

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
должность, уч. степень, звание

подпись, дата

Ю.А. Скобцов
инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Вещественный генетический алгоритм

по дисциплине: Эволюционные методы проектирования программно-информационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

Z1431

номер группы

подпись, дата

М.Д.Быстров

инициалы, фамилия

Студенческий билет №

2021/3572

Санкт-Петербург 2025

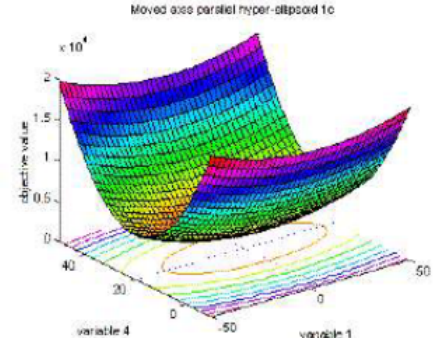
Оглавление

Индивидуальное задание	3
Листинг программы.....	4
Результаты выполнения программы.....	12
Графики исследованных зависимостей.....	16
Ответ на контрольный вопрос.....	20
Вывод.....	21

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.

4	Moved axis parallel hyper-ellipsoid function	<p>global minimum</p> <p>$f(x)=0; x(i)=5*i, i=1:n$</p>	$f_k(x) = \sum_{i=1}^n 5i \cdot x_i^2 \quad -5.12 \leq x_i \leq 5.12$ <p>$f1c(x)=\text{sum}(5*i*x(i)^2),$</p> <p>$i=1:n;$</p>	
---	--	---	---	---

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

% Плоский 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
end

```

```

end

mutated_pop = pop;
end

% Редукция
function reduced_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

    reduced_pop = {};

    while (length(reduced_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(reduced_pop));

