МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

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

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| **Вещественный генетический алгоритм** |
| по дисциплине: Эволюционные методы проектирования программно-информационных систем |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | Z1431 |  |  |  | М.Д.Быстров |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2021/3572 | |  |  |  |

Санкт-Петербург 2025

Оглавление

[Индивидуальное задание 3](#_Toc196340616)

[Листинг программы 4](#_Toc196340617)

[Результаты выполнения программы 12](#_Toc196340618)

[Графики исследованных зависимостей 16](#_Toc196340619)

[Ответ на контрольный вопрос 20](#_Toc196340620)

[Вывод 21](#_Toc196340621)

# Индивидуальное задание

1. Создать программу, использующую ГА для нахождения оптимума

функции согласно таблице вариантов, приведенной в приложении А.

Для всех Benchmark-ов оптимумом является минимум. Программу

выполнить на встроенном языке пакета Matlab.

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. Повторить нахождение решения с использованием стандартного

Genetic Algorithm toolbox. Сравнить полученные результаты.

4. Исследовать зависимость времени поиска, числа поколений

(генераций), точности нахождения решения от основных параметров

генетического алгоритма:

- число особей в популяции

- вероятность кроссинговера, мутации.

Критерий остановки вычислений – повторение лучшего результата

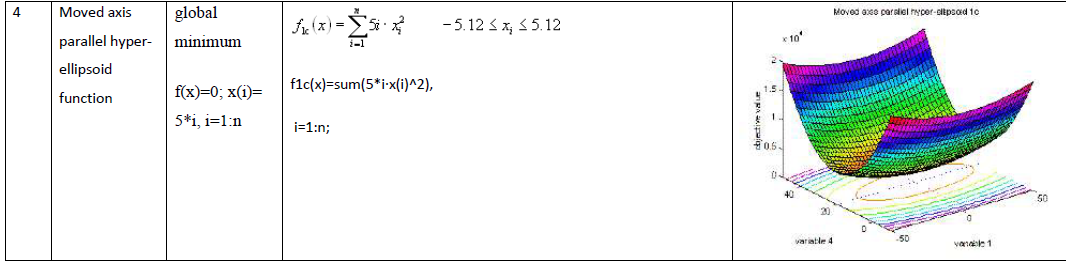
заданное количество раз или достижение популяцией определенного

возраста (например, 100 эпох).

5. Повторить процесс поиска решения для n=3, сравнить результаты,

скорость работы программы.

Вариант 4.



# Листинг программы

Программа выполнена в системе MATLAB. Графики сформированы с помощью встроенных возможностей системы. Решение проверено с помощью встроенного поиска минимума функции генетическим алгоритмом.

% Бенчмарк

% Закомментировать, если нужен обычный запуск

benchmark();

**return**;

% размерность вектора-аргумента

dim = 2;

max\_generations = 500;

population\_size = 100;

res = float\_genetic(@f1c, -5.12, 5.12, dim, population\_size, 50, max\_generations, 0.5, 0.1, true);

fprintf("Результат собственного решения:");

res{1}(:,1)

fprintf("Минимум: %e\n", res{2});

% Решение встроенной функцией

problem.fitnessfcn = @f1c;

problem.nvars = dim;

opt.Generations = max\_generations;

opt.PopulationSize = population\_size;

problem.options = opt;

internal\_res = ga(problem);

fprintf("Результат библиотечного решения:");

internal\_res

fprintf("Минимум: %e\n", f1c(internal\_res));

% Исследование работы алгоритма

**function** **benchmark**()

launches\_num = 20;

% Кол-во в популяции,

% вероятность кросс.,

% вероятность мутации,

% кол-во шагов,

% кол-во запусков

mut\_prob\_test = [

100, 0.5, 0.1, 200, launches\_num;

100, 0.5, 0.01, 200, launches\_num;

100, 0.5, 0.001, 200, launches\_num];

cross\_prob\_test = [

100, 0.9, 0.1, 100, launches\_num;

100, 0.5, 0.1, 100, launches\_num;

100, 0.1, 0.1, 100, launches\_num];

pop\_size\_test = [

20, 0.9, 0.1, 100, launches\_num;

100, 0.5, 0.1, 100, launches\_num;

200, 0.1, 0.1, 100, launches\_num];

units = {

{"Зависимость от вероятности мутаций", mut\_prob\_test};

{"Зависимость от вероятности кроссинговера", cross\_prob\_test};

