ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| д-р техн. наук, профессор |  |  |  | Ю.А. Скобцов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7 |
| Роевые алгоритмы |
| по курсу: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ |
|  |
|  |

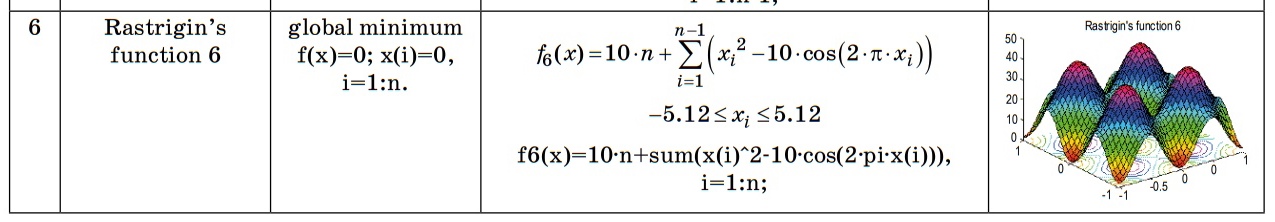
РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4136 |  |  |  | Якшин С.Е. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**Индивидуальное задание по варианту:**

Вариант 22(6)



**Цель работы:**

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

**Теоретические сведения:**

Роевой алгоритм (РА) использует рой частиц, где каждая частица представляет потенциальное решение проблемы.

Поведение частицы в гиперпространстве поиска решения все время подстраивается в соответствии со своим опытом и опытом своих соседей.

Кроме этого, каждая частица помнит свою лучшую позицию с достигнутым локальным лучшим значением целевой (фитнесс-) функции и знает наилучшую позицию частиц - своих соседей, где достигнут глобальный на текущий момент оптимум.

В процессе поиска частицы роя обмениваются информацией о достигнутых лучших результатах и изменяют свои позиции и скорости по определенным правилам на основе имеющейся на текущий момент информации о локальных и глобальных достижениях.

При этом глобальный лучший результат известен всем частицам и немедленно корректируется в том случае, когда некоторая частица роя находит лучшую позицию с результатом, превосходящим текущий глобальный оптимум.

Каждая частица сохраняет значения координат своей траектории с соответствующими лучшими значениями целевой функции, которые обозначим yi, которая отражает когнитивную компоненту.

**Код программы:**

#ifndef LAB7\_HPP

#define LAB7\_HPP

#include <iostream>

#include <vector>

#include <cmath>

#include <random>

#include <algorithm>

#include "Bocchiemark.hpp"

namespace lab7

{

class Swarm;

class Bug;

// Пример функции для минимизации

namespace k

{

uint16\_t N = 100;

const float MIN = -5.12, MAX = 5.12;

const float VEL\_MIN = -1.0; // Минимальная скорость

const float VEL\_MAX = 1.0; // Максимальная скорость

const float C1 = 2.0; // Коэффициент индивидуального ускорения

const float C2 = 2.0; // Коэффициент социального ускорения

const float INERTIA = 0.5; // Инерционная компонента

const size\_t MAX\_ITERATIONS = 100; // Максимальное количество итераций

inline float f(const std::vector<float> &x);

inline std::vector<float> f(const Swarm &x);

} // namespace k

// Класс частицы

class Bug {

public:

std::vector<float> position; // Текущая позиция

std::vector<float> velocity; // Текущая скорость

std::vector<float> best\_position; // Лучшая найденная позиция

float best\_value; // Значение функции в лучшей позиции

Bug(size\_t dimensions, float pos\_min, float pos\_max, float vel\_min, float vel\_max) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> pos\_dist(pos\_min, pos\_max);

std::uniform\_real\_distribution<> vel\_dist(vel\_min, vel\_max);

for (size\_t i = 0; i < dimensions; ++i) {

position.push\_back(pos\_dist(gen));

velocity.push\_back(vel\_dist(gen));

}

best\_position = position;

best\_value = k::f(position); // Инициализируем значение фитнесс-функции

}

};

// Класс роя

class Swarm {

public:

std::vector<Bug> bugs;

std::vector<float> global\_best\_position;

float global\_best\_value;

float c1, c2; // Коэффициенты ускорения

float inertia; // Инерционная компонента

size\_t dimensions; // Размерность задачи

size\_t max\_iterations;