            if (~unique.iskey(ind))
                reduced_pop = [reduced_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

```


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
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

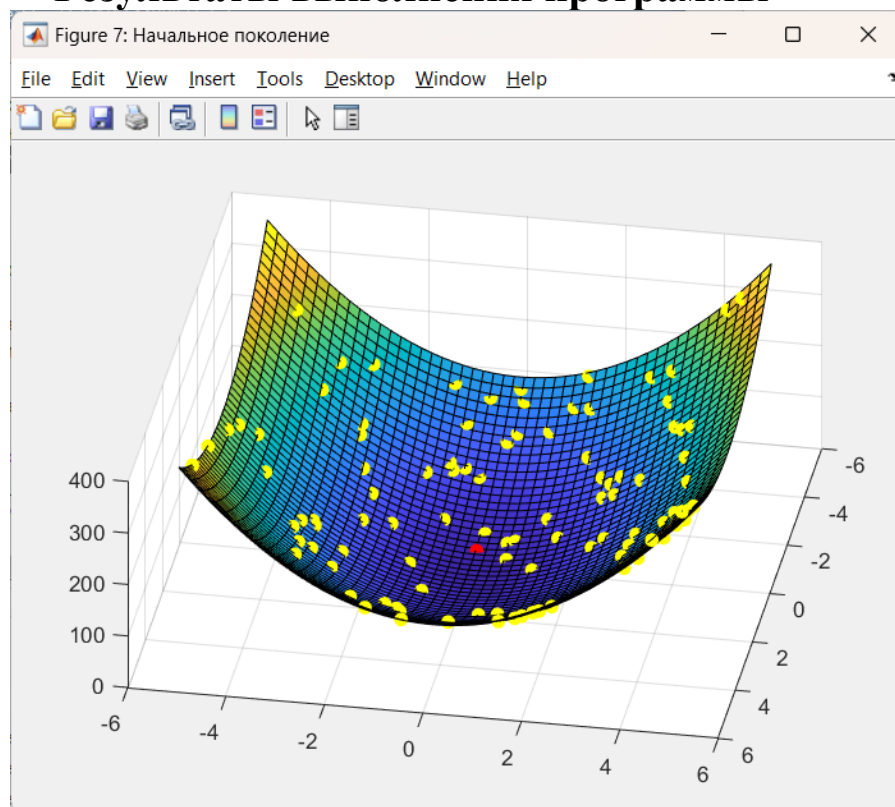


Рисунок 1 Начальное поколение

