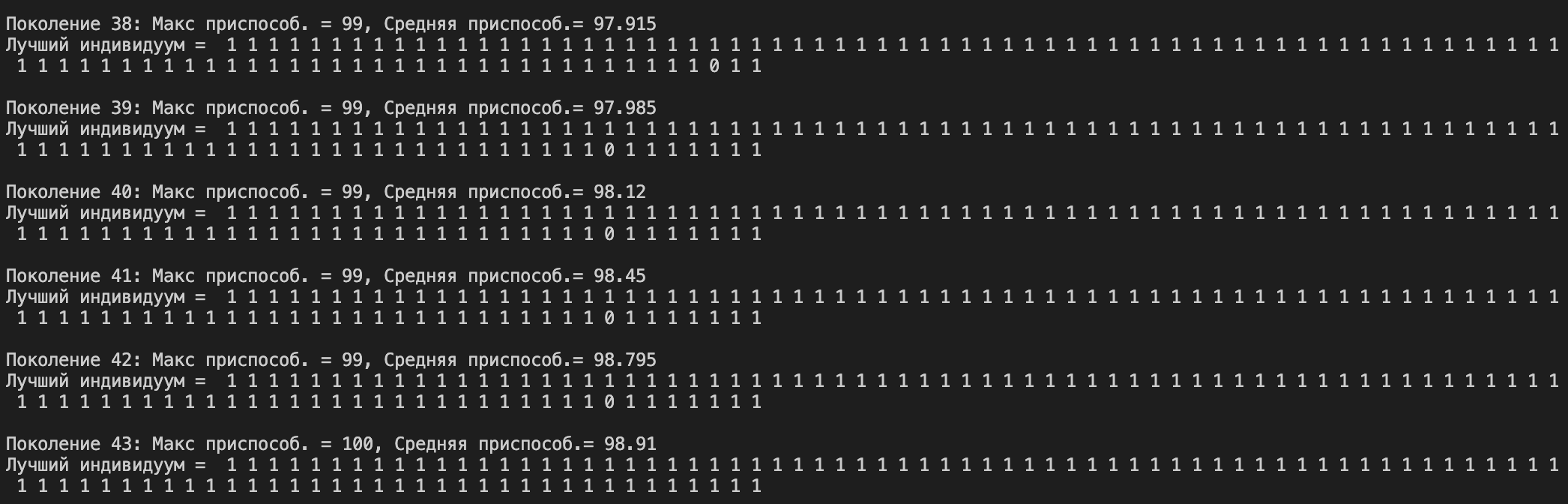


Рисунок 11 — Вывод результата в консоль

## 4 ВЫВОД

В данной лабораторной работе был изучен и воспроизведен механизм простого генетический алгоритма, а именно вариация алгоритма с использованием принципа OneMax. Алгоритм написан на языке Python, с использованием библиотеки pyplot для построения графиков.

# ЛАБОРАТОРНАЯ РАБОТА № 8

## 1 ЦЕЛЬ РАБОТЫ

Ознакомиться с основными методами проведения статистических исследований экспериментальных данных.

## 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

Проведём несколько экспериментов, связанных с изменением основных констант программы.

Изображение выглядит как текст, экран, снимок экрана, закрыть

Автоматически созданное описание

Рисунок 1 — Изначальные константы

Проведём 3 эксперимента.

**Первый**, изменим количество генов у индивида, то есть количество элементов в списке (Рисунок 2).

Изображение выглядит как текст, снимок экрана, экран, закрыть

Автоматически созданное описание

Рисунок 2 — Константы первого эксперимента

По окончанию выставления новых константных данных, запустим программу (Рисунок 3-4).