Swarm(size\_t swarm\_size, size\_t dimensions, float pos\_min, float pos\_max,

float vel\_min, float vel\_max, float c1, float c2, float inertia, size\_t max\_iterations)

: c1(c1), c2(c2), inertia(inertia), dimensions(dimensions), max\_iterations(max\_iterations) {

// Инициализируем частицы

for (size\_t i = 0; i < swarm\_size; ++i) {

bugs.emplace\_back(dimensions, pos\_min, pos\_max, vel\_min, vel\_max);

}

// Инициализируем глобальное лучшее значение

global\_best\_position = bugs[0].best\_position;

global\_best\_value = bugs[0].best\_value;

for (const auto& bug : bugs) {

if (bug.best\_value < global\_best\_value) {

global\_best\_value = bug.best\_value;

global\_best\_position = bug.best\_position;

}

}

}

Bug optimize() {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dist(0.0, 1.0);

for (size\_t iter = 0; iter < max\_iterations; ++iter) {

for (auto& bug : bugs) {

for (size\_t j = 0; j < dimensions; ++j) {

float r1 = dist(gen);

float r2 = dist(gen);

// Корректируем скорость

bug.velocity[j] = inertia \* bug.velocity[j]

+ c1 \* r1 \* (bug.best\_position[j] - bug.position[j])

+ c2 \* r2 \* (global\_best\_position[j] - bug.position[j]);

// Обновляем позицию

bug.position[j] += bug.velocity[j];

}

// Вычисляем новое значение фитнесс-функции

float fitness = k::f(bug.position);

if (fitness < bug.best\_value) {

bug.best\_value = fitness;

bug.best\_position = bug.position;

}

// Обновляем глобальное лучшее значение

if (fitness < global\_best\_value) {

global\_best\_value = fitness;

global\_best\_position = bug.position;

}

}

// Выводим промежуточные результаты

if (iter % 10 == 0)

std::cout << "Iteration " << iter << ": Best Value = " << global\_best\_value << "\n";

}

return \*std::min\_element(bugs.begin(), bugs.end(), [](const Bug& a, const Bug& b)

{

return k::f(a.position) < k::f(b.position);

});

}

};

namespace k

{

inline float f(const std::vector<float> &x)

{

float result = 10.0 \* x.size();

for (float xi : x) {

result += xi \* xi - 10.0 \* std::cos(2.0 \* M\_PI \* xi);

}

return result;

}

inline std::vector<float> f(const Swarm &x)

{

std::vector<float> \_;

for (const auto &\_\_ : x.bugs)

\_.push\_back(f(\_\_.position));

return \_;

}

} // namespace k

inline void generateData(std::vector<std::vector<float>> &x\_list, std::vector<float> &y)

{

float step = 0.1; // Шаг между точками

std::vector<float> x1;

std::vector<float> x2;

for (float i = k::MIN; i <= k::MAX; i += step) {

x1.push\_back(i);

x2.push\_back(i);

}

for (float xi : x1) {

for (float xj : x2) {

x\_list[0].push\_back(xi);

x\_list[1].push\_back(xj);

y.push\_back(k::f({xi, xj}));

}

}

}

int main() {

size\_t runs = 100;

bocchie::mark \_1("1"), \_2("2"), \_3("3");

size\_t swarm\_size = 30;

size\_t dimensions = 2;

//

// for (auto i = 0; i < runs; ++i)

// {

// Swarm swarm(swarm\_size,

// 3,

// k::MIN,

// k::MAX,

// k::VEL\_MIN,

// k::VEL\_MAX,

// k::C1,

// k::C2,

// k::INERTIA,

// k::MAX\_ITERATIONS);

//

// auto best\_bug = \_1.run([&](){return swarm.optimize();});

// }

//

// std::cout << "For dimensions = 3:\n" << \_1.to\_json<bocchie::accuracy::milliseconds>() << std::endl;

//

// for (auto i = 0; i < runs; ++i)

// {

// Swarm swarm(swarm\_size,

// 5,

// k::MIN,

// k::MAX,

// k::VEL\_MIN,

// k::VEL\_MAX,

// k::C1,

// k::C2,

// k::INERTIA,

// k::MAX\_ITERATIONS);

//

// auto best\_bug = \_2.run([&](){return swarm.optimize();});

