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 №5 |
| «Оптимизация многомерных функций с помощью эволюционной стратегии» |
| по дисциплине: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

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

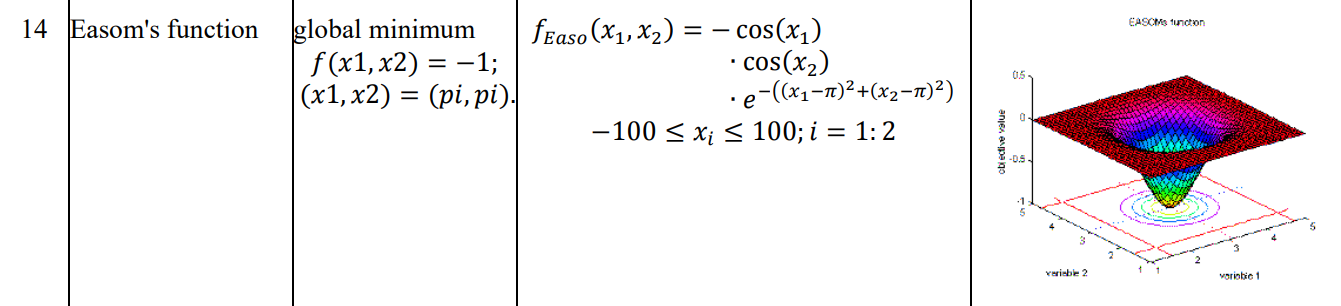
Санкт-Петербург 2022

**Цель работы**

оптимизация функций многих переменных модификация методом эволюционной стратегии. Графическое отображение результатов оптимизации

**Вариант задания 14**

Вид представления: Древовидное



**Теоретические сведения**

Эволюционные стратегии (ЭС), также как и предыдущие парадигмы, основаны на эволюции популяции потенциальных решений, но, в отличие от них, здесь используется генетические операторы на уровне фенотипа, а не генотипа, как это делается в ГА. Разница в том, что ГА работают в пространстве генотипа – кодов решений, в то время как ЭС производят поиск в пространстве фенотипа – векторном пространстве вещественных чисел. В ЭС учитываются свойства хромосомы «в целом», в отличие от ГА, где при поиске решений исследуются отдельные гены. В природе один ген может одновременно влиять на несколько свойств организма. С другой стороны, одно свойство особи может определяться несколькими генами. Естественная эволюция основана на исследовании совокупности генов, а не отдельного (изолированного) гена.

В эволюционных стратегиях целью является движение особей популяции по направлению к лучшей области ландшафта фитнесс-функции. ЭС изначально разработаны для решения многомерных оптимизационных задач, где пространство поиска – многомерное пространство вещественных чисел [1]. Иногда при решении задачи накладываются некоторые ограничения, например, вида gi(x)>0.

Ранние эволюционные стратегии (ЭС) основывались на популяции, состоящей из одной особи, и в них использовался только один генетический оператор – мутация. Здесь для представления особи (потенциального решения) была использована идея, не представленная в классическом генетическом алгоритме, которая заключается в следующем.

Здесь особь представляется парой действительных векторов



где x - точка в пространстве решений и σ - вектор стандартных отклонений (вариабельность) от решения. В общем случае особь популяции определяется вектором потенциального решения и вектором «стратегических параметров» эволюции. Обычно это вектор стандартных отклонений (дисперсия), хотя допускаются (и иногда используются) и другие статистики. Единственным генетическим оператором в классической ЭС [1] является оператор мутации, который выполняется путем сложения координат вектора-родителя со случайными числами, подчиняющимися закону нормального распределения, следующим образом:



где N(0,σ ) - вектор независимых случайных чисел, генерируемых согласно распределению Гаусса (например, табличным способом) с нулевым средним значением и стандартным отклонением σ. Как видно из приведенной формулы величина мутации управляется нетрадиционным способом. Иногда эволюционный процесс используется для изменения и самих стратегических параметров σ, в этом случае величина мутации эволюционирует вместе с искомым потенциальным решением. Это соответствует адаптивному ГА с изменяемым шагом мутации.