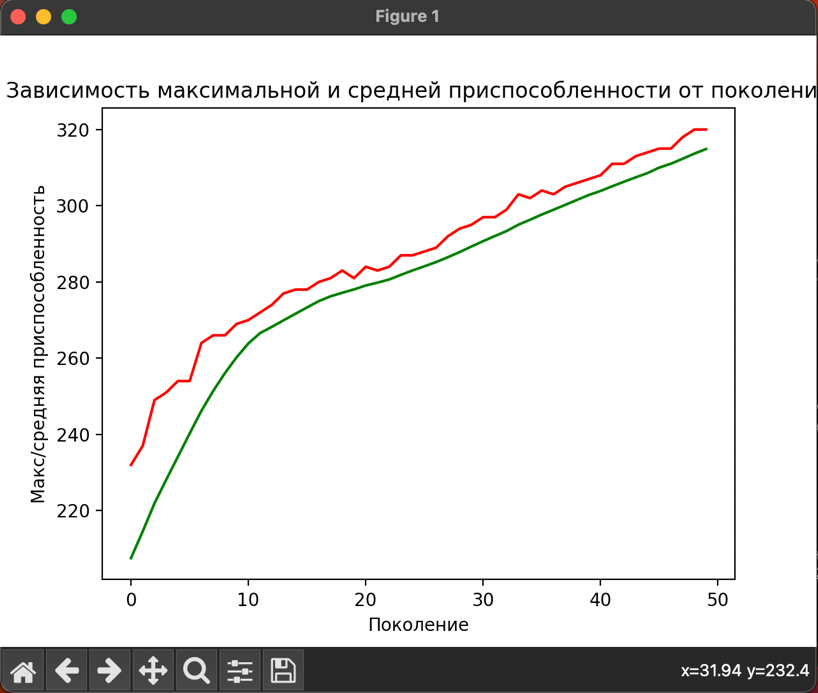


Рисунок 3 — График на основе новых данных

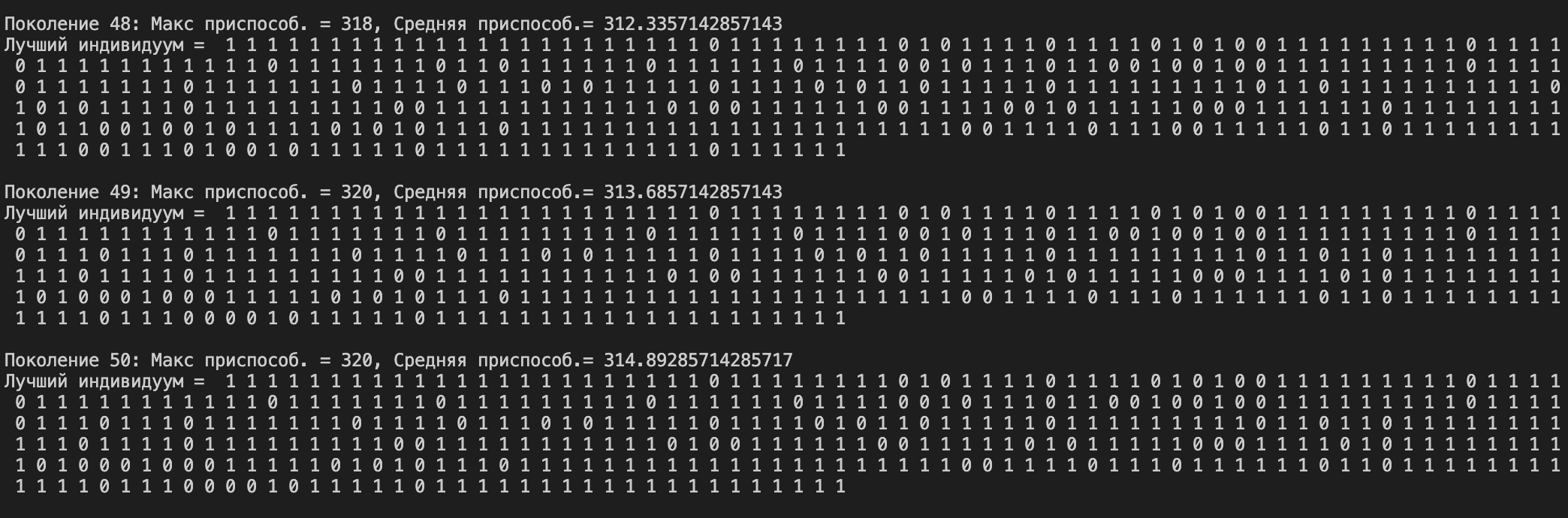


Рисунок 4 — Результат первого эксперимента в консоли

В качестве второго эксперимента изменим имеющиеся константы на следующие значения (Рисунок 5).