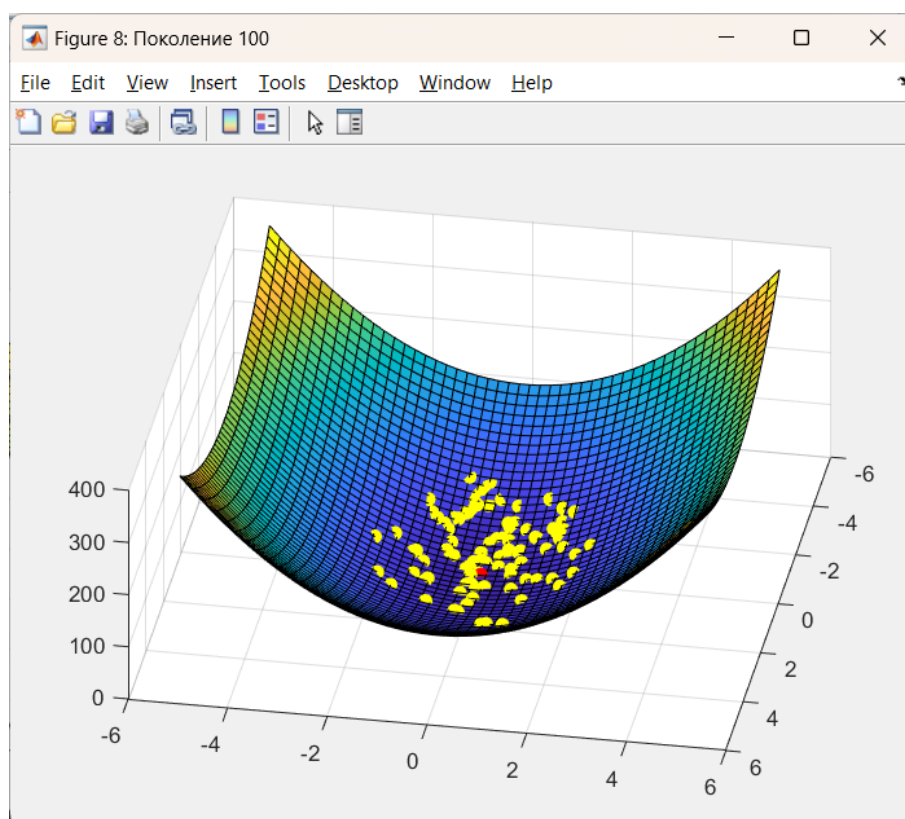


Рисунок 2 Поколение 100

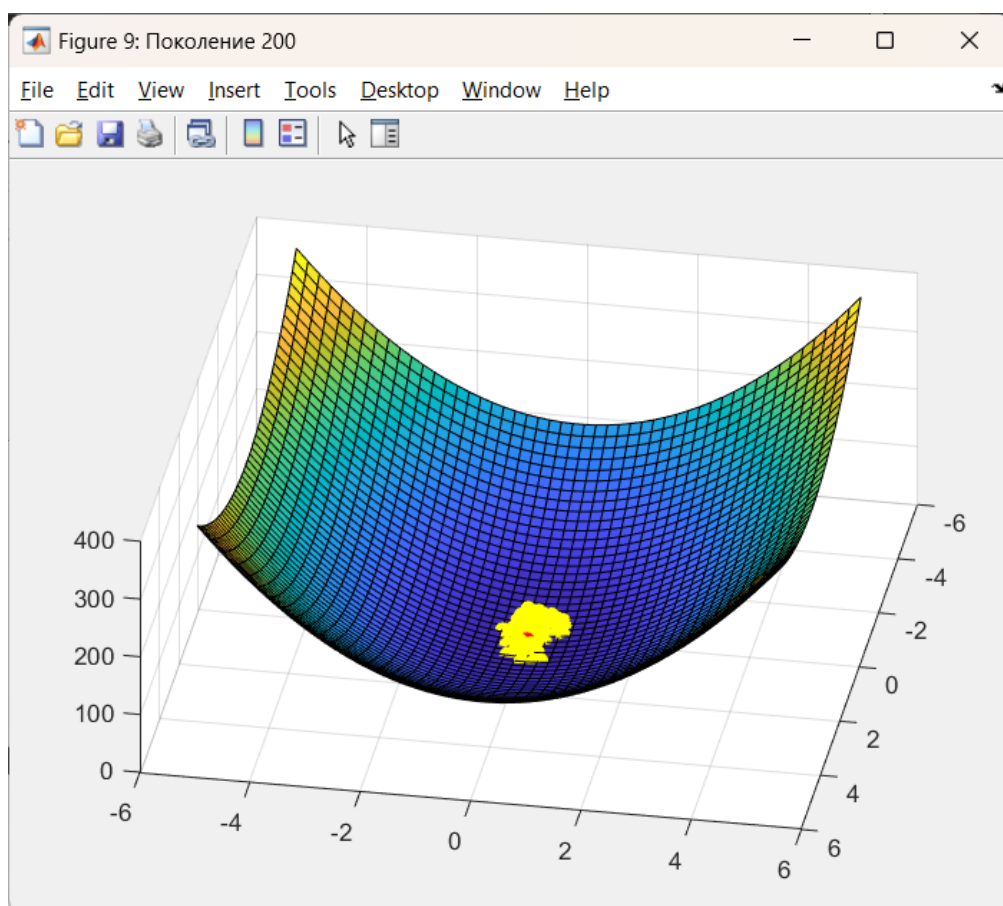


Рисунок 3 Поколение 200

