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 №7 |
| «Оптимизация функций с помощью роевых алгоритмов» |
| по дисциплине: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

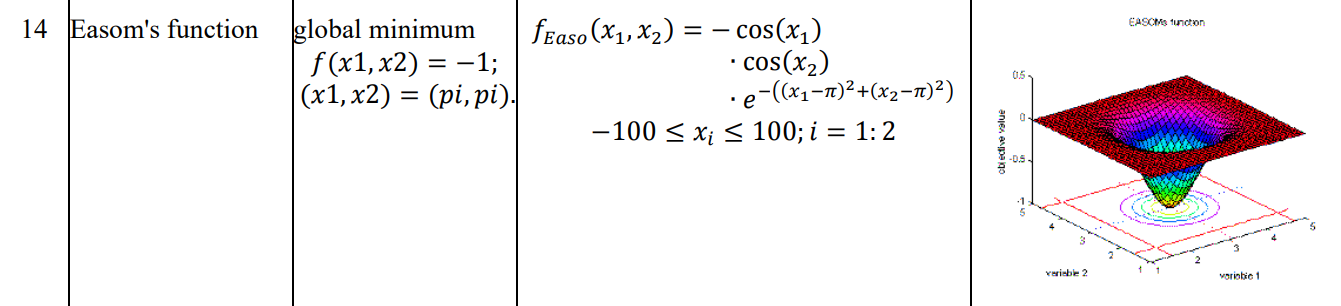
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4936 |  |  |  | Нестеренко М.Ю. |
|  |  |  | подпись, дата |  | фамилия, инициалы |

Санкт-Петербург 2022

**Цель работы**

оптимизация функций многих переменных методом роевого интеллекта. Графическое отображение результатов оптимизации

**Вариант задания 14**



Требуется выполнить следующие задачи:

1. Создать программу, использующую РА для нахождения оптимума функции согласно таблице вариантов, приведённой в Приложении А. Для всех Benchmark-ов оптимумом является минимум. Программу выполнить на встроенном языке пакета Matlab (или любом, доступным вам, языке программирования);
2. Для n=2 вывести на экран график данной функции с указанием найденного экстремума, точек популяции. Для вывода графиков использовать стандартные возможности пакета Matlab. Предусмотреть возможность пошагового просмотра процесса поиска решения.
3. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

• Число особей в популяции;

• Вероятность мутации.

Критерий остановки вычислений – повторение лучшего результата заданное количество раз или достижение популяцией определенного возраста (например, 100 эпох).

1. Повторить процесс поиска решения для n=3, n=5, n=10, сравнить результаты, скорость работы программы.

**Код программы**

Код программы, написанный на языке программирования Kotlin и выводам графика на языке Python с использованием matplotlib:

**Main.kt**

fun main() {  
 GA().run()  
}

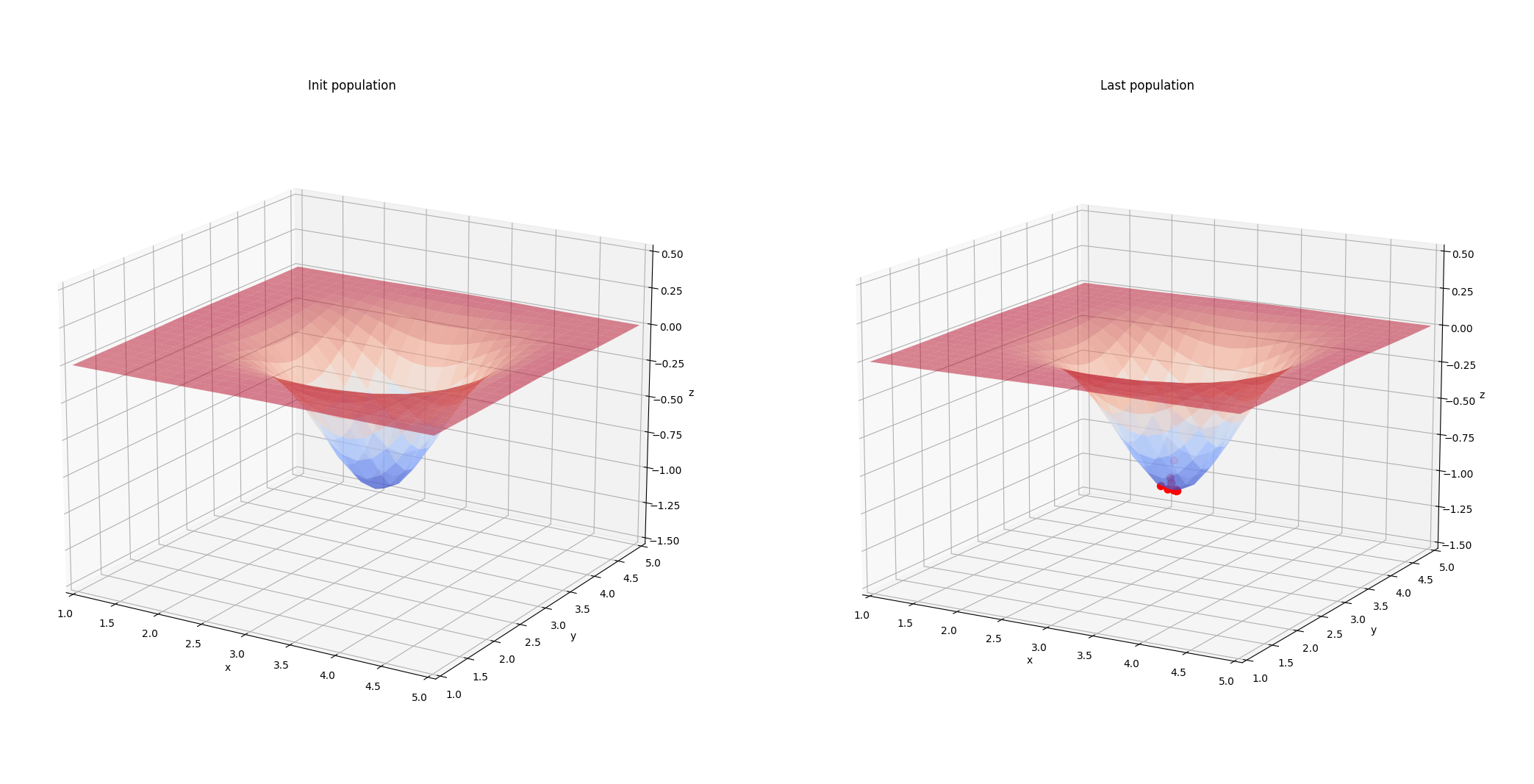
**GA.kt**

import java.io.File  
import kotlin.random.Random  
  
const val *VARIANT* = "#14:\n-cos(x1) \* cos(x2) \* e^(-((x1-pi)^2 + (x2-pi)^2))\n"  
const val *FROM* = -100.0  
const val *TO* = 100.0  
const val *INIT\_POPULATION\_SIZE* = 100  
const val *MAX\_GENERATION\_COUNT* = 200  
const val *RESULT\_FILE* = "result.txt"  
  
class GA {  
  
 private val particles = *MutableList*(*INIT\_POPULATION\_SIZE*) **{** Particle(  
 position = Position(  
 x1 = Random.nextDouble(*FROM*, *TO*),  
 x2 = Random.nextDouble(*FROM*, *TO*)  
 ),  
 speed = *mutableMapOf*("x1" *to* Random.nextDouble(*FROM*, *TO*), "x2" *to* Random.nextDouble(*FROM*, *TO*))  
 )  
 **}** private var globalBest = Position(particles.*best*.x1, particles.*best*.x2)  
  
 private val MutableList<Particle>.*best* get() = *minBy* **{ it**.localBest.fitness **}** private fun correctPosition() {  
 particles.*forEach* **{  
 it**.correctPosition()  
 **}** }  
  
 private fun correctSpeed() {  
 particles.*forEach* **{  
 it**.correctSpeed(globalBest)  
 **}** }  
  
 private fun calcGlobal() {  
 val b = particles.*best* if (globalBest.fitness > b.localBest.fitness) globalBest = b.position.copy()  
 }  
  
 fun run() {  
 *println*(*VARIANT*)  
 File(*RESULT\_FILE*).delete()  
 particles.*savePopulation*()  
  
 val begin = System.currentTimeMillis()  
 particles.*apply* **{** *repeat*(*MAX\_GENERATION\_COUNT*) **{** correctPosition()  
 correctSpeed()  
 calcGlobal()  
 *println*("global = $globalBest \t fitness = ${globalBest.fitness}")  
 **}  
 }** val end = System.currentTimeMillis()  
 *println*("\nTime spent: ${end - begin} ms")  
  
 particles.*savePopulation*()  
 Runtime.getRuntime().exec("python3 script.py")  
 }  
  
 private fun MutableList<Particle>.savePopulation() {  
 *forEach* **{** File(*RESULT\_FILE*).*appendText*("${**it**.localBest.x1}\t${**it**.localBest.x2}\t${**it**.localBest.fitness}\n")  
 **}** File(*RESULT\_FILE*).*appendText*("\n")  
 }  
}