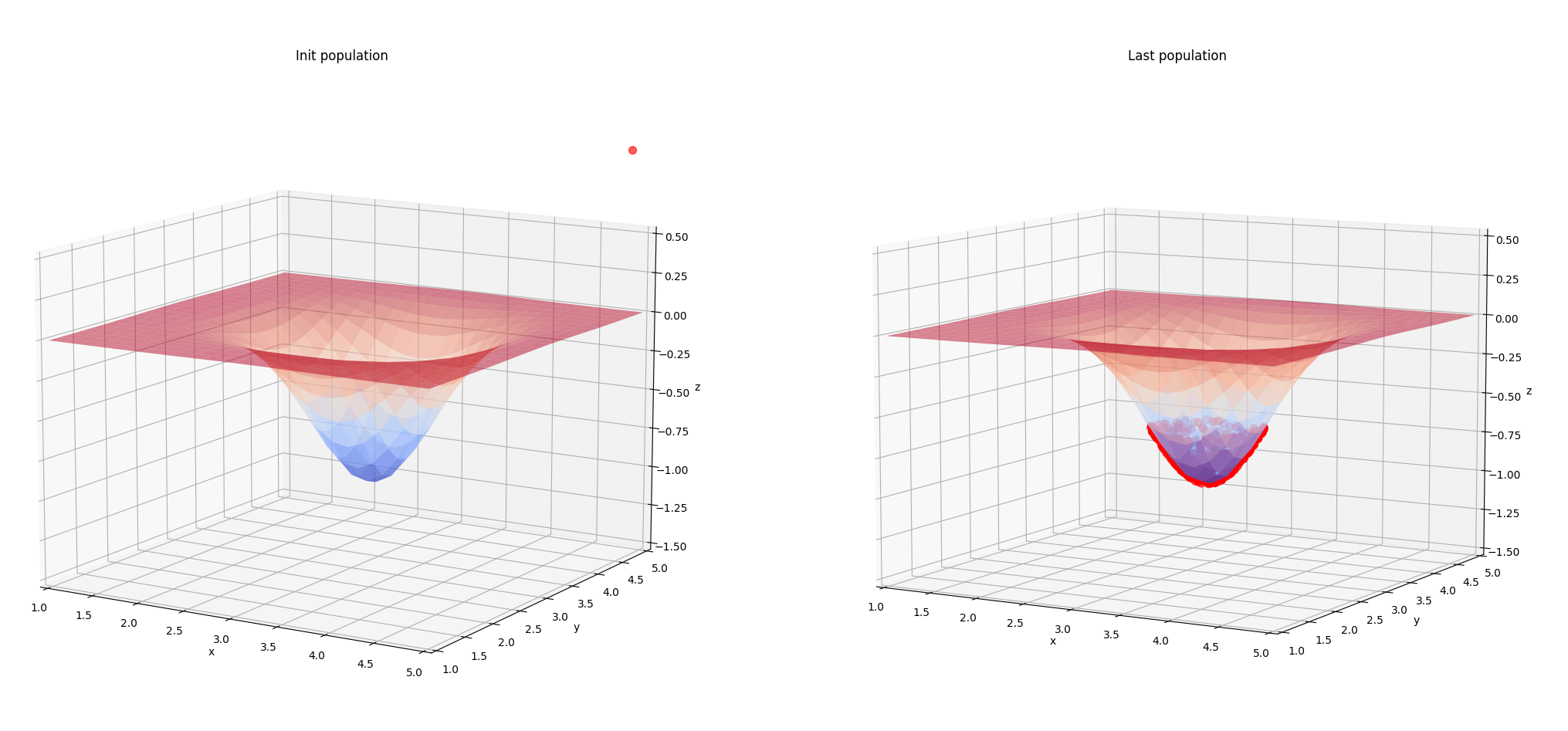
**Код программы**

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

**Main.kt**  
import java.io.File  
import kotlin.math.\*  
import kotlin.random.Random  
  
const val *VARIANT* = "#14:\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 *DEFAULT\_DEVIANCE* = 1.0  
const val *INIT\_POPULATION\_SIZE* = 100  
const val *MAX\_GENERATION\_COUNT* = 100  
const val *RESULT\_FILE* = "result.txt"  
  
  
fun main() {  
 File(*RESULT\_FILE*).delete()  
 *println*(*VARIANT*)  
  
 val population = *mutableListOf*<Individual>()  
 val begin = System.currentTimeMillis()  
 population.*apply* **{** *initPopulation*()  
 *savePopulation*()  
  
 *println*("\nextreme fitness before = $*extremeFitness*\n")  
 *evolution*()  
 *println*("\nextreme fitness after = $*extremeFitness*\n")  
 *savePopulation*()  
 **}** val end = System.currentTimeMillis()  
 *println*("\nTime spent: ${end - begin} ms")  
  