Изображение выглядит как текст, экран, снимок экрана, закрыть

Автоматически созданное описание

Рисунок 5 — Значения констант для второго эксперимента

После чего запустим программу. Результат выполнения представлен на рисунке 6–7.

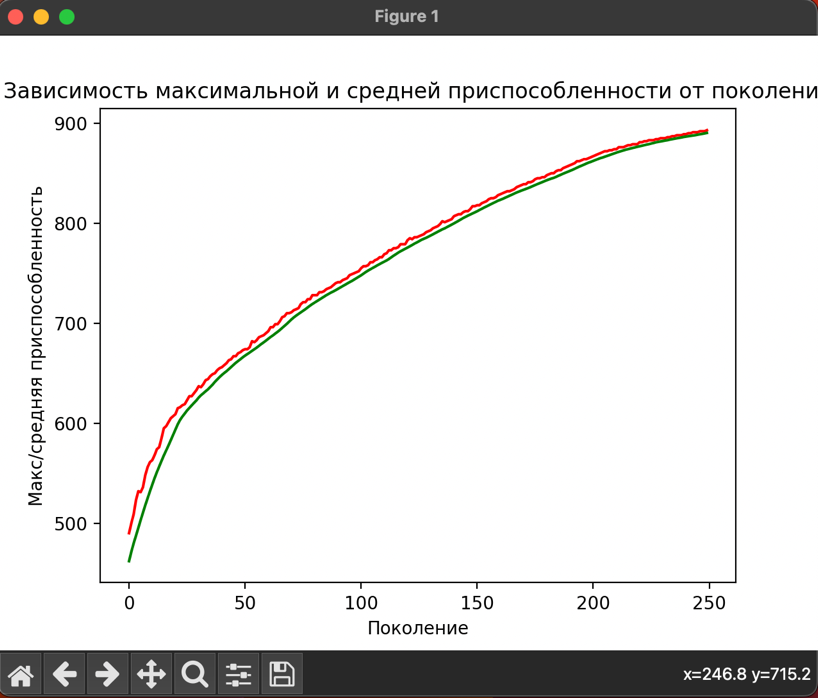


Рисунок 6 — Результат второго эксперимента

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 — Результат второго эксперимента в консоли

Изображение выглядит как текст, снимок экрана, экран, закрыть

Автоматически созданное описание

Рисунок 8 — Новые значения констант

Последним, третьим экспериментом будет изменение констант следующим образом (Рисунок 8).

После этого запустим программу (Рисунок 9-10).

