МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| профессор |  |  |  | Скобцов Ю. А. |
| должность, звание |  | подпись, дата |  | фамилия, инициалы |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
| «Оптимизация многомерных функций с помощью ГА» |
| по дисциплине: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

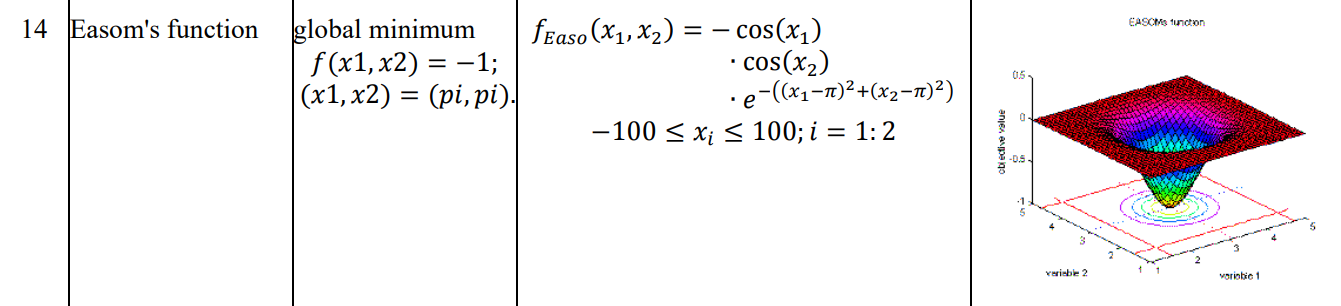
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4936 |  |  |  | Нестеренко М.Ю. |
|  |  |  | подпись, дата |  | фамилия, инициалы |

Санкт-Петербург 2022

**Цель работы**

модификация представления хромосомы и операторов рекомбинации ГА для оптимизации многомерных функций. Графическое отображение результатов оптимизации.

**Вариант задания 14**



**Программа и результат ее выполнения**

Код программы написан на языке программирования Kotlin с применением межпроцессного взаимодействия с Python сценарием с использованием библиотеки matplotlib и numpy.

import java.io.File  
import kotlin.math.\*  
import kotlin.random.Random  
  
const val *VARIANT* = "\n-cos(x1) \* cos(x2) \* e^(-((x1-pi)^2 + (x2-pi)^2))\n"  
const val *IS\_MAX\_SEARCHING* = false  
const val *FROM* = -100.0  
const val *TO* = 100.0  
const val *P\_M* = 0.05  
const val *P\_C* = 0.5  
const val *TOURNAMENT\_COUNT* = 3  
const val *INIT\_POPULATION\_COUNT* = 1000  
const val *MAX\_GENERATION\_COUNT* = 100  
const val *MAX\_REPEATING\_COUNT* = 100  
const val *MAX\_I* = 2  
const val *RESULT\_FILE* = "result.txt"  
  
  
val *population* = *mutableListOf*<MutableList<Double>>()  
var *generation* = 0  
  
fun main() {  
 val begin: Long = System.currentTimeMillis()  
 File(*RESULT\_FILE*).delete()  
 *println*("VARIANT #14:\n$*VARIANT*\n")  
 *population*.*apply* **{** *initPopulation*()  
 *saveGeneration*()  
 *println*("\nextreme fitness before = $*extremeFitness*\n")  
 *evolution*()  
 *println*("\nextreme fitness after = $*extremeFitness*\n")  
 *saveGeneration*()  
 **}** val end: Long = System.currentTimeMillis()  
 *println*("\nTime spent: ${ end - begin } ms")  
  
 Runtime.getRuntime().exec("python3 script.py")  
}  
  
  
fun MutableList<MutableList<Double>>.initPopulation() {  
 *repeat*(*INIT\_POPULATION\_COUNT*) **{** add(*MutableList*(*MAX\_I*) **{** Random.nextDouble(*FROM*, *TO* + 1.0) **}**)  
 **}**}  
  
fun MutableList<MutableList<Double>>.evolution() {  
 var lastFitness = this.*extremeFitness* var fitnessRepeat = 0  
 while (*generation*++ <= *MAX\_GENERATION\_COUNT* && fitnessRepeat <= *MAX\_REPEATING\_COUNT*) {  
 val offspring = *selection*().*apply* **{** *crossing*()  
 *mutation*()  
 **}** *population*.clear()  
 *population*.addAll(offspring)  
 if (*extremeFitness* == lastFitness) fitnessRepeat++  
 else lastFitness = *extremeFitness*.*also* **{** fitnessRepeat = 0 **}** *print*()  
 }  
 *println*("GENERATION COUNT = ${*generation* - 2}")  
 *println*("REPEATING COUNT = $fitnessRepeat")  
}  
  
