****

**Министерство науки и высшего образования Российской Федерации**

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

Факультет: «Информатика и системы управления»

Кафедра: «Информационная безопасность (ИУ8)» / ГУИМЦ

**ТЕОРИЯ СИСТЕМ И СИСТЕМНЫЙ АНАЛИЗ**

**Рубежный контроль №3 на тему:**

«Исследование генетических алгоритмов в задачах поиска экстремумов»

Вариант 4

**Преподаватель:** Коннова Н. С.

**Студент:** Месяцева Н. В.

**Группа**: ИУ8Ц-51

Москва

2019

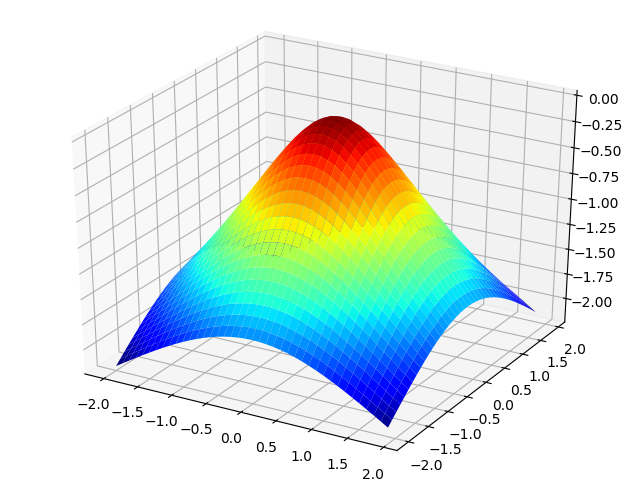
# **Цель работы**

Изучить основные принципы действия генетических алгоритмов на примере решения задач оптимизации функций двух переменных.

# **Постановка задачи**

Найти максимум функции в области D с помощью простого (классического) генетического алгоритма. За исходную популяцию принять 4 случайных точки. Хромосома каждой особи состоит из двух генов: значений координат . 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 работы**

Известно, что на области D = дана функция . Ниже представлен график данной функции.

**Рисунок 1. График функции f(x) = -ln (**

Максимизируем функцию с помощью простого генетического алгоритма. Для этого нужно написать программу, реализующую данный алгоритм. Первое, что в программе должно быть, - это создание четырёх особей (популяцию), имеющих по 2 хромосомы каждая (координаты ). Причём создание особей происходит случайным способом с помощью генератора. После создания особей программа должна выполнить расчет среднего и максимального значения FIT-функции для популяции. Также программа производит селекцию (*рулеточный отбор*) и последующий кроссовер особей (обмен генами). Учитывается мутация, принятая равной 25%. Данные действия повторяются для каждого поколения (итерации алгоритма) для достижения критерия останова алгоритма, в данном случае – номера поколения N = 10.

Программы произвела вычисления, которые собраны в таблице ниже.