// }

//

// std::cout << "For dimensions = 5:\n" << \_2.to\_json<bocchie::accuracy::milliseconds>() << std::endl;

//

// for (auto i = 0; i < runs; ++i)

// {

// Swarm swarm(swarm\_size,

// 10,

// k::MIN,

// k::MAX,

// k::VEL\_MIN,

// k::VEL\_MAX,

// k::C1,

// k::C2,

// k::INERTIA,

// k::MAX\_ITERATIONS);

//

// auto best\_bug = \_3.run([&](){return swarm.optimize();});

// }

//

// std::cout << "For dimensions = 10:\n" << \_3.to\_json<bocchie::accuracy::milliseconds>() << std::endl;

Swarm swarm(swarm\_size,

dimensions,

k::MIN,

k::MAX,

k::VEL\_MIN,

k::VEL\_MAX,

k::C1,

k::C2,

k::INERTIA,

k::MAX\_ITERATIONS);

auto print\_swarm\_plot = [&](){

std::vector<std::vector<float>> x\_list(dimensions);

std::vector<float> y;

generateData(x\_list, y);

std::vector<std::vector<float>> p\_list;

p\_list.reserve(swarm\_size);

for (const auto &\_p : swarm.bugs) {

p\_list.emplace\_back();

for (const auto &gene : \_p.position)

p\_list.back().push\_back(gene);

}

std::vector<float> p;

for (const auto &\_p : swarm.bugs)

p.push\_back(k::f(\_p.position));

plot::print3d(x\_list, y, p\_list, p);

};

print\_swarm\_plot();

auto best\_bug = swarm.optimize();

print\_swarm\_plot();

{

std::vector<std::vector<float>> x\_list(dimensions);

std::vector<float> y;

generateData(x\_list, y);

std::vector<std::vector<float>> p\_list;

p\_list.push\_back(best\_bug.position);

std::vector<float> p;

p.push\_back(k::f(best\_bug.position));

plot::print3d(x\_list, y, p\_list, p);

}

return 0;