{"Зависимость от размера популяции", pop\_size\_test}};

units\_results = cell(length(units), 1);

**for** i = 1:length(units)

test\_unit = units{i}{2};

results = zeros(size(test\_unit, 1), 2);

**for** t = 1:size(test\_unit, 1)

pop\_size = test\_unit(t, 1);

cross\_prob = test\_unit(t, 2);

mut\_prob = test\_unit(t, 3);

step\_num = test\_unit(t, 4);

launches = test\_unit(t, 5);

duration\_sum = 0;

solve\_sum = 0;

**for** l = 1:launches

tic

res = float\_genetic(@f1c, -5.12, 5.12, 2, pop\_size, 50, step\_num, cross\_prob, mut\_prob, 0);

duration\_sum = duration\_sum + toc;

solve\_sum = solve\_sum + res{2};

**end**

duration\_sum = duration\_sum ./ launches;

solve\_sum = solve\_sum ./ launches;

results(t, :) = [solve\_sum, duration\_sum];

fprintf("Запуск %d, тест %d: Средняя погрешность %e, среднее время %f\n", i, t, solve\_sum, duration\_sum);

**end**

units\_results{i} = results;

**end**

figure("Name", "Зависимость погрешности от вероятности мутаций");

yscale log

x = compose("Вероятность мутации = %f", mut\_prob\_test(:, 3));

y = units\_results{1}(:,1);

bar(x, y);

legend("Погрешность")

figure("Name", "Зависимость времени решения от вероятности мутаций");

yscale log

x = compose("Вероятность мутации = %f", mut\_prob\_test(:, 3));

y = units\_results{1}(:,2);

bar(x, y);

legend("Время решения, с")

figure("Name", "Зависимость погрешности от вероятности скрещивания");

yscale log

x = compose("Вероятность скрещивания = %f", cross\_prob\_test(:, 2));

y = units\_results{2}(:,1);

bar(x, y);

legend("Погрешность")

figure("Name", "Зависимость времени решения от вероятности скрещивания");

yscale log

x = compose("Вероятность скрещивания = %f", cross\_prob\_test(:, 2));

y = units\_results{2}(:,2);

bar(x, y);

legend("Время решения, с")

figure("Name", "Зависимость погрешности от размера популяции");

yscale log

x = compose("Размер популяции = %d", pop\_size\_test(:, 1));

y = units\_results{3}(:,1);

bar(x, y);

legend("Погрешность")

figure("Name", "Зависимость времени решения от размера популяции");

yscale log

x = compose("Размер популяции = %d", pop\_size\_test(:, 1));

y = units\_results{3}(:,2);

bar(x, y);

legend("Время решения, с")

**end**

**function** **benchmark\_launch**(props)

**for** i = 1:size(props, 2)

% todo подсчет бенчмарка

**end**

**end**

% генетический алгоритм

**function** **res** = **float\_genetic**( ...

f, ...

from, ...

to, ...

dim, ...

population\_size, ...

ratio, ...

max\_steps, ...

cross\_prob, ...

mutation\_prob, ...

is\_draw)

% стартовая популяция

start\_population = generate\_population(population\_size, dim, from, to);

**if** (dim == 2 && is\_draw)

figure("Name","Начальное поколение");

draw\_pop\_on\_function(f, from, to, start\_population, ratio);

view([100, 55]);

**end**

population = start\_population;

**for** step = 1:max\_steps

% отбор родителей

parents = reproduction(population, f);

% рождение и мутация детей

children = crossover(parents, cross\_prob);

children = mutation(children, mutation\_prob);

% редукция

new\_population = [children; population];

population = reduction(f, new\_population, population\_size);

**if** (dim == 2 && mod(step, 100) == 0 && is\_draw)

figure("Name","Поколение " + step);

draw\_pop\_on\_function(f, from, to, population, ratio);

view([100, 55]);

**end**

**end**

**if** (dim == 2 && is\_draw)

figure("Name","Итог после " + step + " шагов");

draw\_pop\_on\_function(f, from, to, population, ratio);

view([100, 55]);

**end**

best = best\_individ(f, population);

res = {population{best}; f(population{best})};

**end**

% Скрещивание

**function** **pop** = **crossover**(parents, prob)

len = length(parents);

pop = cell(1, 1);

**for** i = 1:len

p1 = parents{round(rand\_range(1, len))};

p2 = parents{round(rand\_range(1, len))};