**Particle.kt**  
import java.util.\*  
import kotlin.math.\*  
  
data class Particle(  
 var position: Position,  
 var speed: MutableMap<String, Double>  
) {  
  
 companion object {  
 private const val c1 = 2.0 *// в локально лучшую* private const val c2 = 5.0 *// в глобально лучшую* private val COMMON\_SPEED = (2.0 \* 0.3 / *abs*(2.0 - (c1 + c2) - *sqrt*((c1 + c2).*pow*(2) - 4.0 \* (c1 + c2))))  
 }  
  
 val x1 get() = position.x1  
 val x2 get() = position.x2  
  
 private val random = Random()  
  
 var localBest = position.copy()  
 private set  
  
 fun correctPosition() {  
 position.x1 += speed["x1"]!!  
 position.x2 += speed["x2"]!!  
  
 if (x1 !in (*FROM*..*TO*)) position.x1 -= speed["x1"]!!  
 if (x2 !in (*FROM*..*TO*)) position.x2 -= speed["x2"]!!  
  
 if (position.fitness <= localBest.fitness) localBest = position.copy()  
 }  
  
 fun correctSpeed(globalBest: Position) {  
 val r1 = random.nextGaussian()  
 val r2 = random.nextGaussian()  
  
 speed.keys.*forEach* **{** speed[**it**] =  
 (speed[**it**]!! \* COMMON\_SPEED) +  
 (c1\*r1 \* (localBest[**it**] - position[**it**]) \* COMMON\_SPEED) +  
 (c2\*r2 \* (globalBest[**it**] - position[**it**]) \* COMMON\_SPEED)  
 **}** }  
}  
  
data class Position(var x1: Double, var x2: Double) {  
 operator fun get(x: String) = map[x]!!  
 override fun toString() = "($x1, $x2)"  
 private val map get() = *mapOf*("x1" *to* x1, "x2" *to* x2)  
 val fitness get() = -*cos*(x1) \* *cos*(x2) \* *exp*(-((x1 - *PI*).*pow*(2) + (x2 - *PI*).*pow*(2)))  
}