 *//Runtime.getRuntime().exec("python3 script.py")*}  
  
  
fun MutableList<Individual>.initPopulation() {  
 *repeat*(*INIT\_POPULATION\_SIZE*) **{** add(  
 Individual(  
 Random.nextDouble(*FROM*, *TO*),  
 Random.nextDouble(*FROM*, *TO*),  
 *DEFAULT\_DEVIANCE* )  
 )  
 **}**}  
  
fun MutableList<Individual>.evolution() {  
 var counter = 0  
 while (++counter < *MAX\_GENERATION\_COUNT*) {  
 *mutation*()  
 *selection*()  
 *printBest*()  
 }  
 *println*("GENERATION COUNT = $counter")  
}  
  
fun MutableList<Individual>.selection() {  
 *sortByDescending* **{ it**.mutations **}** if (*IS\_MAX\_SEARCHING*)  
 *sortByDescending* **{ it**.fitness **}** else  
 *sortBy* **{ it**.fitness **}** val newPopulation = subList(0, *INIT\_POPULATION\_SIZE*).*toList*()  
 clear()  
 addAll(newPopulation)  
}  
  
fun MutableList<Individual>.mutation() {  
 *repeat*(*INIT\_POPULATION\_SIZE*) **{** add(get(**it**).mutate())  
 **}**}  
  
val MutableList<Individual>.*extremeFitness* get() = if (*IS\_MAX\_SEARCHING*) *maxFitness* else *minFitness*val MutableList<Individual>.*extremeIndividual* get() = if (*IS\_MAX\_SEARCHING*) *maxBy* **{ it**.fitness **}** else *minBy* **{ it**.fitness **}**val MutableList<Individual>.*maxFitness* get() = this.*maxOf* **{ it**.fitness **}**val MutableList<Individual>.*minFitness* get() = this.*minOf* **{ it**.fitness **}**fun MutableList<Individual>.printBest() {  
 *with*(*extremeIndividual*) **{** *println*("${fitness}\t${x1}\t${x2}") **}**}  
  
fun MutableList<Individual>.savePopulation() {  
 *forEach* **{** File(*RESULT\_FILE*).*appendText*("${**it**.x1}\t${**it**.x2}\t${**it**.fitness}\n")  
 **}** File(*RESULT\_FILE*).*appendText*("\n")  
}  
  
data class Individual(  
 val x1: Double,  
 val x2: Double,  
 val param: Double,  
 private val successMutation: Int = 0,  
 val mutations: Int = 0  
) {  
  
 companion object {  
 private const val K\_SUCCESS = 0.2  
 private const val MEAN = 0.0  
 private const val C\_I = 1.22  
 private const val C\_D = 0.82  
 }  
  
 private val rand = java.util.Random()  
 private val fi get() = successMutation.toDouble() / mutations  
 val fitness get() = -*cos*(x1) \* *cos*(x2) \* *exp*(-((x1 - *PI*).*pow*(2) + (x2 - *PI*).*pow*(2)))  
 private fun childFitness(x1: Double, x2: Double) =  
 -*cos*(x1) \* *cos*(x2) \* *exp*(-((x1 - *PI*).*pow*(2) + (x2 - *PI*).*pow*(2)))  
  
 fun mutate(): Individual {  
 val parentFitness = fitness  
 val x1Child = x1 + rand.nextGaussian(MEAN, param)  
 val x2Child = x2 + rand.nextGaussian(MEAN, param)  
 return Individual(  
 x1 = x1Child,  
 x2 = x2Child,  
 param =  
 if (fi < K\_SUCCESS) param \* C\_D  
 else if (fi > K\_SUCCESS) param \* C\_I  
 else param,  
 successMutation =  
 if (childFitness(x1Child, x2Child) < parentFitness) successMutation + 1  
 else successMutation,  
 mutations = mutations + 1  
 )  
 }  
}