fun MutableList<MutableList<Double>>.selection(): MutableList<MutableList<Double>> {  
 val newPopulation = *mutableListOf*(this.*extremeIndividual*)  
 *repeat*(*INIT\_POPULATION\_COUNT* - 1) **{** var ies = *List*(*TOURNAMENT\_COUNT*) **{** (0 *until INIT\_POPULATION\_COUNT*).*random*() **}** while (ies.size != ies.*distinct*().size) {  
 ies = *List*(*TOURNAMENT\_COUNT*) **{** (0 *until INIT\_POPULATION\_COUNT*).*random*() **}** }  
 val tournamentIndividuals = *MutableList*(*TOURNAMENT\_COUNT*) **{** this[ies[**it**]] **}** newPopulation.add(tournamentIndividuals.*extremeIndividual*)  
 **}** return newPopulation  
}  
  
fun MutableList<MutableList<Double>>.crossing() {  
 for (i in 0 *until INIT\_POPULATION\_COUNT step* 2) {  
 if (Random.nextDouble() <= *P\_C*) {  
  
 *repeat*(2) **{** val n = Random.nextInt(2, 6)  
 val u = Random.nextDouble()  
 val beta = if (u <= 0.5) (2\*u).*pow*(1 / (n+1)) else (1 / (2\*(1-u))).*pow*(1 / (n+1))  
  
 val h = if (**it** % 2 == 0)  
 *MutableList*(*MAX\_I*) **{** j **->** val cj1 = this[i][j]  
 val cj2 = this[i + 1][j]  
 0.5 \* ((1 - beta) \* cj1 + (1 + beta) \* cj2)  
 **}** else  
 *MutableList*(*MAX\_I*) **{** j **->** val cj1 = this[i][j]  
 val cj2 = this[i + 1][j]  
 0.5 \* ((1 + beta) \* cj1 + (1 - beta) \* cj2)  
 **}** this[i + **it**] = h  
 **}** }  
 }  
}  
  
fun MutableList<MutableList<Double>>.mutation() {  
 *repeat*(*INIT\_POPULATION\_COUNT*) **{** if (Random.nextDouble() <= *P\_M*) {  
 val a = Random.nextInt(0, 2)  
 val k = Random.nextInt(0, *MAX\_I*)  
 val ck = this[**it**][k]  
 val y = *abs*(ck)  
 val r = Random.nextDouble(0.0, 1.0)  
 val t = *generation* val T = *MAX\_GENERATION\_COUNT* val b = 2  
 val delta = y \* (1 - r.*pow*((1.0 - t / T).*pow*(b)))  
 val newChromosome = if (a == 0) ck + delta else ck - delta  
 this[**it**][k] = newChromosome  
 }  
 **}**}  
  
val MutableList<MutableList<Double>>.*extremeFitness* get () = if (*IS\_MAX\_SEARCHING*) *maxFitness* else *minFitness*val MutableList<MutableList<Double>>.*extremeIndividual* get () = if (*IS\_MAX\_SEARCHING*) *maxByOrNull* **{ it**.*fitness* **}**!! else *minByOrNull* **{ it**.*fitness* **}**!!  
val MutableList<MutableList<Double>>.*maxFitness* get () = this.*maxOf* **{ it**.*fitness* **}**val MutableList<MutableList<Double>>.*minFitness* get () = this.*minOf* **{ it**.*fitness* **}**fun MutableList<MutableList<Double>>.print() {  
 val e = *extremeIndividual  
 println*("${e.*fitness*}\t${e.*x1*}\t${e.*x2*}")  
}  
fun MutableList<MutableList<Double>>.saveGeneration() {  
 *forEach* **{** File(*RESULT\_FILE*).*appendText*("${**it**.*x1*}\t${**it**.*x2*}\t${**it**.*fitness*}\n")  
 **}** File(*RESULT\_FILE*).*appendText*("\n")  
}  
  
val MutableList<Double>.*x1*: Double get() = this[0]  
val MutableList<Double>.*x2*: Double get() = this[1]  
val MutableList<Double>.*fitness*: Double  
 get() = -*cos*(*x1*) \* *cos*(*x2*) \* *exp*(-((*x1* - *PI*).*pow*(2) + (*x2* - *PI*).*pow*(2)))