}

}

#endif //LAB7\_HPP

**Результаты выполнения:**

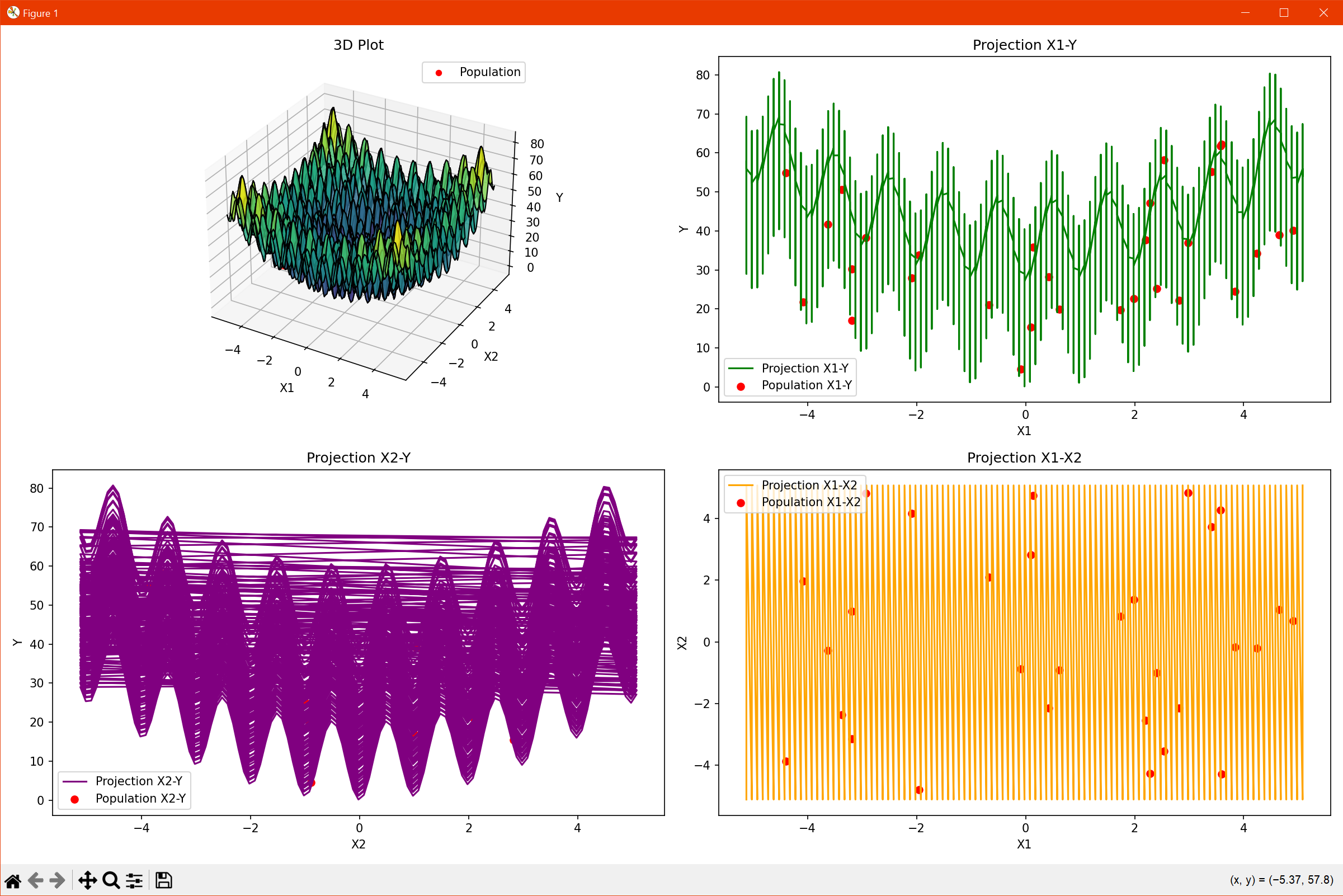


Рис. 1 – Начальная генерация частиц (для функции 2-ух переменных)