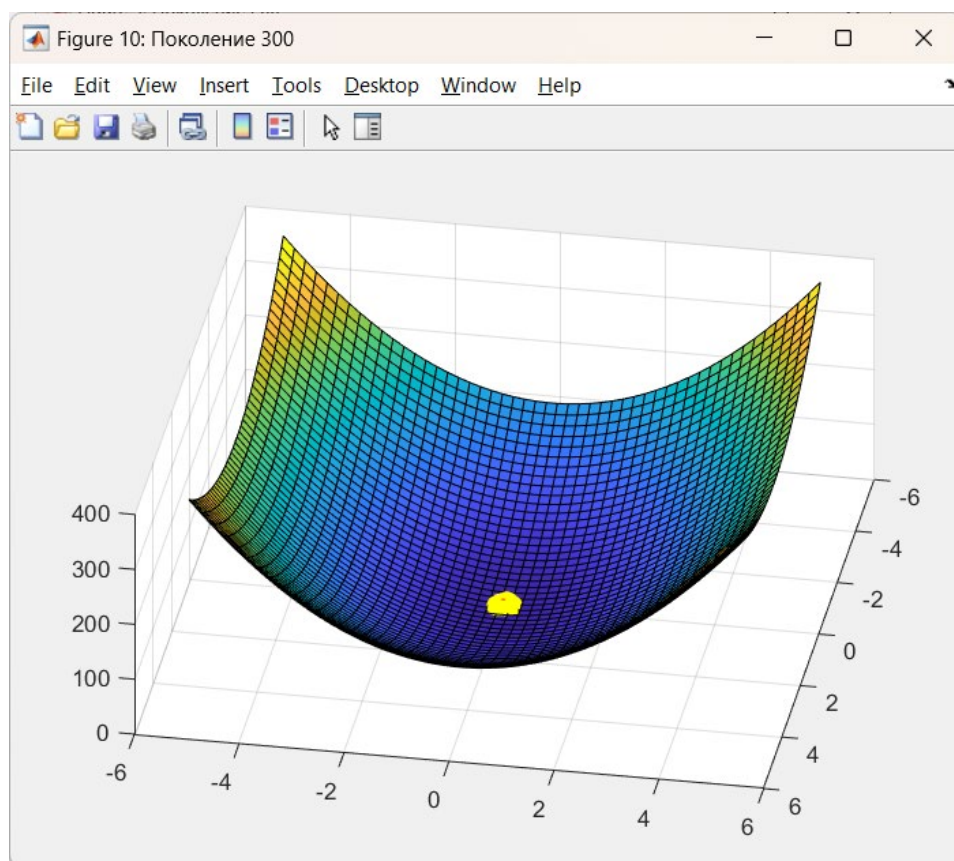


Рисунок 4 Поколение 300

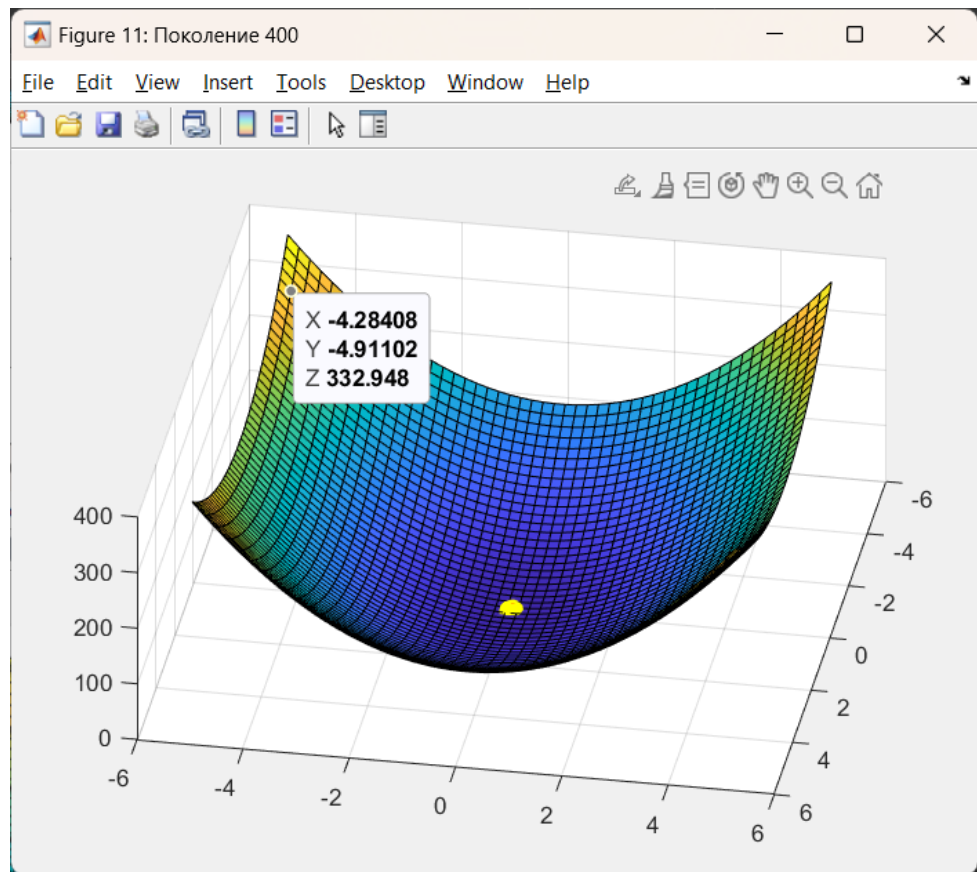


Рисунок 5 Поколение 400

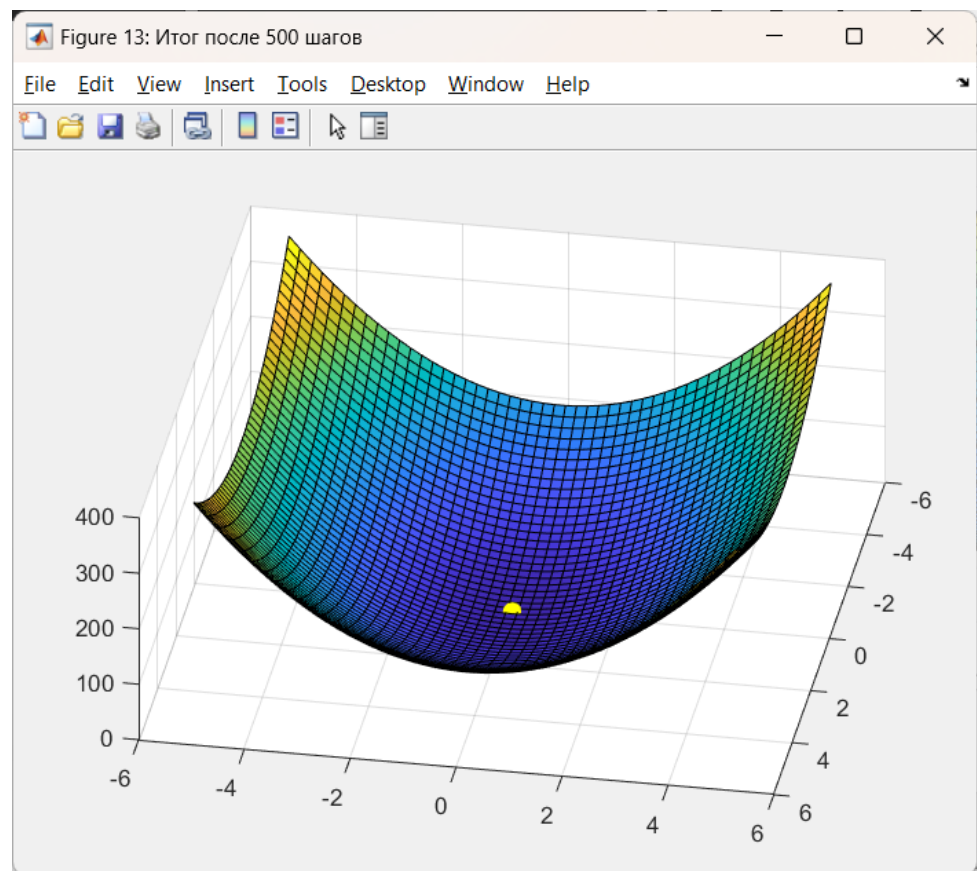