**if** (rand() < prob)

child = flat\_crossover(p1, p2);

pop{length(pop)} = child;

**end**

**end**

**end**

% Плосский кроссовер

**function** **child** = **flat\_crossover**(p1, p2)

dim = length(p1);

child = zeros(dim);

**for** i = 1:dim

child(i) = rand\_range(p1(i), p2(i));

**end**

**end**

% Мутация

**function** **mutated\_pop** = **mutation**(pop, prob)

**for** i = 1:length(pop)

**if** (rand() < prob)

val = pop{i};

**for** j = 1:length(val)

val(j) = rand\_range(val(j) - 0.1, val(j) + 0.1);

**end**

pop{i} = val;

**end**

**end**

mutated\_pop = pop;

**end**

% Редукция

**function** **reducted\_pop** = **reduction**(f, pop, len)

valind = configureDictionary("double", "cell");

cnt = 0;

arr = {{}};

**for** i = 1:length(pop)

val = f(pop{i});

**if** cnt > 0

**if** (valind.isKey(val))

arr = valind.lookup(val);

**else**

arr = {{}};

**end**

**else**

arr = {{}};

**end**

subarr = arr{1};

subarr{length(subarr) + 1} = pop{i};

arr{1} = subarr;

% if (cnt == 0)

% val2ind = dictionary(val, arr);

% end

valind = valind.insert(val, arr);

cnt = cnt + 1;

**end**

reducted\_pop = {};

**while** (length(reducted\_pop) < len & ~isempty(valind.keys()))

minval = min(valind.keys);

arr = valind.lookup(minval);

unique = configureDictionary("double", "double");

**for** j = 1:length(arr{1})

ind = arr{1}{j};

% is\_member = ismember([ind], cell2mat(reducted\_pop));

**if** (~unique.isKey(ind))

reducted\_pop = [reducted\_pop; ind];

unique = unique.insert(ind, 1);

**end**

**end**

valind = valind.remove(minval);

**end**

**end**

**function** **draw\_pop\_on\_function**(f, from, to, pop, ratio)

draw\_function(f, from, to, ratio);

hold on;

draw\_population(f, pop);

hold off;

**end**

**function** **intermediate\_population** = **reproduction**(population, f)

pop\_size = length(population);

% Оператор репродукции

intermediate\_population = cell(pop\_size, 1);

individ\_values = zeros(pop\_size, 1);

% сумма всех значений и значение для каждой особи

**for** i = 1:pop\_size

val = f(population{i});

individ\_values(i) = val;

**end**

% смещаем, чтобы не было отрицательных значений, считаем сумму

% min\_poten = min(individ\_values);

% offset = 0 - min\_poten;

%

% for i = 1:length(individ\_values)

% individ\_values(i) = individ\_values(i) + offset;

% end

values\_sum = sum(individ\_values);

% все особи одинаковы

**if** values\_sum == 0

intermediate\_population = population;

**return**

**end**

potentials = zeros(pop\_size, 1);

% подсчет потенциала для каждой особи

% потенциалы отрицательны, т.к. ищем минимум

**for** i = 1:length(individ\_values)

val = individ\_values(i);

prob = -(val / values\_sum);

potentials(i) = prob;

**end**

min\_potential = min(potentials);

% сдвигаем потенциалы в положительную часть

**for** i = 1:length(individ\_values)

potentials(i) = potentials(i) - min\_potential;

**end**

sum\_potentials = sum(potentials);

% выбор такого же кол-ва особей

**for** i = 1:length(population)

% крутите барабан

shot = rand\_range(0, sum\_potentials);

individ\_num = 0;

tmp\_sum = 0;

% определяем куда попали барабаном

**for** j = 1:length(population)

individ\_num = j;

tmp\_sum = tmp\_sum + potentials(individ\_num);

**if** (tmp\_sum >= shot)

**break**

**end**

**end**

% добавляем выбранную особь в промежуточную популяцию

individ = population{individ\_num};

intermediate\_population{i} = individ;

**end**

**end**

% Генерация начальной популяции

**function** **pop** = **generate\_population**(pop\_size, ind\_size, from, to)

pop = cell(pop\_size, 1);

% pop = zeros(pop\_size, ind\_size);

**for** i = 1:pop\_size

ind = zeros(ind\_size);

**for** j = 1:ind\_size

ind(j) = rand\_range(from, to);

**end**

pop{i} = ind;