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

def f(x, y):

return -np.cos(x) \* np.cos(y) \* np.exp(-((x - np.pi)\*\*2 + (y - np.pi)\*\*2))

x = np.linspace(1, 5, 25)

y = np.linspace(1, 5, 25)

X, Y = np.meshgrid(x, y)

Z = f(X, Y)

xdata = []

ydata = []

zdata = []

with open('result.txt') as f:

lines = f.readlines()

x = []

y = []

z = []

for line in lines:

if line == "\n":

xdata.append(x)

ydata.append(y)

zdata.append(z)

x = []

y = []

z = []

continue

line = line.split('\t')

point = [i.strip().replace(',', '.') for i in line]

x.append(float(point[0]))

y.append(float(point[1]))

z.append(float(point[2]))

fig = plt.figure()

titles = ['Init population', 'Last population']

for i in range(0,2):

ax = fig.add\_subplot(1, 2, i+1, projection='3d')

ax.set\_xlabel('x')

ax.set\_ylabel('y')

ax.set\_zlabel('z')

ax.plot\_surface(X, Y, Z, cmap='coolwarm', alpha=0.5)

ax.scatter3D(xdata[i], ydata[i], zdata[i], s=50, c='#ff0000')

ax.set\_title(titles[i])

ax.set\_xlim(1, 5)

ax.set\_ylim(1, 5)

ax.set\_zlim(-1.5, 0.5)

plt.show()

**Анализ работы программы**

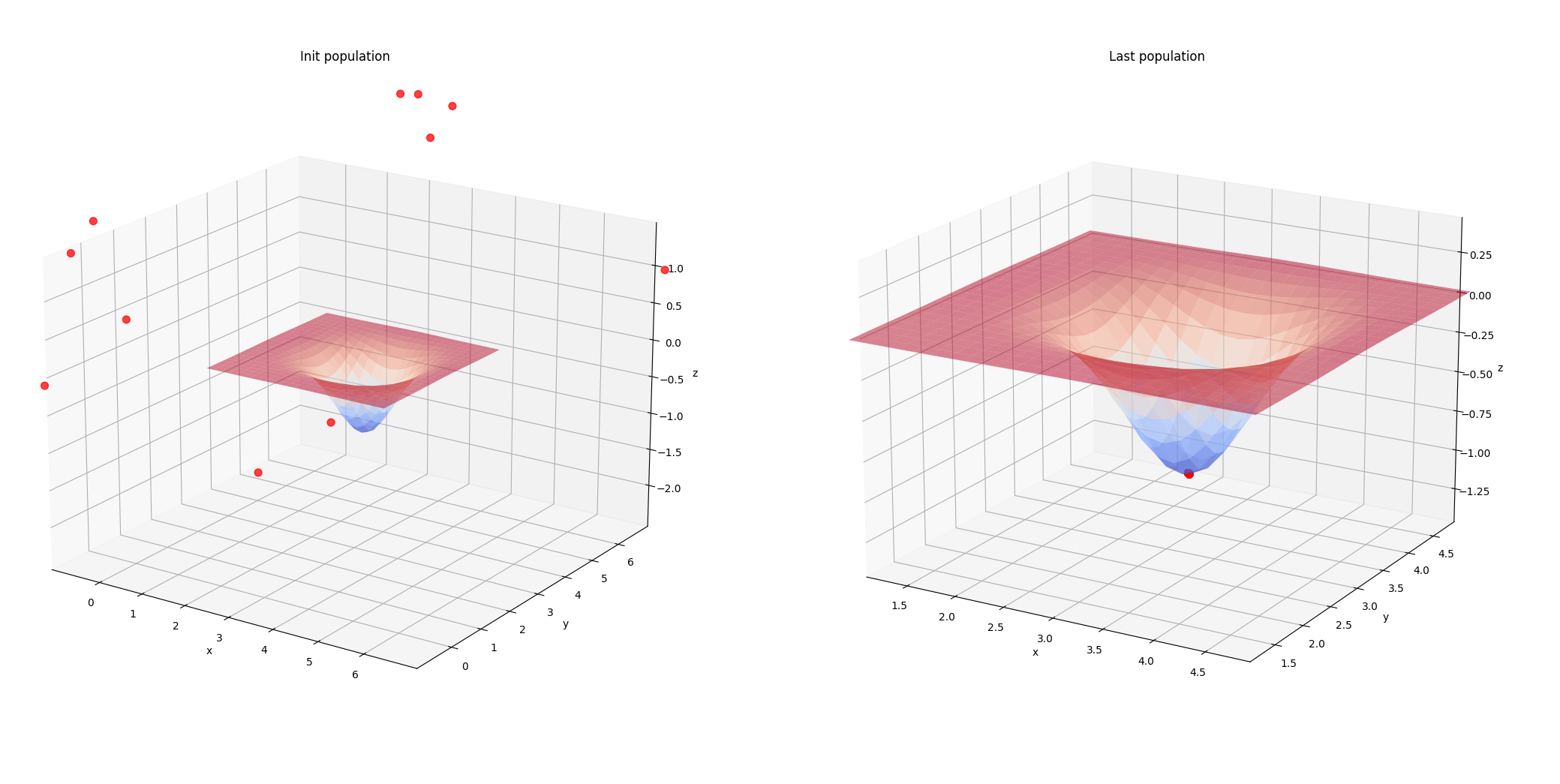
Для n = 2.

