**Министерство образования Российской Федерации**

**МОСКВОСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э.БАУМАНА**

Факультет: Информатика и системы управления (ИУ)  
Кафедра: Информационная безопасность (ИУ8)

**МЕТОДЫ ОПТИМИЗАЦИИ**

**Рубежный контроль №2 на тему:**   
«Исследование генетических алгоритмов в задачах поиска экстремумов»

Вариант – 1

**Преподаватель:**   
Коннова Н.С.

**Студент:**   
Александров А. Н.

**Группа:**   
ИУ8-34

Москва, 2020

**Цель работы:**

изучить основные принципы действия генетических алгоритмов на примере

решения задач оптимизации функций двух переменных.

**Постановка задачи:**

Найти максимум функции в области с помощью простого (классического) генетического алгоритма.

За исходную популяцию принять 4 случайных точки. Хромосома каждой особи состоит из двух генов: значений координат x , y . В качестве потомков следует выбирать результат скрещивания лучшего решения со вторым и третьим в порядке убывания значений функции приспособленности с последующей случайной мутацией обоих генов.

В качестве критерия остановки эволюционного процесса задаться номером конечной популяции (). Визуализировать результаты расчетов.

**Ход решения:**

Исходная функция имеет вид:

Область допустимых значений

Для решения задачи была написана программа на языке **Python** (см. Приложение А):

***1. genetic\_algorithm.py***

Содержит реализацию класса ***GeneticAlgorithm,*** который инициализируется фит функцией и границами поиска экстремума.

Поля класса ***GeneticAlgorithm:***

Поле **fit\_func\_:** инициализируется лямбда функцией от двух аргументов – x и y.

Поле **iterations\_count\_:** содержит число итераций, которые совершает генетический алгоритм.

Поле **bounds\_:** содержит область допустимых значений аргументов x и y.

Поле **species\_count\_:** содержит число осыбей в популяции.

Поле **population\_number\_:** содержит номер текущей популяции.

Поле **population\_:** содержит текущую популяцию.

Поле **max\_result\_:** содержит максимальное значение фит функции за популяцию.

Поле **mean\_result\_:** содержит среднее значение фит функции за популяцию.

Методы класса ***GeneticAlgorithm:***

Метод ***PrintPopulation(self):*** выводит таблицу популяции, а также максимальное и среднее значения фит функции в ней.

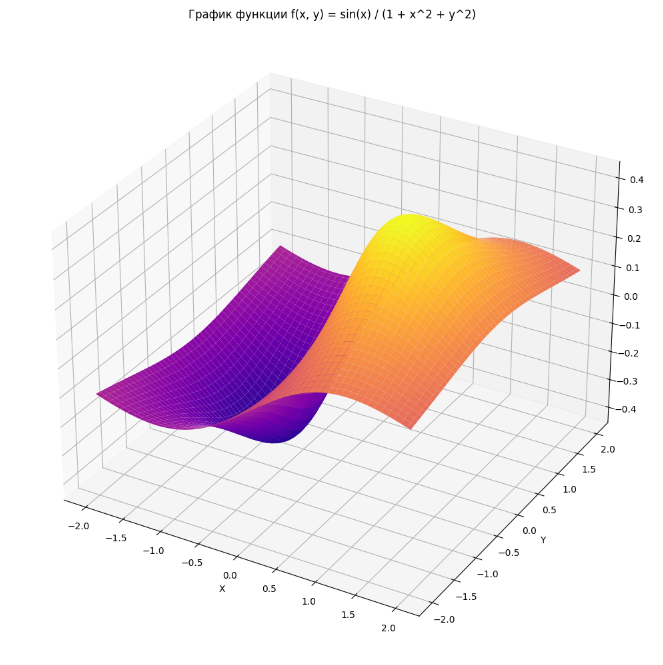
Метод ***Solve(self):*** производит решение задачи указанное число итераций.

Метод ***InitPopulation(self)***: задание начальной популяции случайными значениями хромосом в пределах допустимых значений аргументов.