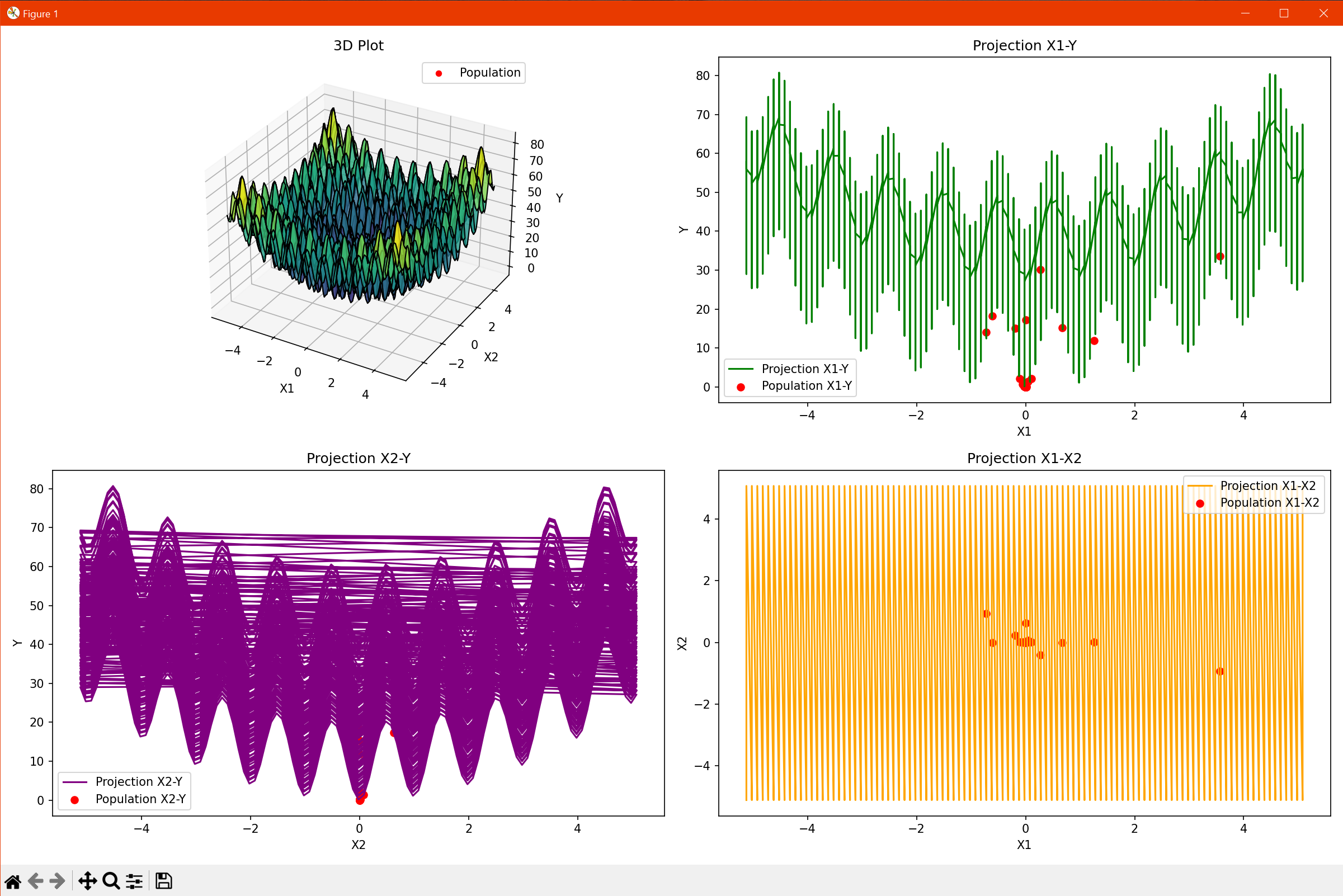
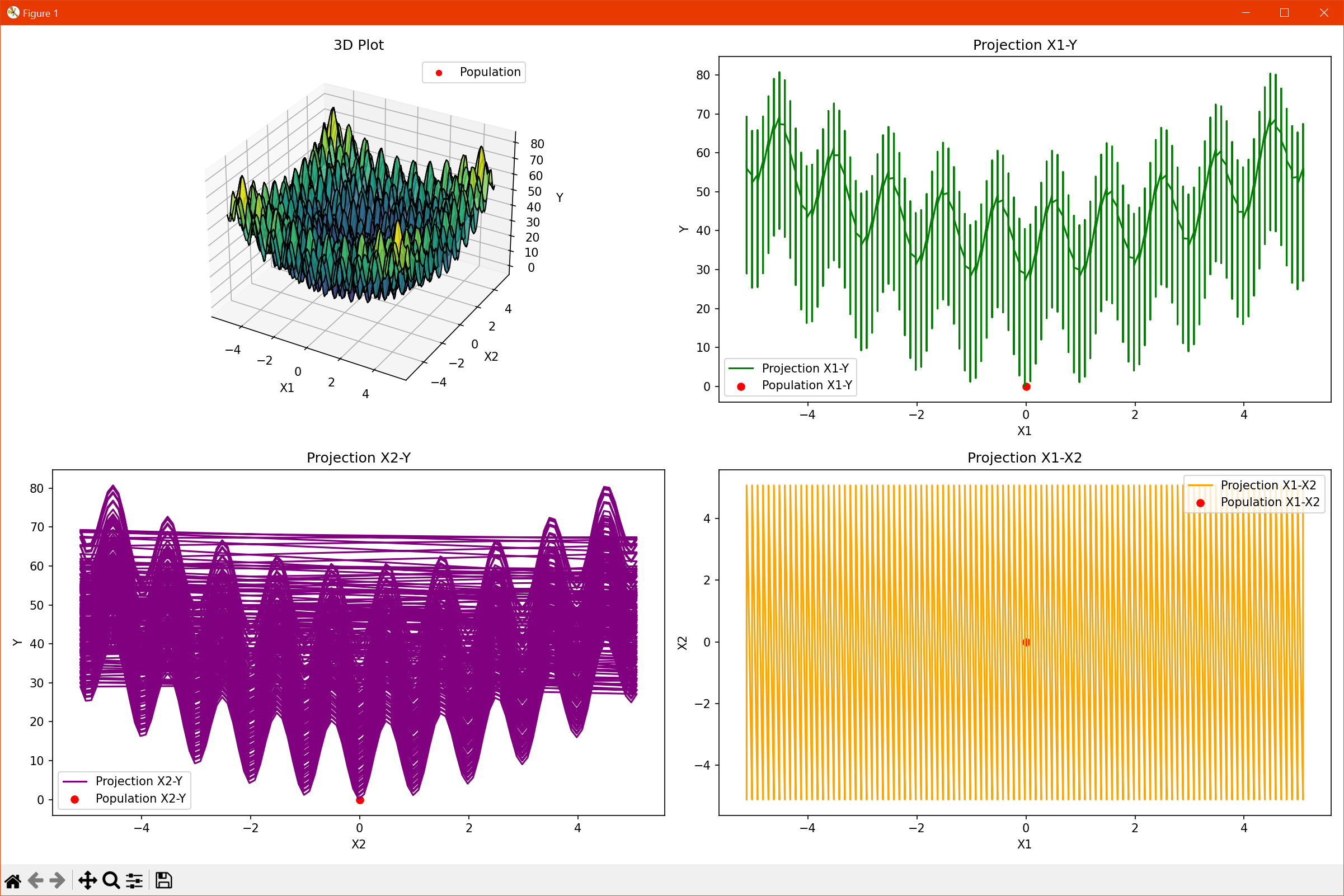