**script.py**  
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()

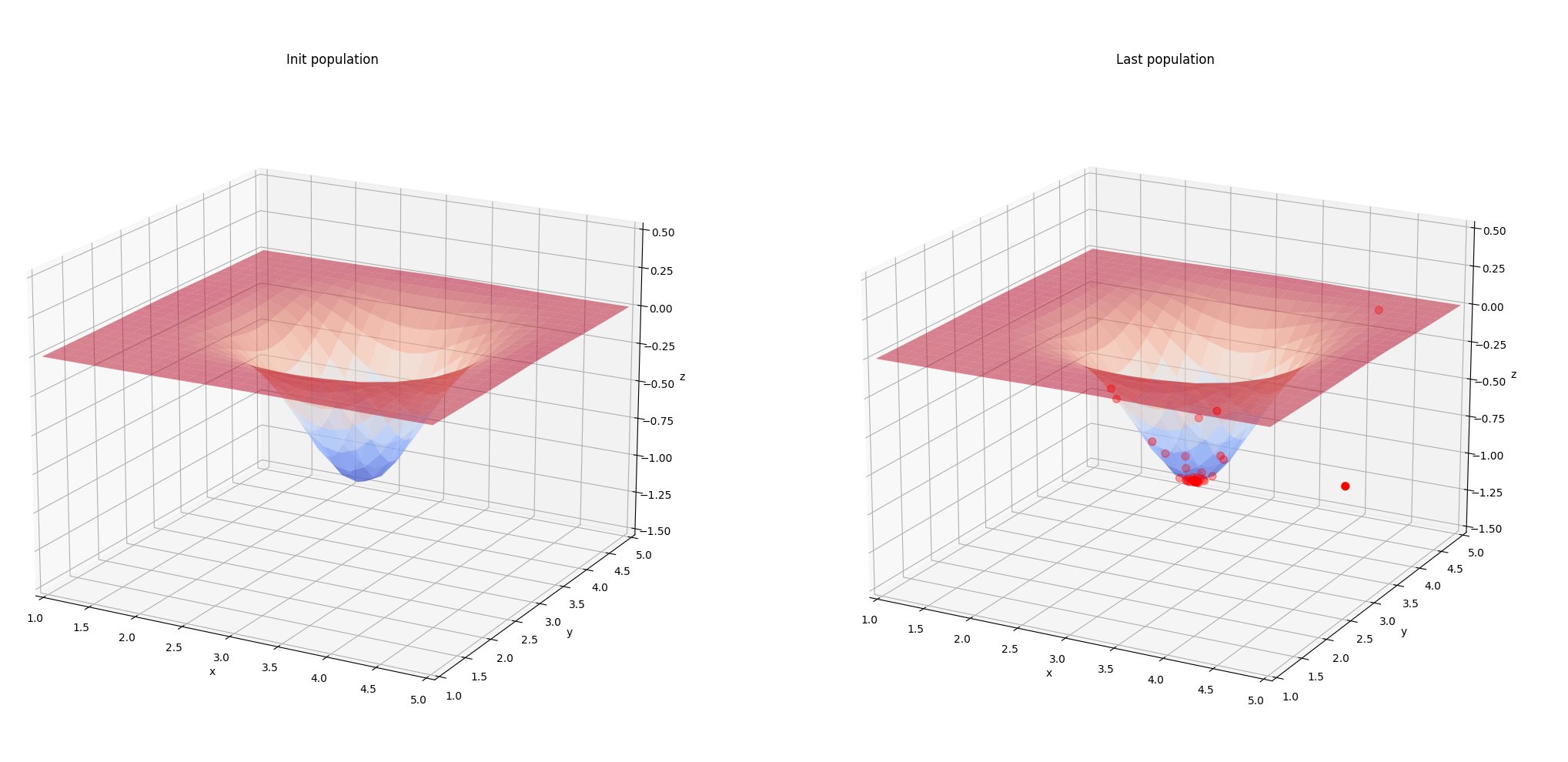
**Результат выполнения**

****

|  |  |  |  |
| --- | --- | --- | --- |
| Nпок | Nпоп | A­лок | A­глоб |
| 100 | 200 | 2 | 5 |