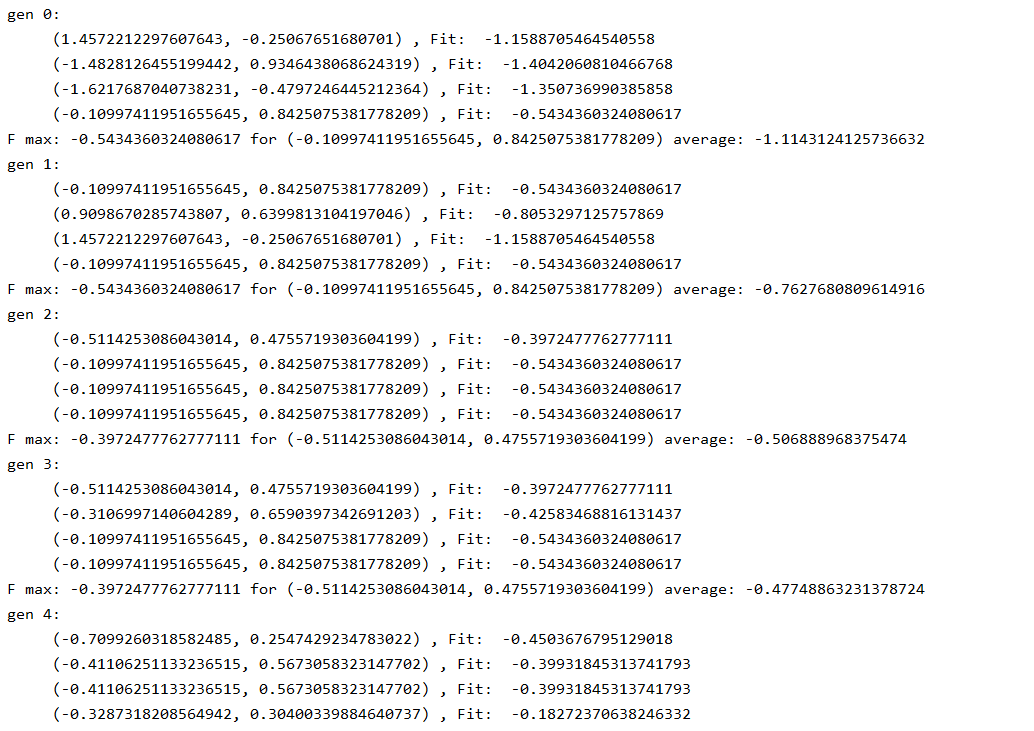
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№**  **поколения** | **X** | **Y** | **FIT** | **Максимальный результат** | **Средний результат** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0**  **(исходное)** | 1.457221 | -0.250676 | -1.1588705 | -0.543436 | -1.114312 |
| -1.482812 | 0.934643 | -1.404206 |
| -1.621768 | -0.479724 | -1.350736 |
| -0.109974 | 0.842507 | -0.543436 |
| **1** | -0.109974 | 0.842507 | -0.543436 | -0.5434360 | 0.7627680 |
| 0.909867 | 0.639981 | -0.805329 |
| 1.457221 | -0.250676 | -1.158870 |
| -0.109974 | 0.842507 | -0.543436 |
| **2** | -0.511425 | 0.475571 | -0.397247 | -0.397247 | -0.506888 |
| -0.109974 | 0.842507 | -0.543436 |
|  | -0.109974 | 0.842507 | -0.543436 |  |  |
| -0.109974 | 0.842507 | -0.543436 |
| **3** | -0.511425 | 0.475571 | -0.397247 | -0.397247 | -0.477488 |
| -0.310699 | 0.659039 | -0.425834 |
| -0.109974 | 0.842507 | -0.543436 |
| -0.109974 | 0.842507 | -0.543436 |
| **4** | -0.709926 | 0.254742 | -0.450367 | -0.182723 | -0.357932 |
| -0.411062 | 0.567305 | -0.399318 |
| -0.411062 | 0.567305 | -0.399318 |
| -0.328731 | 0.304003 | 0.182723 |
| **5** | -0.328731 | 0.304003 | -0.182723 | -0.110823 | -0.273045 |
| -0.411062 | 0.144023 | -0.110823 |
| -0.411062 | 0.567305 | -0.399318 |
| -0.411062 | 0.567305 | -0.399318 |
| **6** | -0.310571 | 0.144023 | -0.110823 | -0.110823 | -0.136545 |
| -0.319651 | 0.224013 | -0.141811 |
| -0.328731 | 0.304003 | -0.182723 |
| -0.310571 | 0.144023 | 0.110823 |
| **7** | -0.310571 | 0.144023 | -0.110823 | -0.110823 | -0.111762 |
| -0.310571 | 0.144023 | -0.110823 |
| -0.310571 | 0.144023 | -0.110823 |
| -0.115139 | 0.328853 | -0.114579 |
| **8** | -0.310571 | 0.144023 | -0.110823 | -0.110823 | -0.179502 |
| -0.676031 | -0.115709 | -0.385539 |
| -0.310571 | 0.144023 | -0.110823 |
| -0.310571 | 0.144023 | -0.110823 |
| **9** | -0.310571 | 0.144023 | -0.110823 | -0.110823 | -0.150053 |
| -0.310571 | 0.144023 | -0.110823 |
| -0.550314 | -0.064563 | -0.267745 |
| -0.310571 | 0.144023 | -0.110823 |
| **10** | -0.310571 | 0.144023 | -0.110823 | 0.014377 | -0.086711 |
| -0.310571 | 0.144023 | -0.110823 |
| 0.021636 | -0.118378 | 0.014377 |
| -0.310571 | 0.144023 | -0.110823 |

# **4. Листинг**

**Рисунок 2. График зависимости средних значений от номера поколения**

**import** math  
**import** random  
**import** matplotlib.pyplot **as** plt  
**import** numpy  
**import** pylab  
**from** matplotlib **import** cm  
**from** mpl\_toolkits.mplot3d **import** Axes3D  
  