Рисунок 6 Итоговый результат

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

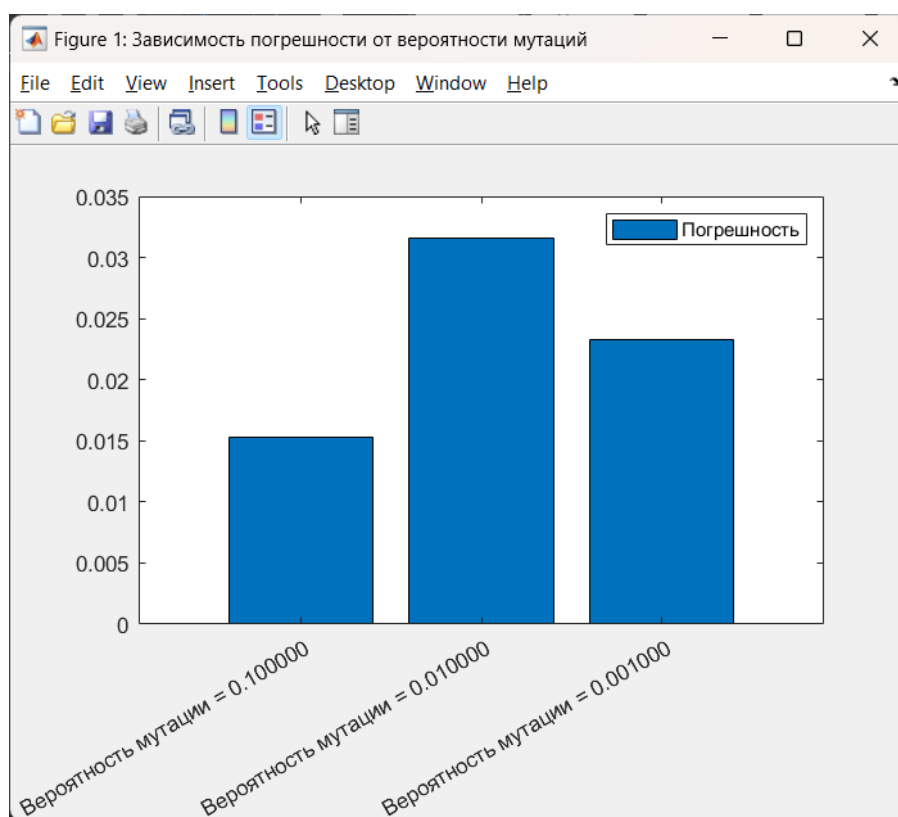


Рисунок 7