**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()

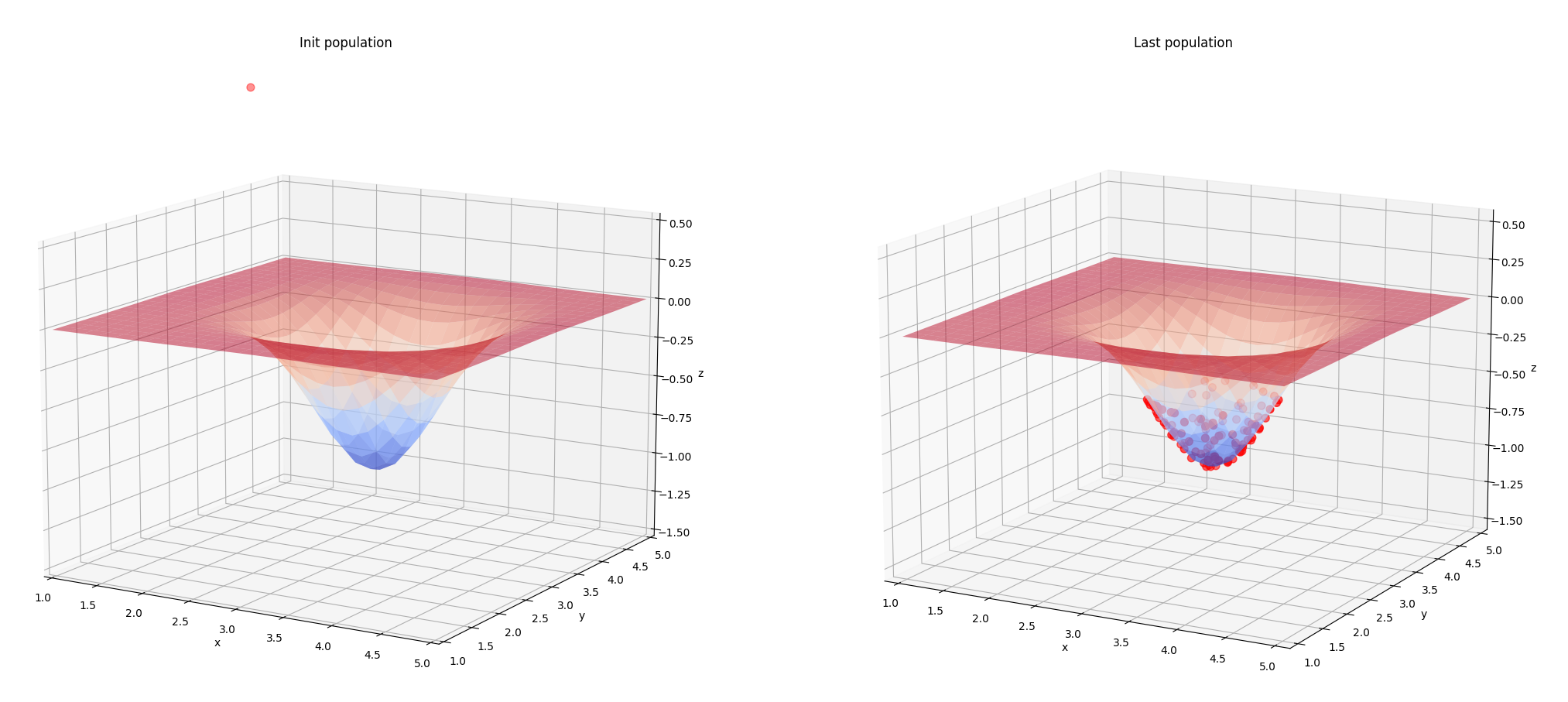
**Результат выполнения**

****

|  |  |  |
| --- | --- | --- |
| Nпок | Nпоп | σнач |
| 100 | 500 | 1 |

Значение фитнесс функции: -0.99879

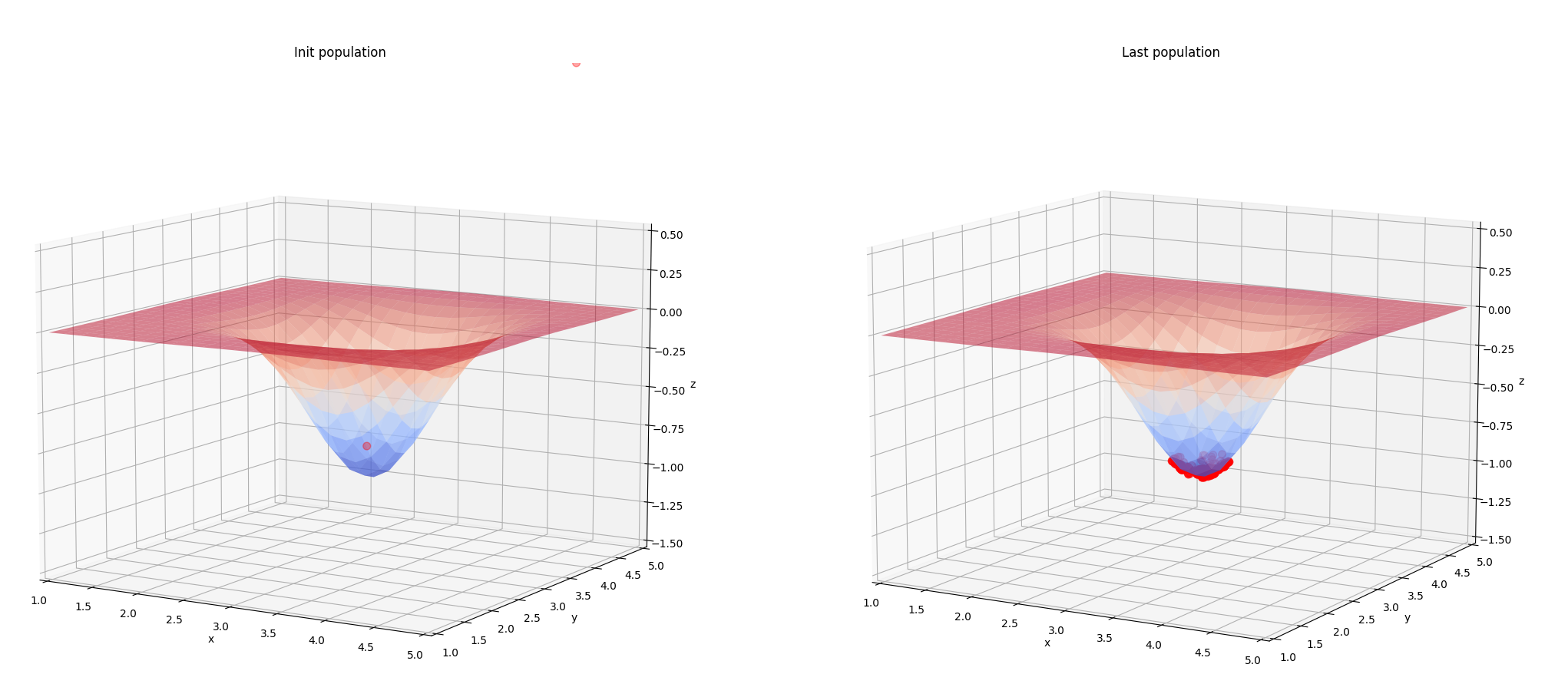
Время выполнения: 347 мс



|  |  |  |
| --- | --- | --- |
| Nпок | Nпоп | σнач |
| 100 | 100 | 1 |

Значение фитнесс функции: -0.9989

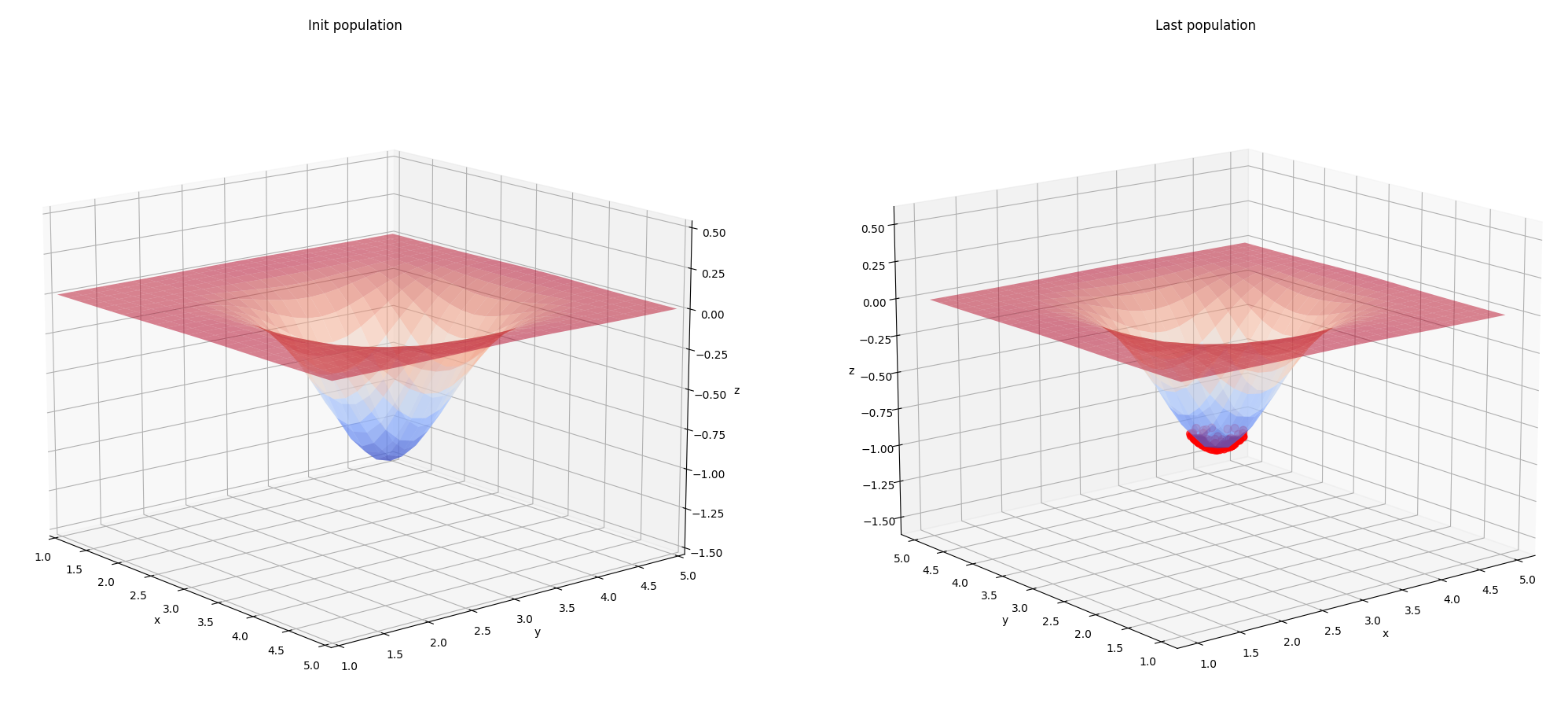
Время выполнения: 132 мс



|  |  |  |
| --- | --- | --- |
| Nпок | Nпоп | σнач |
| 100 | 100 | 5 |

Значение фитнесс функции: -0.99936

Время выполнения: 107 мс



|  |  |  |
| --- | --- | --- |
| Nпок | Nпоп | σнач |
| 500 | 100 | 1 |

Значение фитнесс функции: -0.9996

Время выполнения: 216 мс

**Зависимость работы ГА от параметров**

При очень больших значениях отклонения нормального распределения (σнач) эволюционная стратегия резко увеличивает погрешность вычислений, так как алгоритм делает слишком большие шаги