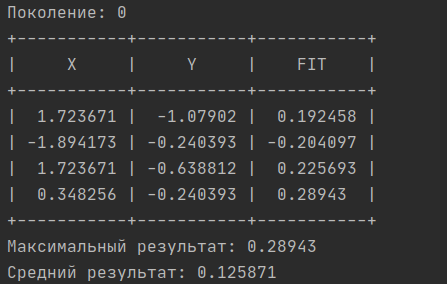
Метод ***Mutation(self)****:* производит мутацию текущего поколения.

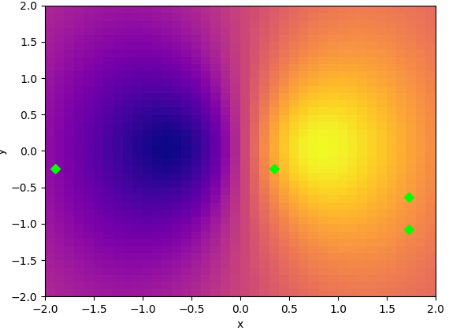
Метод ***CrossoverAndCalcFit(self)****:* производит скрещивание осыбей промежуточной популяции путём кроссовера и вычисляет значение фит функции.

Метод ***VisualizePopulation(self)****:* визуализирует поколение и сохраняет его в директорию /results.

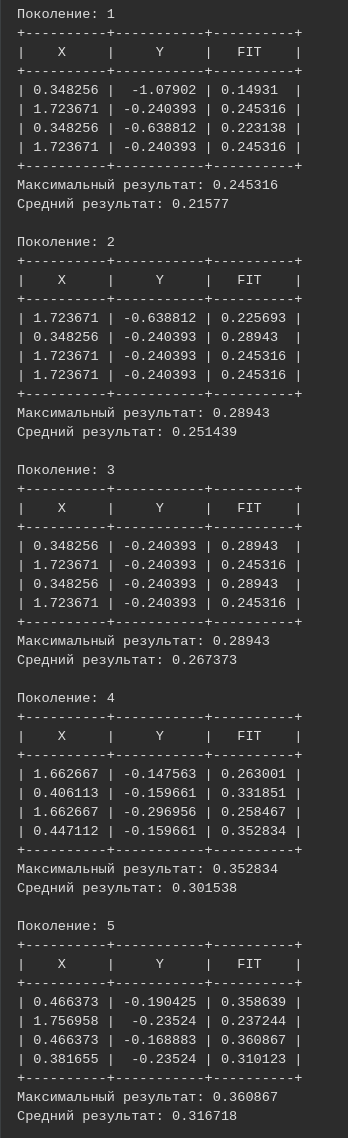
Рисунок 1 График исходной поверхности.

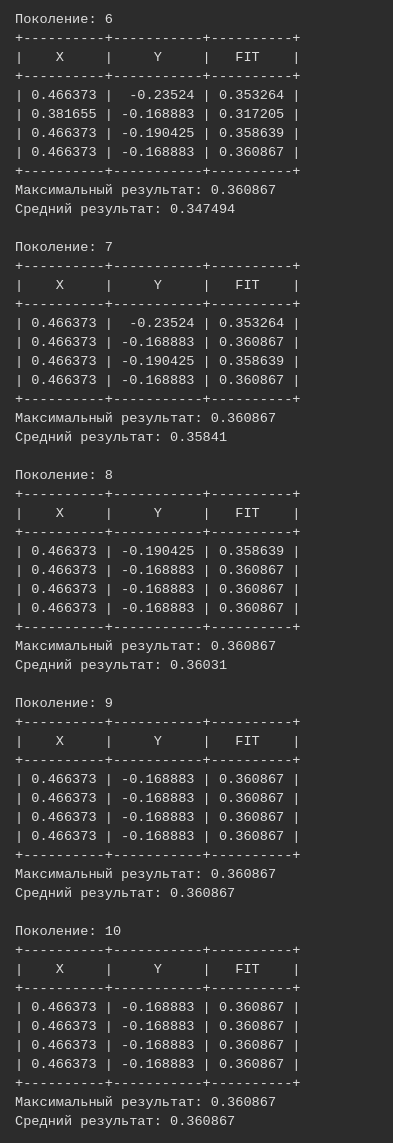
Начальное поколение:

****

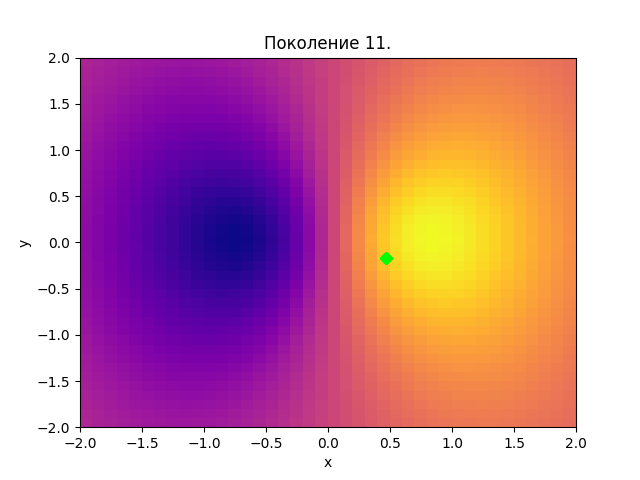
Рисунок 2 Начальное поколение

Первые 10 итераций алгоритма:

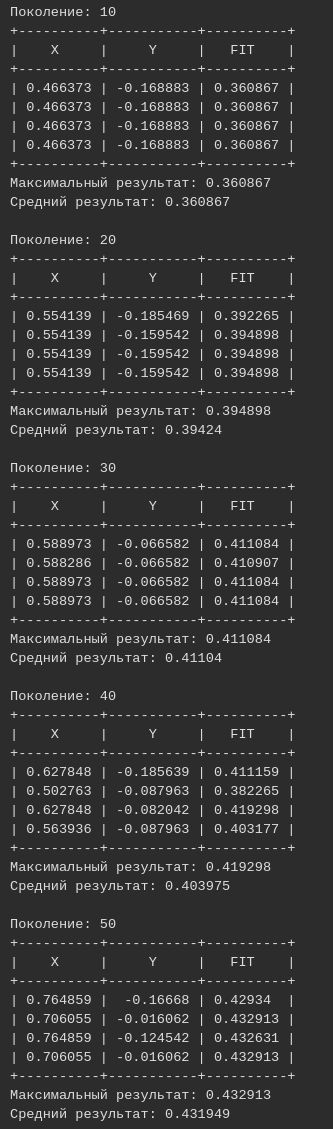
Рисунок 3 Первые 5 поколений.

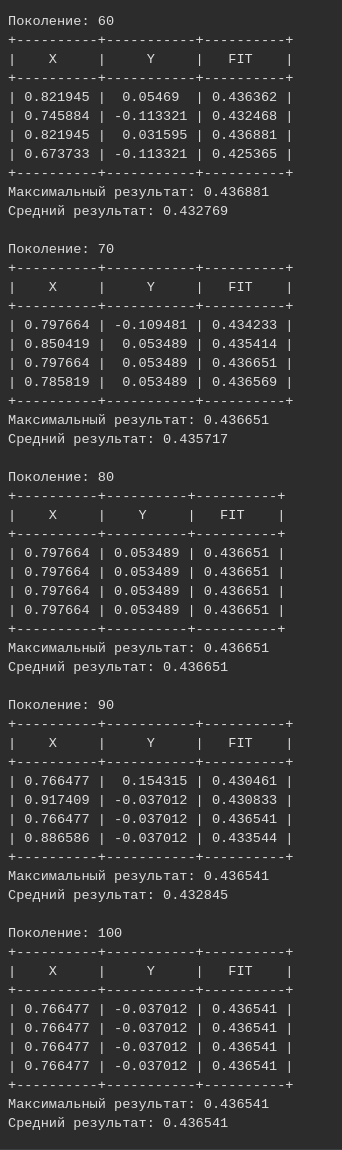
Рисунок 4 Поколения 5-10

Полученный результат 10 поколения:

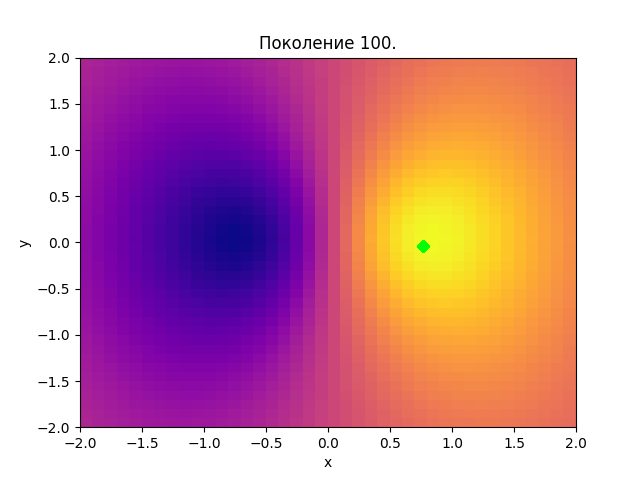
Рисунок 5 Визуализация поколения 10.

**Популяции N = 10..100 с шагом 10:**

Рисунок 6 Поколения 10, 20, 30, 40, 50.

Рисунок 7 Поколения 60, 70, 80, 90, 100

Полученный результат 100 поколения:

Рисунок 8 Поколение 100

**Вывод:**

Проделав работу, я изучил основные принципы действия генетических алгоритмов на примере поиска экстремума функции двух переменных; изучил основные шаги алгоритма, возможные критерии останова, виды алгоритмов селеции, операторов кроссовера и мутации.

По результатам численного эксперимента мы видим, что на 10 поколении результат ещё далековат от необходимого экстремума, но на 100 - совсем близок к идеальному.

Идея эволюционных алгоритмов в том, чтобы найти некое приближенное к лучшему, к оптимальному решению, которое скорее всего будет нас удовлетворять. Им можно найти применение для решения различных инженерных и практических задач.

К недостаткам генетических алгоритмов можно отнести ограниченность в способности нахожения верного решения. Они не могут регулировать логику развития поиска, существует вероятность, что в некоторых случаях алгоритм будет существенно ошибаться.

**Приложение А**

*Код программы*

***Файл “main.py”***

# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>  
  
import genetic\_algorithm as genetic  
import numpy as np  
  
# Вариант 1.   
# f(x, y) = sin(x) / (1 + x^2 + y^2)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 fit\_function = lambda x, y: np.sin(x) / (1 + x \*\* 2 + y \*\* 2)  
 bounds = [-2., 2., -2., 2.]  
 genetic.GeneticAlgorithm(fit\_function, bounds).Solve()

***Файл “genetic\_algorithm.py”***

# Copyright 2020 Alexey Alexandrov <sks2311211@yandex.ru>  
import os  
import random  
  
import imageio  
from mpl\_toolkits.mplot3d import Axes3D  
from matplotlib import cm  
import matplotlib.pyplot as plt  
  
import numpy as np  
from prettytable import PrettyTable  
  
ITERATIONS\_COUNT = 101  
SPECIES\_COUNT = 4  
MUTATE\_CHANCE = 0.25  
MUTATE\_CONSTANT = 5  
ROUND\_CONSTANT = 6  
  
  
class GeneticAlgorithm:  
 def \_\_init\_\_(self, fit\_func, bounds):  
 *"""Инициализация задачи по фит функции и границам диапазона."""*  
self.fit\_func\_ = fit\_func  
 self.iterations\_count\_ = ITERATIONS\_COUNT  
 self.bounds\_ = bounds  
 self.species\_count\_ = SPECIES\_COUNT  
 self.population\_number\_ = 0  
 self.population\_ = self.InitPopulation()  
  
 self.max\_result\_ = np.max(self.population\_[:, 2])  
 self.mean\_result\_ = np.mean(self.population\_[:, 2])  
  
 def PrintPopulation(self):  
 *"""Вывод таблицы."""*  
print(f"Поколение: {self.population\_number\_}")  
 table = PrettyTable()  
 table.field\_names = ["X", "Y", "FIT"]  
  