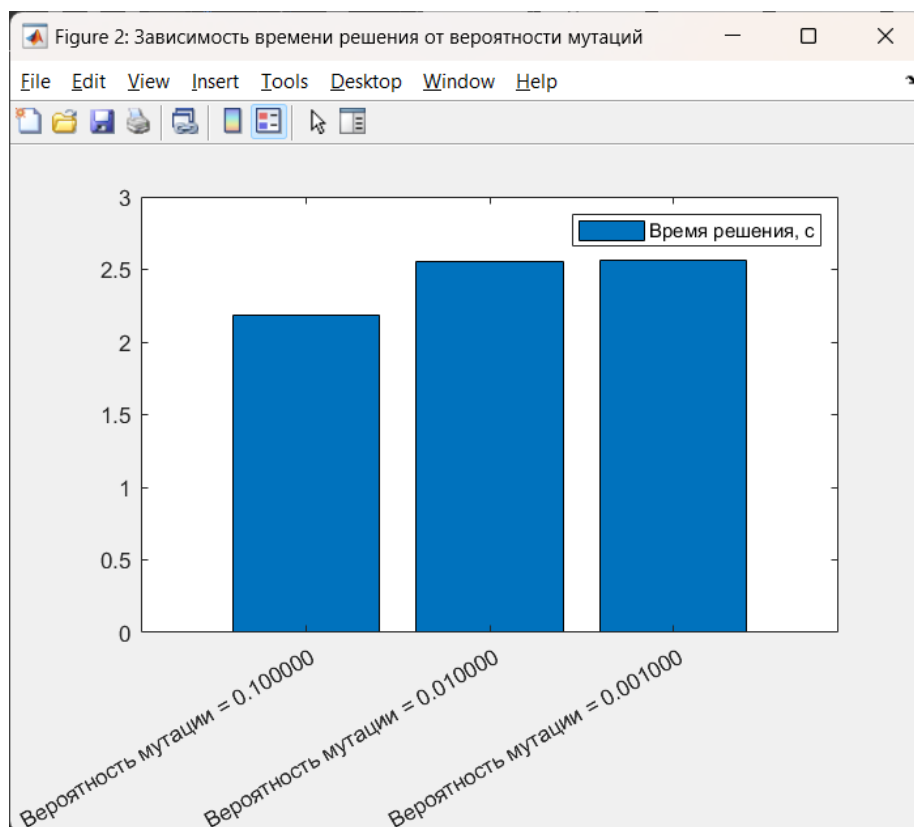


Рисунок 8

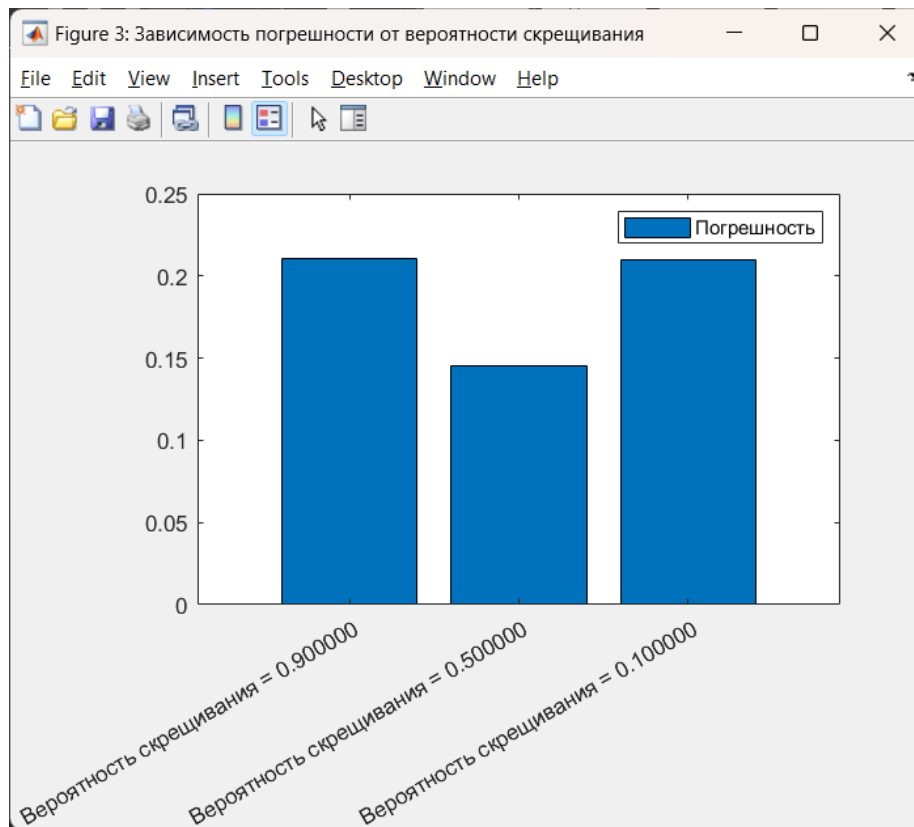


Рисунок 9

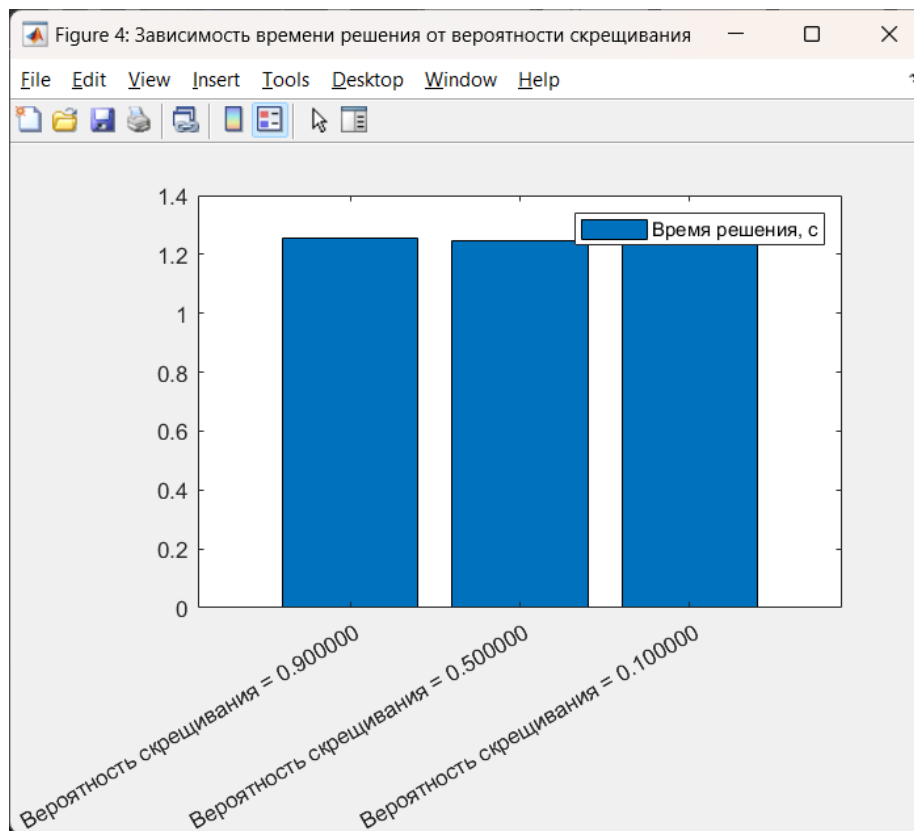


Рисунок 10