**end**

**end**

% случайное число в диапазоне

**function** **ret** = **rand\_range**(from, to)

ret = (to-from) .\* rand() + from;

**end**

**function** **num** = **best\_individ**(f, pop)

min\_val = intmax();

num = 0;

**for** i = 1:length(pop)

val = f([pop{i}(1), pop{i}(2)]);

**if** (val < min\_val)

min\_val = val;

num = i;

**end**

**end**

**end**

% Вывод объемного графика функции двух переменных

**function** **draw\_function**(f, from, to, dim)

%tmp = @(x,y) f([x, y]);

x = linspace(from, to, dim);

y = x;

[X,Y] = meshgrid(x,y);

VAL = zeros(length(X), length(Y));

**for** i = 1:length(X)

**for** j = 1:length(Y)

VAL(i, j) = f([X(i,j), Y(i,j)]);

**end**

**end**

surf(X, Y, VAL)

**end**

% нарисовать популяцию с выделением лучшей особи

**function** **draw\_population**(f, pop)

best = best\_individ(f, pop);

best\_val = 0;

**for** i = 1:length(pop)

ind = pop{i};

x1 = ind(1);

y1 = ind(2);

l = plot3(x1,y1,f([x1,y1]), 'ko');

**if** (i == best || best\_val == f([x1,y1]))

l.Color = "red";

l.MarkerFaceColor = "red";

best\_val = f([x1,y1]);

**else**

l.Color = "yellow";

l.MarkerFaceColor = "yellow";

**end**

**end**

**end**

% Оптимизируемая функция

**function** **v** = **f1c**(x)

v = 0.;

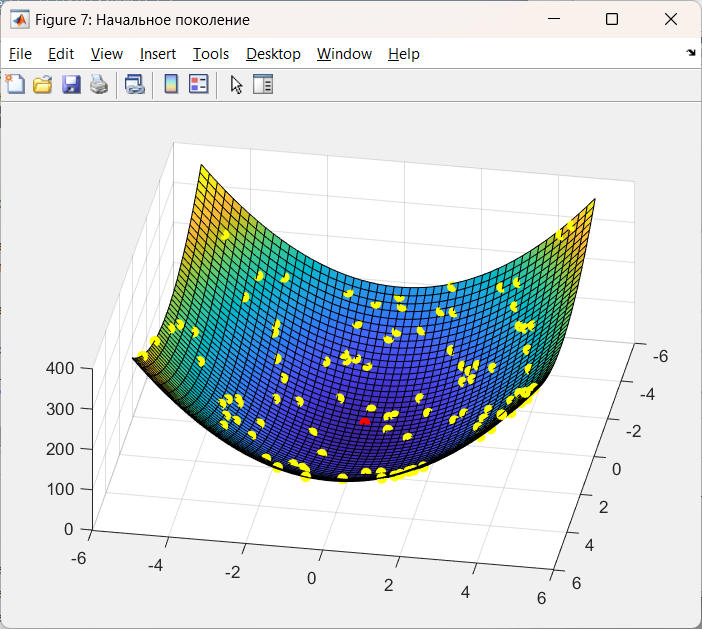
**for** i = 1:length(x)

v = v + 5 .\* i .\* (x(i).^2);