*# Построение трёхмерного графика функции (взято из справочника по Matplotlib)***def** makeData ():  
 x = numpy.arange (-2, 2, 0.1)  
 y = numpy.arange (-2, 2, 0.1)  
 xgrid, ygrid = numpy.meshgrid(x, y)  
 zgrid = - numpy.log(1 + xgrid \* xgrid + ygrid \* ygrid)  
 **return** xgrid, ygrid, zgrid  
x, y, z = makeData()  
  
fig = pylab.figure()  
axes = Axes3D(fig)  
  
axes.plot\_surface(x, y, z, rstride=1, cstride=1, cmap = cm.jet)  
pylab.show()  
  
*# Функция, которую нужно оптимизировать согласно номеру варианта***def** f(p):  
 x, y = p  
 **return** -math.log(1 + x \* x + y \* y)  
  
  
*# Получение значения по первой координате (исходя из данной области)***def** get\_x():  
 **return** -2 + random.random() \* 4  
  
  
*# Получение значения по второй координате***def** get\_y():  
 **return** -2 + random.random() \* 4  
  
n = 1  
  
*# Инициализация популяции***def** init\_population(n):  
 population = []  
  
 *# Заполняем популяцию случайными особями* **for** i **in** range(n):  
 x = get\_x()  
 y = get\_y()  
 population.append((x, y))  
  
 **return** population  
  
  
*# Получение статистики по популяции***def** stat(population):  
 imax = 0  
 max\_f = f(population[0])  
 avg\_f = 0  
  
 **for** i **in** range(len(population)):  
 **if** f(population[i]) > max\_f:  
 imax = i  
 max\_f = f(population[i])  
  
 avg\_f += f(population[i])  
  
 **return** imax, max\_f, avg\_f / len(population) *# Возвраащем индекс максимума, сам максимум и среднее значение  
  
  
# Получение родителя/Селекция (рулеточный отбор)***def** get\_parent(population, n, sum\_f):  
 p = random.random()  
 curr\_f = 0  
  
 **for** i **in** range(n):  
 curr\_f += f(population[i]) / sum\_f *# Вычисляем следующую вероятность* **if** p < curr\_f: *# Если подошло,* **return** i *# возвраащем номер* **return** n - 1 *# Возвращаем последнюю  
  
  
# Функция скрещивания***def** crossingover(population, n):  
 sum\_f = 0  
 *# Считаем сумму* **for** i **in** range(n):  
 sum\_f += f(population[i])  
 *# Получаем родителей* parent1 = get\_parent(population, n, sum\_f)  
 parent2 = get\_parent(population, n, sum\_f)  
 *# Получаем значения родителей* x1, y1 = population[parent1]  
 x2, y2 = population[parent2]  
  
 **return** ((x1 + x2) / 2, (y1 + y2) / 2) *# Итого скрещиваем  
  
  
# Функция мутации***def** mutation(p):  
 x, y = p  
 x += random.random() - 0.5  
 y += random.random() - 0.5  
 **return** x, y  
  
  
n = int(input(**"Enter size of population: "**)) *# Размер популяции (В консоли задаём 4)*parents = n // 2 *# Число родителей*epochs = int(input(**"Enter epochs: "**)) *# Число поколений (Задаём 10)*mutation\_p = 0.25 *# Указанная вероятность мутации (25%)*population = init\_population(n) *# Создаём (инициализируем) начальную популяцию*avg = [] *# Список средних значений  
  
# запускаем алгоритм***for** epoch **in** range(epochs):  
 print(**'gen '** + str(epoch) + **':'**)  
  
 *# Выводим популяцию* **for** p **in** population:  
 print(**' '**, p, **', Fit: '**, f(p))  
  
 population.sort(key=f, reverse=**True**) *# Сортируем по убыванию скор функции* imax, max\_f, avg\_f = stat(population) *# Получаем информацию о популяции* print(**'F max:'**, max\_f, **'for'**, population[imax], **'average:'**, avg\_f) *# Выводим информацию* avg.append(avg\_f) *# Добавляем в список средних значений* new\_population = [] *# Создаём новую популяцию* new\_population.append(population[0]) *# добавляем победителя  
  
 # Добавляем остальных как результат скрещивания* **for** i **in** range(1, n):  
 new\_population.append(crossingover(population, parents))  
  
 *# Внедряем мутацию* **for** i **in** range(n):  
 **if** random.random() < mutation\_p:  
 new\_population[i] = mutation(new\_population[i])  
  
 population = new\_population *# Обновляем популяцию и снова повторяем круг до завершения цикла  
  
# Выводим график средних значений*plt.plot(avg, **'ro'**)  
plt.ylabel(**'Average fit by x'**)  
plt.grid(**True**)  
plt.show()

**Скриншот вывода (не полный) программы:**