как по правильному, так и по неверному направлению от нужной точки, тем самым теряя потенциальные решения. Оптимальными значениями являются 2.5 – 0.25.

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.

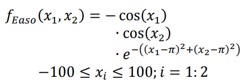
При увеличении размера популяции (N) точность алгоритма экспоненциально растет. Так как изначальная большая выборка позволяет выбрать наиболее удачные решения и модифицировать их.

**Сравнение с обычным эволюционным алгоритмом**

В отличие от классического эволюционного алгоритма эволюционная стратегия позволяет более точно и быстро рассчитывать нужные результаты, так как может направлять развитие каждой особи индивидуально.

ЭС позволяет сильно сократить выборки и максимальное количество итераций алгоритма, что существенно может снизить нагрузка на вычислительную машину.

Однако, при достаточно малой выборке (25 и менее) и большом доступном отрезке (например [-100; 100] могут случаться выбросы (неудачные расчеты), так как первоначальная выборка была далека от искомого результата и не смогла направить развитие в нужное направление.

Например, , при больших значениях x1 и x2 выдает бесконечно малые (одинаковые) значения целевой функции, чем запутывает стратегию и не дает найти решение.

Решается увеличением изначальной выборки.

**Контрольный вопрос**

Какой генетический оператор применяется в ЭС?

В ЭС на прямую применяется только один генетический оператор – оператор мутации. В ЭС мутация выполняется посредством сложения переменных родителя и случайной величины, распределенной по нормальному закону.



Поскольку ЭС имеет сильные различия от обычных эволюционных алгоритмов, то операторы кроссинговера и отбора либо отсутствуют, либо принимают иные формы, не похожие на классические представления.

**Вывод**

В ходе выполнения лабораторной работы были получены основные навыки разработки и алгоритма с эволюционной стратегией и сравнение с результатами обычного эволюционного алгоритма.