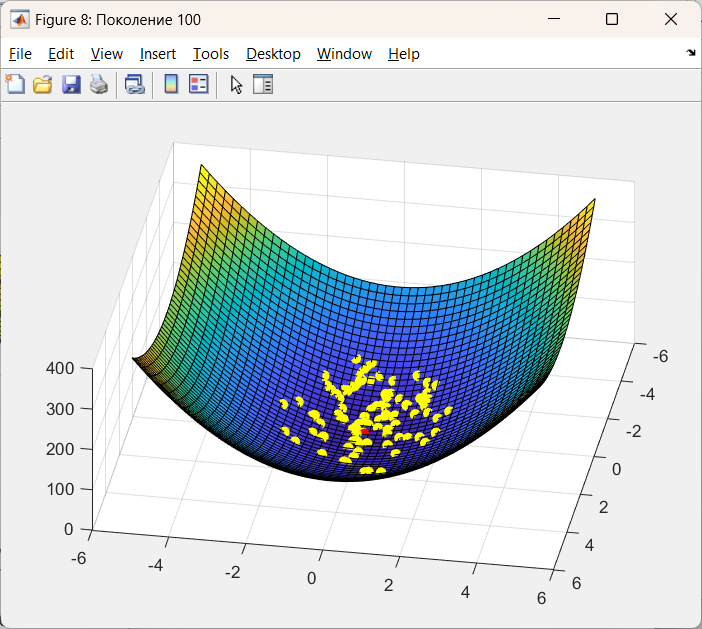
**end**

**end**

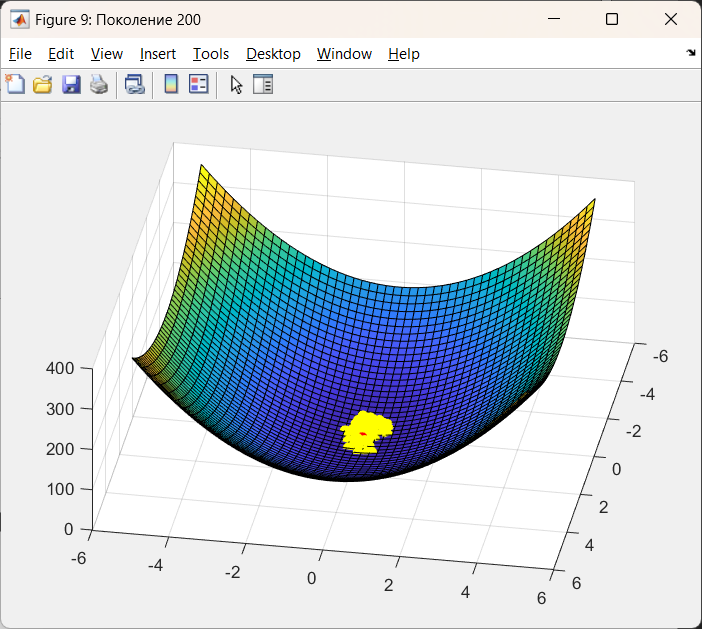
# Результаты выполнения программы



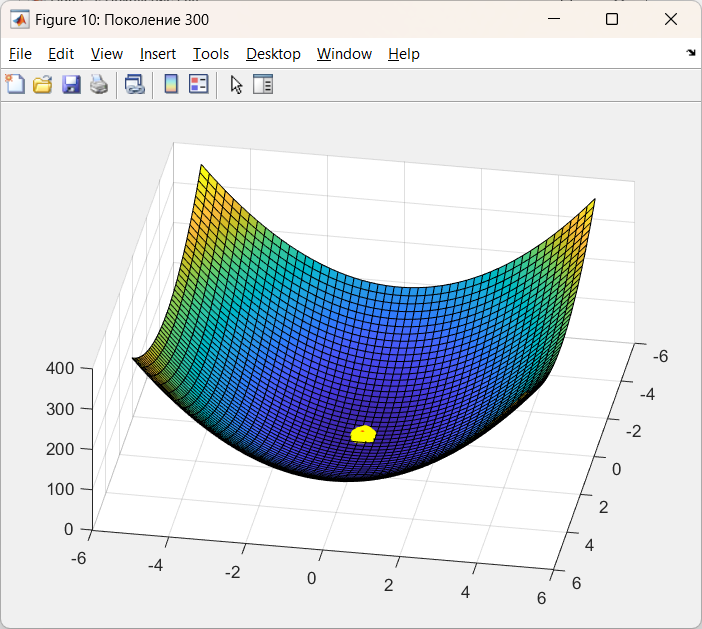
*Рисунок 1 Начальное поколение*



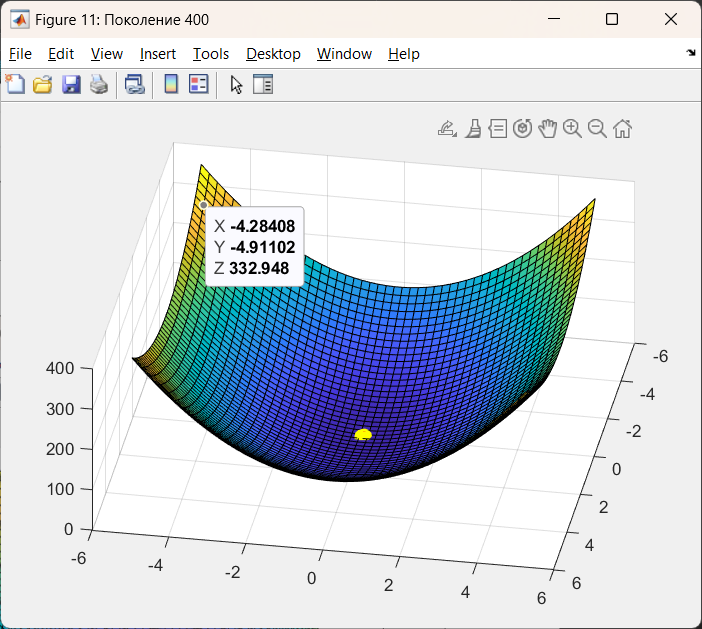
*Рисунок 2 Поколение 100*



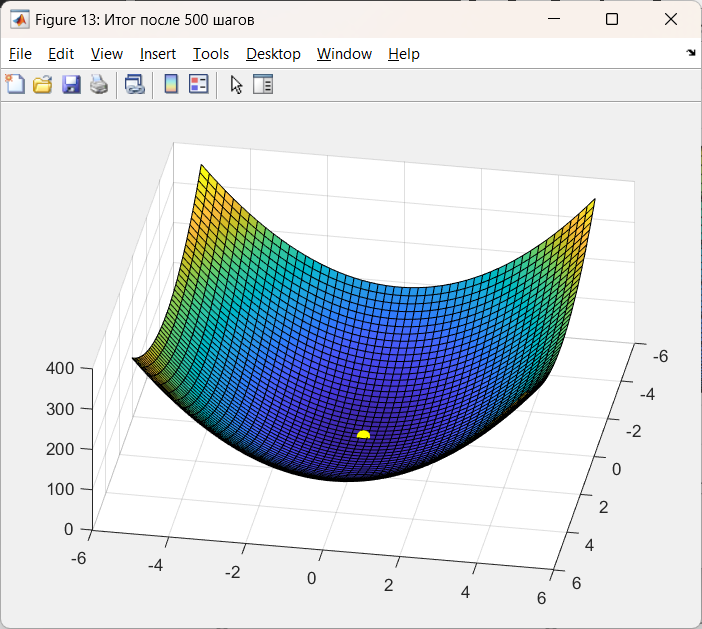
*Рисунок 3 Поколение 200*



*Рисунок 4 Поколение 300*



*Рисунок 5 Поколение 400*



*Рисунок 6 Итоговый результат*

На графиках особи популяции отображаются желтыми точками, самая приспособленная особь отображается красной точкой.