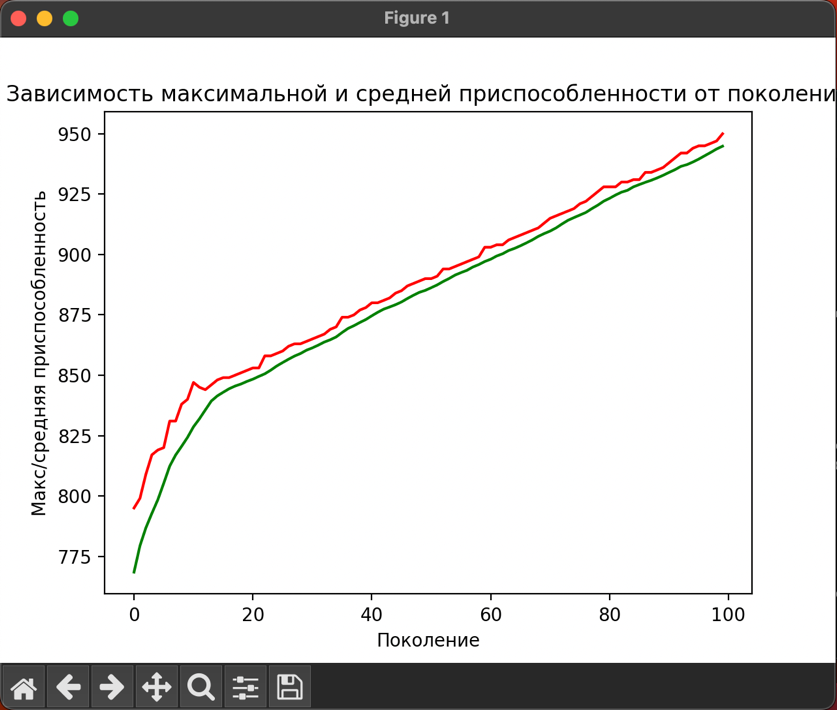


Рисунок 9 — Результат третьего эксперимента

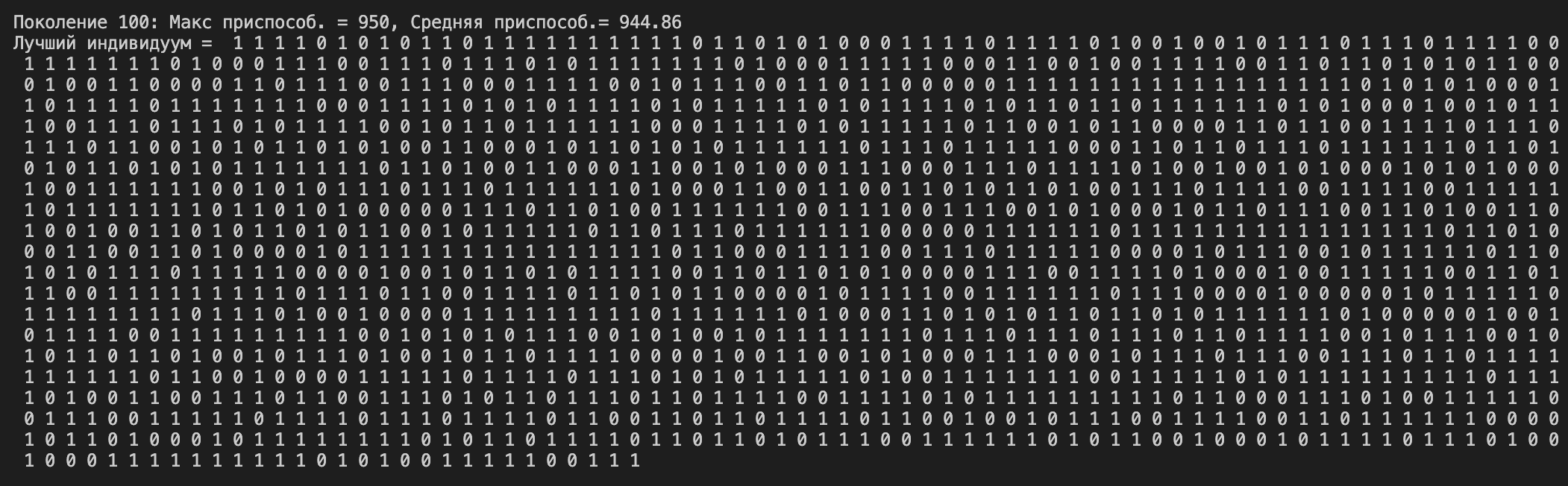


Рисунок 10 — Результат третьего эксперимента в консоли

## 4 ВЫВОД

Исходя из представленных экспериментов и полученных результатов можно сделать вывод, чтобы добиться более близкого значения между средней приспосабливаемостью и максимальной можно добиться, только при большем количестве генов у индивида, большей вероятности мутации генов индивида, и большего количества поколений, или большего количества индивидов в популяции.

По результатм исследований видно, что для достижения близких значений между средней и максимальной приспосабливаемостью необходимо большое количество генов у индивида, большая вероятность мутации генов индивида, большее количество поколений или больший объем популяции.