 for i in range(self.species\_count\_):  
 table.add\_row(list(np.round(self.population\_[i], ROUND\_CONSTANT)))  
  
 self.population\_number\_ += 1  
 print(table)  
  
 print(f"Максимальный результат: {round(self.max\_result\_, ROUND\_CONSTANT)}")  
 print(f"Средний результат: {round(self.mean\_result\_, ROUND\_CONSTANT)}")  
  
 def Solve(self):  
 *"""Решение задачи."""*  
for i in range(self.iterations\_count\_):  
 self.VisualizePopulation()  
 self.CrossoverAndCalcFit()  
 self.PrintPopulation()  
  
 # Mutate in 25% of population.  
 if random.uniform(0, 1) <= 0.25:  
 self.Mutation()  
  
 self.CreateGifVisualization()  
  
 def InitPopulation(self):  
 *"""Начальная популяция."""*  
population = np.random.rand(self.species\_count\_, 3)  
  
 # Столбец X.  
 population[:, 0] = self.bounds\_[0] + population[:, 0] \* (self.bounds\_[1] - self.bounds\_[0])  
 # Столбец Y.  
 population[:, 1] = self.bounds\_[2] + population[:, 1] \* (self.bounds\_[3] - self.bounds\_[2])  
 # Столбец FIT.  
 population[:, 2] = self.fit\_func\_(population[:, 0], population[:, 1])  
  
 return population  
  
 def Mutation(self):  
 *"""Мутация"""*  
delta = (np.random.rand(\*self.population\_.shape) - 0.5) / MUTATE\_CONSTANT  
 self.population\_ = self.population\_ + delta  
 x\_column = self.population\_[:][0]  
 y\_column = self.population\_[:][1]  
  
 satisfying\_x = np.where(np.logical\_and(x\_column > self.bounds\_[0], x\_column < self.bounds\_[1]))  
 satisfying\_y = np.where(np.logical\_and(y\_column > self.bounds\_[2], y\_column < self.bounds\_[0]))  
 if len(satisfying\_x) != len(x\_column) or len(satisfying\_y) != len(y\_column):  
 self.population\_ = self.population\_ - delta  
  
 def CrossoverAndCalcFit(self):  
 *"""Кроссовер и вычисление значений фит функции."""*  
curr\_population = self.population\_[self.population\_[:, 2].argsort()].copy()  
  
 x3 = curr\_population[3][0]  
 y3 = curr\_population[3][1]  
 x2 = curr\_population[2][0]  
 y2 = curr\_population[2][1]  
 x1 = curr\_population[1][0]  
 y1 = curr\_population[1][1]  
  
 new\_population = np.array(  
 [[x3, y1, self.fit\_func\_(x3, y1)], [x1, y3, self.fit\_func\_(x1, y3)],  
 [x3, y2, self.fit\_func\_(x3, y2)], [x2, y3, self.fit\_func\_(x2, y3)]])  
  
 self.max\_result\_ = np.max(new\_population[:, 2])  
 self.mean\_result\_ = np.mean(new\_population[:, 2])  
  
 self.population\_ = new\_population.copy()  
  
 def VisualizePopulation(self):  
 if not os.path.exists('results/'):  
 os.makedirs('results/')  
 x0, x1 = self.bounds\_[:2]  
 x = np.arange(x0, x1 + 0.1, 0.1)  
 y0, y1 = self.bounds\_[2:]  
 y = np.arange(y0, y1 + 0.1, 0.1)  
 x, y = np.meshgrid(x, y)  
 z = self.fit\_func\_(x, y)  
  
 x\_p, y\_p = np.transpose(self.population\_[:, :2])  
 plt.title(f"Поколение {self.population\_number\_}.")  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.pcolormesh(x, y, z, cmap=cm.plasma)  
 plt.plot(x\_p, y\_p, marker='D', linestyle='', color='lime')  
 plt.savefig('results/' + str(self.population\_number\_) + '.png')  
 # plt.show()  
 plt.clf()