**Пример решения для вектора X размерностью 2**

Результат собственного решения:

X =1.0e-03 \*0.2198, 0.3962

Минимум f: 1.811266e-06

Результат библиотечного решения:

X = 0.0895,-0.0555

Минимум f: 7.084752e-02

Собственная реализация показала результат, превышающий встроенный алгоритм с настройками по умолчанию.

**Пример решения для вектора X размерностью 3**

Результат собственного решения:

X = 0.0061,-0.0038,0.0008

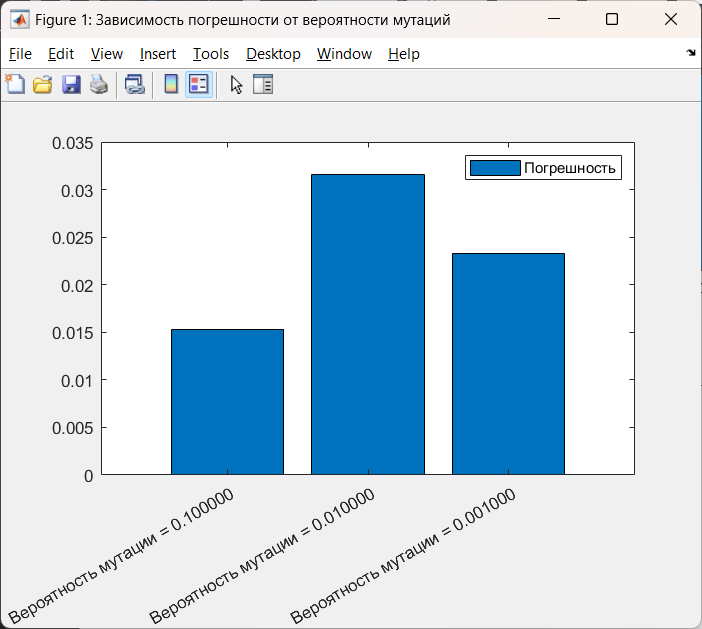
Минимум f: 3.424036e-04

Результат библиотечного решения:

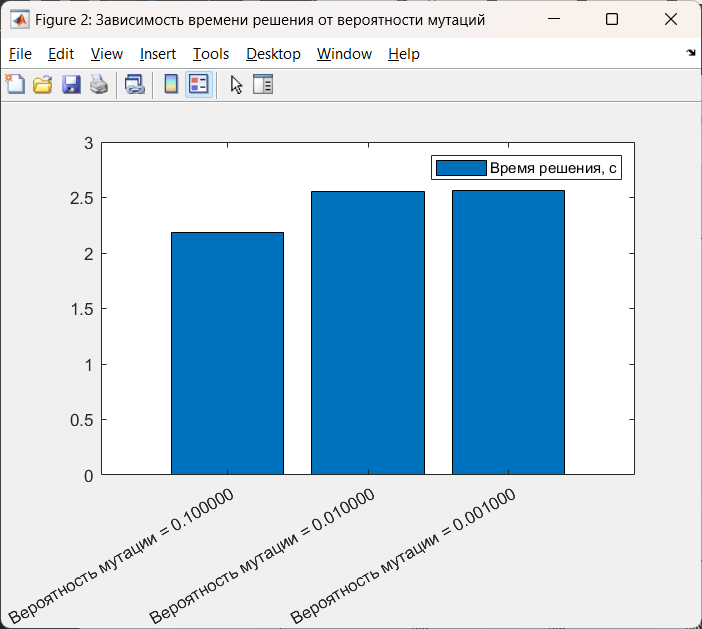
X =0.2796,0.1612,-0.0356

Минимум: 6.697236e-01

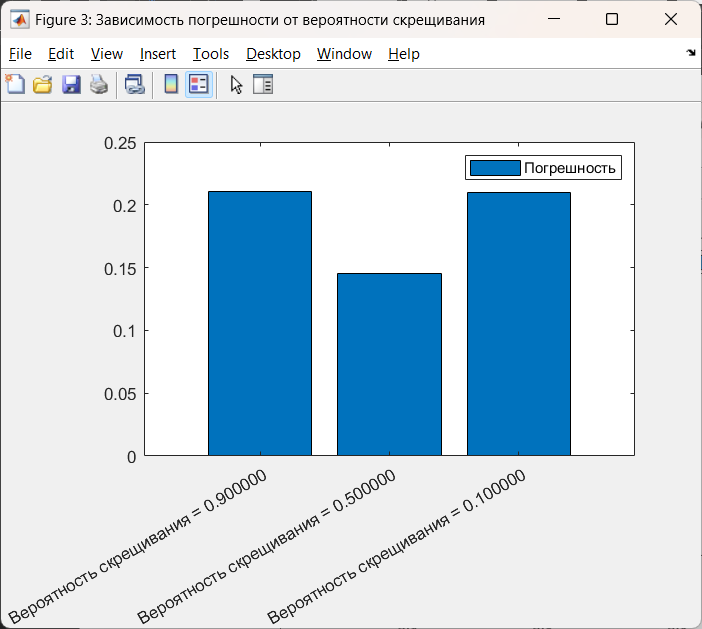
# Графики исследованных зависимостей



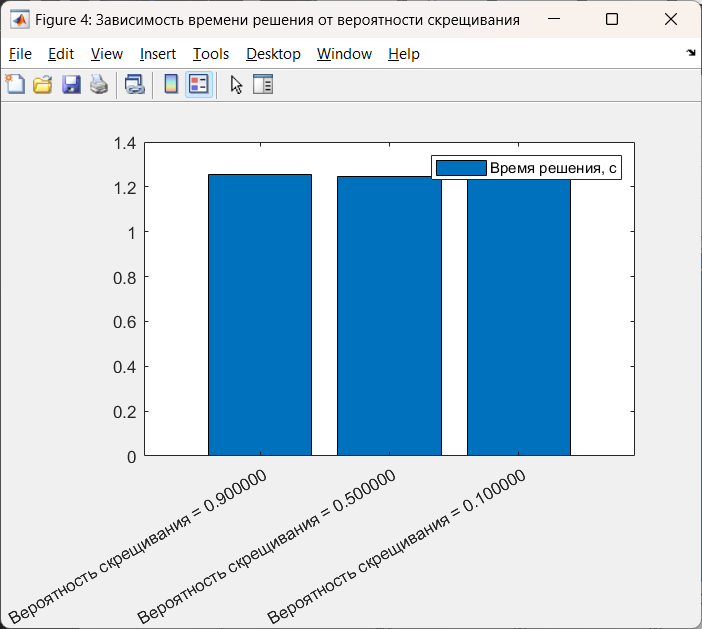
*Рисунок 7*



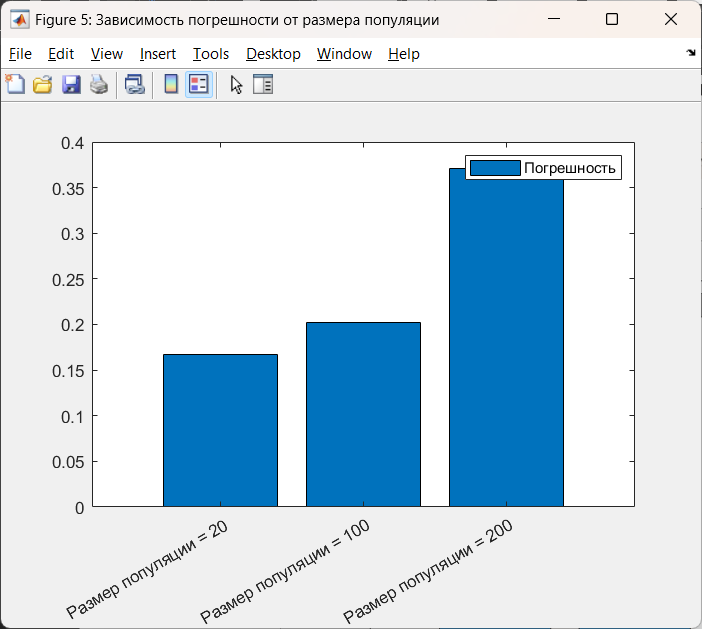
*Рисунок 8*



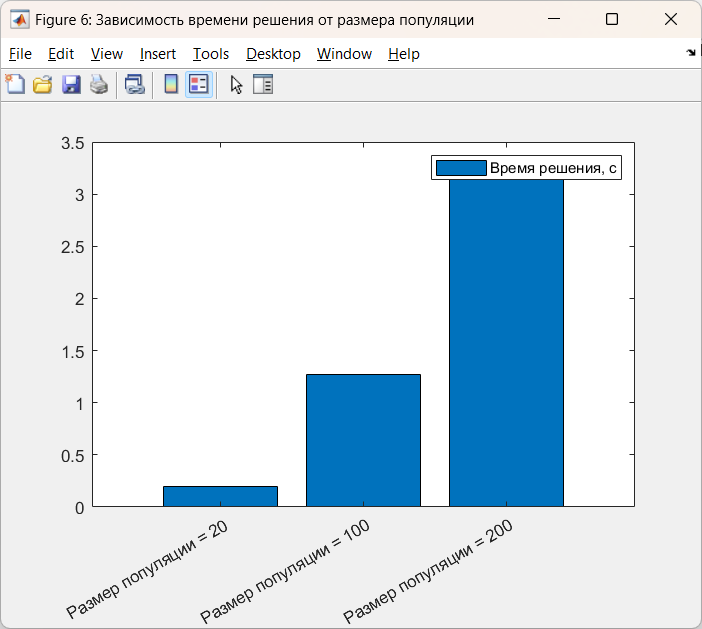
*Рисунок 9*



*Рисунок 10*



*Рисунок 11*



*Рисунок 12*

Результаты исследований представлены на рисунках 7-12. По умолчанию использовались параметры: размер популяции 100, вероятность мутации 0.1, вероятность скрещивания 0.5, 200 шагов для исследования мутации, 100 шагов для остальных исследований, 20 запусков каждой конфигурации.

# Ответ на контрольный вопрос

4. Что является целью оптимизационной задачи?

Задача нахождения экстремума (минимума или максимума) целевой функции в некоторой области конечномерного векторного пространства, ограниченной набором линейных и/или нелинейных равенств или неравенств.

# Вывод

В ходе выполнения второй лабораторной работы была написана программа в среде MATLAB для поиска оптимума функции нескольких переменных с помощью вещественного генетического алгоритма.

Используется плоский оператор кроссинговера и случайная мутация.

Приведены графики решения в 3-размерном пространстве для размерности X=2, в том числе промежуточные.

Приведены графики исследования зависимости скорости и погрешности работы алгоритма от различных параметров алгоритма.

Дан ответ на контрольный вопрос согласно варианту.