# ПРИЛОЖЕНИЕ А

**Программа простого ГА**

Ниже будет представлен исходный код программы простого генетического алгоритма.

"""Генетический алгоритм"""

import random

import matplotlib.pyplot as plt

# константы задачи

ONE\_MAX\_LENGTH = 100 # длина подлежащей оптимизации битовой строки

# константы генетического алгоритма

POPULATION\_SIZE = 200 # количество индивидуумов в популяции

P\_CROSSOVER = 0.9 # вероятность скрещивания

P\_MUTATION = 0.1 # вероятность мутации индивидуума

MAX\_GENERATIONS = 50 # максимальное количество поколений

RANDOM\_SEED = 42

#random.seed(RANDOM\_SEED)

class FitnessMax():

def \_\_init\_\_(self):

self.values = [0]

class Individual(list):

def \_\_init\_\_(self, \*args):

super().\_\_init\_\_(\*args)

self.fitness = FitnessMax()

def oneMaxFitness(individual):

return sum(individual), # кортеж

def individualCreator():

return Individual([random.randint(0, 1) for i in range(ONE\_MAX\_LENGTH)])

def populationCreator(n = 0):

return list([individualCreator() for i in range(n)])

population = populationCreator(n=POPULATION\_SIZE)

generationCounter = 0

fitnessValues = list(map(oneMaxFitness, population))

for individual, fitnessValue in zip(population, fitnessValues):

individual.fitness.values = fitnessValue

maxFitnessValues = []

meanFitnessValues = []

def clone(value):

ind = Individual(value[:])

ind.fitness.values[0] = value.fitness.values[0]

return ind

def selTournament(population, p\_len):

offspring = []

for n in range(p\_len):

i1 = i2 = i3 = 0

while i1 == i2 or i1 == i3 or i2 == i3:

i1, i2, i3 = random.randint(0, p\_len-1), random.randint(0, p\_len-1), random.randint(0, p\_len-1)

offspring.append(max([population[i1], population[i2], population[i3]], key=lambda ind: ind.fitness.values[0]))

return offspring

def cxOnePoint(child1, child2):

s = random.randint(2, len(child1)-3)

child1[s:], child2[s:] = child2[s:], child1[s:]

def mutFlipBit(mutant, indpb=0.01):

for indx in range(len(mutant)):

if random.random() < indpb:

mutant[indx] = 0 if mutant[indx] == 1 else 1

fitnessValues = [individual.fitness.values[0] for individual in population]

while max(fitnessValues) < ONE\_MAX\_LENGTH and generationCounter < MAX\_GENERATIONS:

generationCounter += 1

offspring = selTournament(population, len(population))

offspring = list(map(clone, offspring))

for child1, child2 in zip(offspring[::2], offspring[1::2]):

if random.random() < P\_CROSSOVER:

cxOnePoint(child1, child2)

for mutant in offspring:

if random.random() < P\_MUTATION:

mutFlipBit(mutant, indpb=1.0/ONE\_MAX\_LENGTH)

freshFitnessValues = list(map(oneMaxFitness, offspring))

for individual, fitnessValue in zip(offspring, freshFitnessValues):

individual.fitness.values = fitnessValue

population[:] = offspring

fitnessValues = [ind.fitness.values[0] for ind in population]

maxFitness = max(fitnessValues)

meanFitness = sum(fitnessValues) / len(population)

maxFitnessValues.append(maxFitness)

meanFitnessValues.append(meanFitness)

print(f"Поколение {generationCounter}: Макс приспособ. = {maxFitness}, Средняя приспособ.= {meanFitness}")

best\_index = fitnessValues.index(max(fitnessValues))

print("Лучший индивидуум = ", \*population[best\_index], "\n")

plt.plot(maxFitnessValues, color='red')

plt.plot(meanFitnessValues, color='green')

plt.xlabel('Поколение')

plt.ylabel('Макс/средняя приспособленность')

plt.title('Зависимость максимальной и средней приспособленности от поколения')

plt.show()