Рис. 2 – Частицы для функции 2-ух переменных на момент конца работы программы



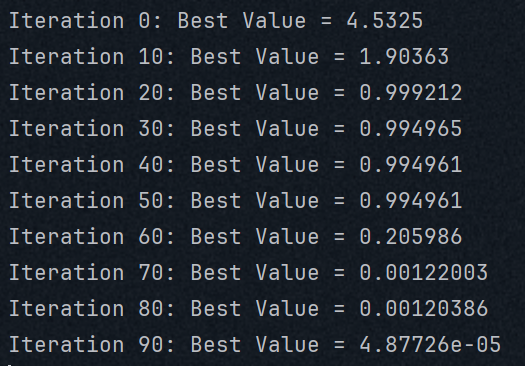


Рис. 3 - Только лучшая особь

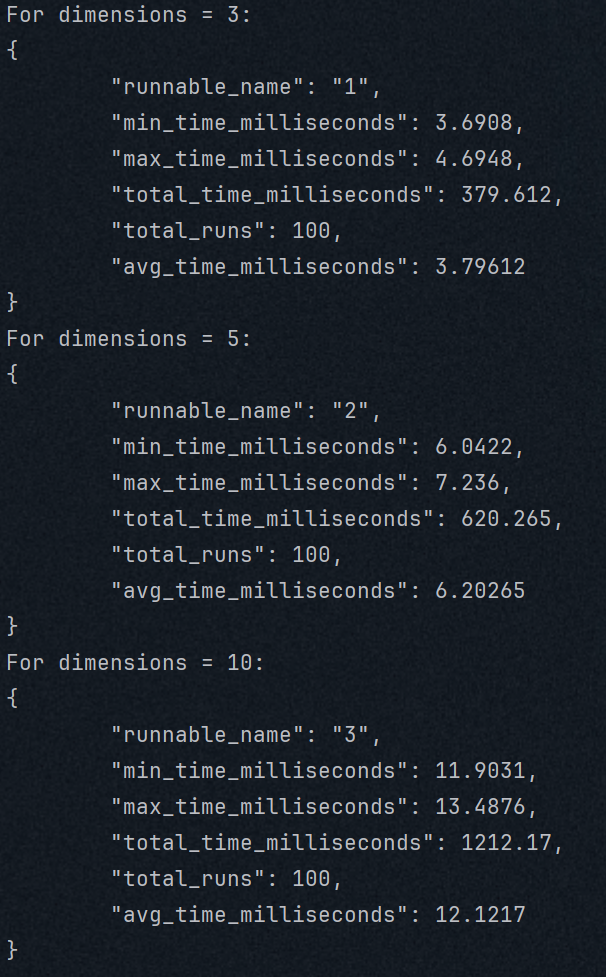


Рис. 4 – Результаты работы алгоритма для функций с различным количеством аргументов

В сравнении с результатами лабораторных работ 2 и 5, роевые алгоритмы проигрывают в сходимости всей популяции. Но если смотреть только лучший результат, то алгоритм работает на том же уровне. Также очевидно: чем больше размерность - тем дольше длятся рассчеты. На такой скромной выборке увеличение времени происходит примерно линейно

**Ответ на контрольный вопрос:**

**10.** **Определите глобальную лучшую позицию.**

Глобальная лучшая позиция в роевом алгоритме — это положение частицы, которое дало наилучшее значение целевой функции среди всех частиц за все итерации.