Pc = 0.5

Pm = 0.05

MAX\_GENERATIONS = 100

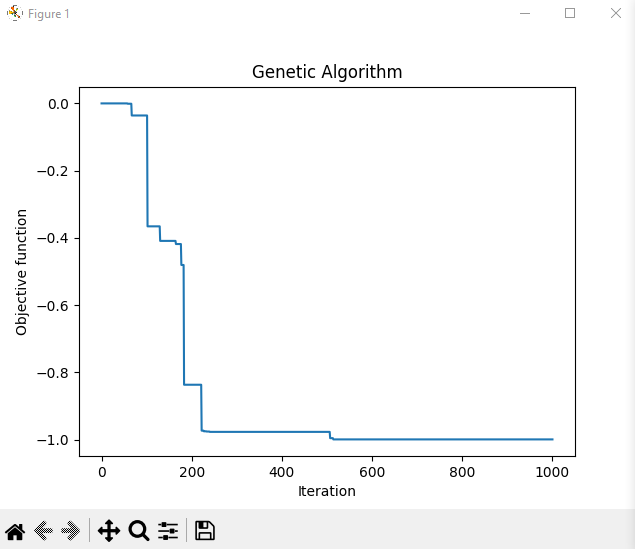
POPULATION = 1000

Значение целевой функции после выполнения алгоритмы: -0.9995586127872885

Времени затрачено: 530 мс

**Сравнение с встроенными библиотеками**

Для n = 2. Python Genetic Algorithm



Значение целевой функции после выполнения алгоритмы: -0.9992948869581629

Времени затрачено: ~20 000 мс

**Исследование зависимостей**

Исследуем параметры (время выполнения, полученные решения и необходимое количество поколений) работы алгоритма в зависимости от Pc, Pm и N:

1. Зависимость от изменений параметра Pc
2. Зависимость от изменений параметра Pm
3. Зависимость от изменений особей в популяции

**Контрольный вопрос**

Приведите основные понятия и определения генетических алгоритмов.

**Генетический алгоритм** – это алгоритм поиска для решения задач оптимизации путем случайного подбора, комбинирования и вариации искомых параметров, аналогично естественному отбору в природе;

**Ген** – элементарная единица в хромосоме, определяющая один из его параметров;

**Хромосома** – упорядоченная совокупность всех генов;

**Особь** – одно потенциальное решение проблемы, которая представляется хромосомой;

**Популяция** – множество особей(решений);

**Поколение** – текущая или n-ная популяция в работе генетического алгоритма;

**Генотип** – набор хромосом одной из особей;

**Генофонд** – совокупность всех генотипов;

**Мощность** **популяции** – число особей, в популяции;

**Число поколений** – количество итераций, которые алгоритм произвел в процессе поисках решения;

**Фитнесс-функция** – целевая математическая функция, которая определяет ценность отдельной особи, показывает меру приспособления особи;

**Отбор** – процесс, в котором из текущего поколения отбираются более приспособленные особи для передачи своих генов следующему поколению;

**Скрещивание** – процесс, в котором одна из особей поколения случайным образом комбинирует свои гены с другой особью поколения для получения новых решений;

**Мутация** – определенный процесс, в котором один из генов какой-либо особи с некоторой вероятность может измениться, независимо от текущего генофонда. Необходимы для избежания застаивания и получения новых решений;

**Pc** – вероятность скрещивания, как правило, равняется 0,5;

**Pm** – вероятность мутации гена, имеет низкое значение (~0.001), так как оказывает на генофонд воздействие “извне”.

**Вывод**

В ходе выполнения лабораторной работы были получены основные навыки реализации и оптимизации с многомерных математических функций с помощью генетических алгоритмов на высокоуровневых языках программирования.