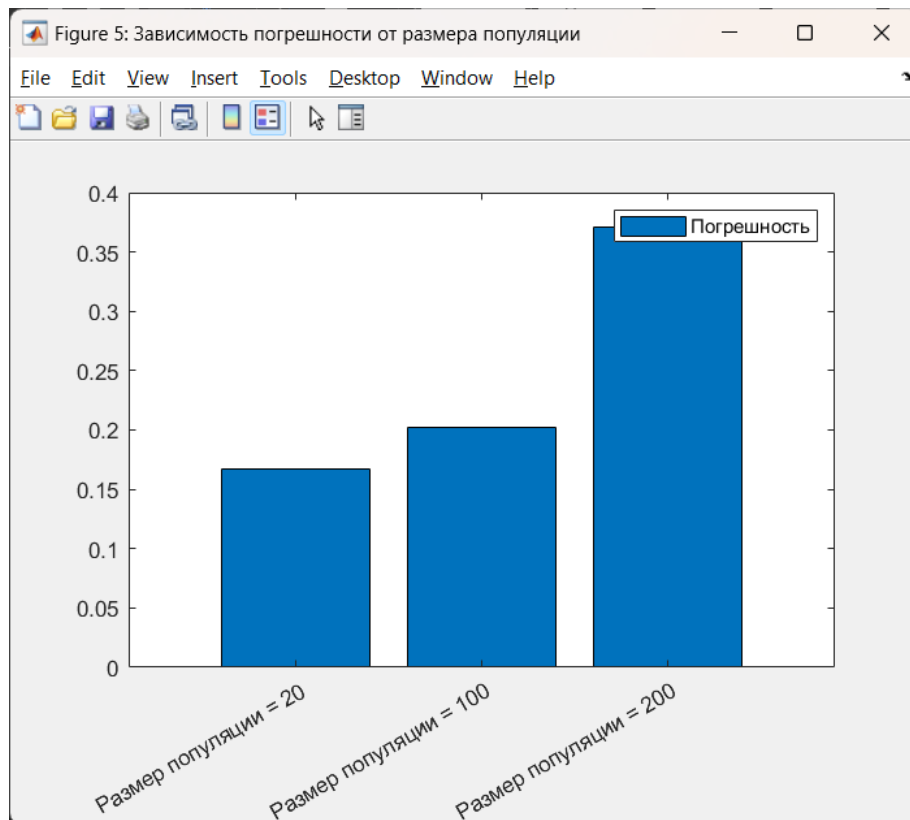


Рисунок 11

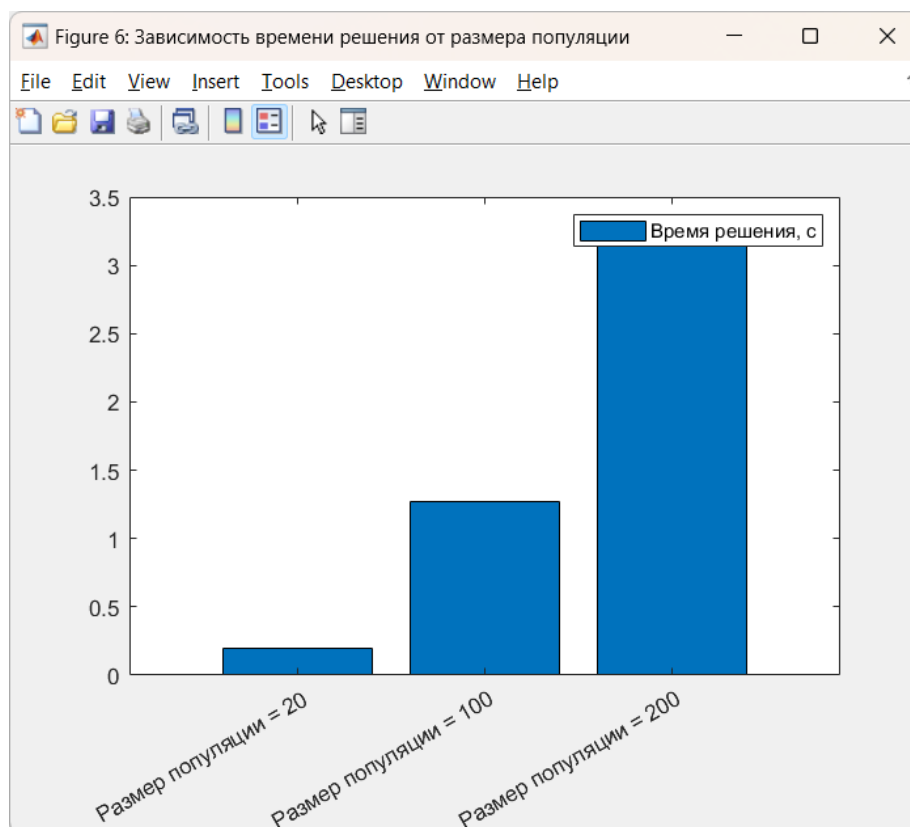


Рисунок 12

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.