Значение фитнесс функции: -0.9999999999999997

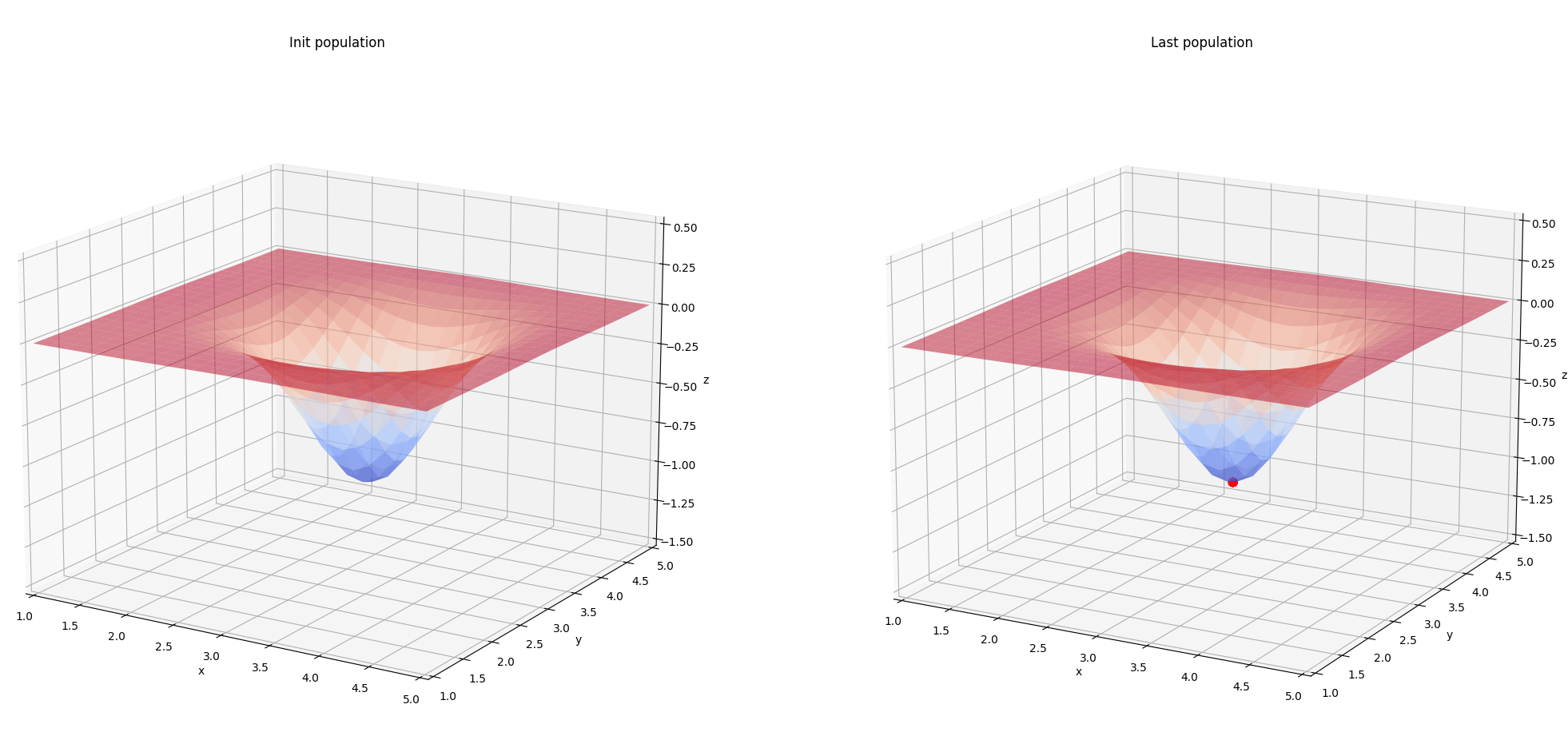
Время выполнения: 77 мс



|  |  |  |  |
| --- | --- | --- | --- |
| Nпок | Nпоп | A­лок | A­глоб |
| 100 | 100 | 2 | 2 |

Значение фитнесс функции: -0.9999999999926975

Время выполнения: 54 мс



|  |  |  |  |
| --- | --- | --- | --- |
| Nпок | Nпоп | A­лок | A­глоб |
| 100 | 500 | 3 | 1 |

Значение фитнесс функции: -1.0

Время выполнения: 154 мс

**Зависимость работы ГА от параметров**

При уменьшении максимального количества поколений (Nпок) точность алгоритма экспоненциально падает. Это объясняется тем, что алгоритм просто не успевает найти более точное решение, так как ограничен малым числом итераций. Оптимальным количеством поколений является интервал [50; 100], который дает лучшее соотношение точности и скорости.

При увеличении размера популяции (N) точность алгоритма экспоненциально растет. Так как изначальная большая выборка позволяет выбрать наиболее удачные решения и модифицировать их. Точность алгоритма заметно падает при популяции меньше 50.

При изменении стремления частицы роя к глобальному оптимуму точность алгоритма начинает уменьшаться при уменьшении стремления. Так как частица начинает в большей степени ориентироваться только на свои показатели, игнорирую информацию других частиц роя. Как можно заметить, при значениях Аглоб близких к 1 и меньше алгоритм не смог найти решения, что объясняется тем, что частицы почти полностью игнорируют остальной рой.

**Сравнение с другими алгоритмами оптимизации многомерных функций**

В сравнении с эволюционной стратегией роевой алгоритм имеет большое преимущество, так как позволяет каждой особи ориентироваться не только на собственные показатели, но и следить за текущим состоянием роя, что позволяет гораздо быстрее находить точку оптимума.

Роевой алгоритм позволяет уменьшать популяцию и количество поколений до малых значений, что позволяет получить большую производительность в сравнении с классическим генетическим алгоритмом и эволюционной стратегией.

|  |  |
| --- | --- |
| Роевой алгоритм | Эволюционная стратегия |
|  |  |
|  |  |

**Вывод**

В ходе выполнения лабораторной работы были получены основные навыки разработки и использования роевого алгоритма и его сравнения другими подобными алгоритмами на примере задачи нахождения оптимума многомерных математических функций.