目　录

一、作业要求

进化算法是一类模拟自然生物进化机制的智能优化算法。它基于种群，通过对多个个体编码表示，运用选择、交叉、变异等遗传操作，或类似的迭代更新规则，让种群逐代演化，适应度高的个体更易留存繁衍，最终逼近问题最优解。

本次是要要求选取基于某种生物的进化算法，并且将其应用于三个实际问题中，通过可视化界面和迭代数据，展现出该算法的优势。

二、实验原理

2.1 鸽群算法特征

鸽群算法特征	进化算法对应特征
种群初始化	**种群初始化**
鸽群算法通过随机初始化鸽子的位置信息，表示多个潜在的解。	进化算法通过随机初始化种群中的个体，表示多个潜在解。
自然选择	**选择操作**
鸽群算法根据鸽子的适应度（距离目标巢穴的距离）进行选择，适应度高的鸽子会更接近最优解。	进化算法通过选择适应度较高的个体作为父代，淘汰适应度较低的个体。
局部搜索与全局搜索	**变异操作**
鸽群算法通过鸽子之间相互调整位置，既进行局部搜索也进行全局搜索，寻找到最优巢穴。	进化算法通过变异操作在解空间中探索新的可能解，避免陷入局部最优。
适应度评估	**适应度评估**
鸽群算法通过评估每只鸽子到目标栖息地的距离，来衡量其适应度。	进化算法通过评估个体的适应度，决定其在种群中的存活与否。
种群多样性	**保持种群多样性**
鸽群算法保持了多样性的群体行为，防止鸽群集中在某个区域，保证搜索空间的广度。	进化算法通过种群的多样性避免种群陷入局部最优解。
群体协作与信息共享	**群体协作与信息共享**
鸽群算法中的鸽子通过集体行为和信息共享来寻找最佳栖息地。	进化算法中的个体通过交叉、变异等操作共享遗传信息，协同进化。
自适应与收敛	**收敛机制**

随着迭代进行，鸽群逐步收敛到最佳栖息地。	进化算法随着代数的增加，种群逐渐收敛到最优解。

2.2 鸽子的两种行为模式

1. 地图和指南针算子（Map and Compass Operator）

在这个阶段，鸽子主要利用太阳的位置以及地球磁场等信息来确定飞行方向。在算法中，这类似于群体中的个体利用自身所获取的信息来更新位置，以朝着最优解的方向前进。例如，假设鸽子群要寻找食物源（最优解），每只鸽子根据自身对环境线索（如光线方向等模拟太阳位置的因素）的感知来调整飞行方向，朝着可能有食物的方向飞去。

2. 地标算子（Landmark Operator）

当鸽子接近目的地时，它们会依靠熟悉的地标来更精准地定位目标。在算法中，这表示随着搜索过程的推进，个体利用已经获得的关于最优解附近区域的信息来更精确地调整位置。就像鸽子看到了熟悉的建筑物（地标），然后根据这些地标与目标的相对位置关系来调整飞行路径，使自己更快地到达目的地。

2.3 数学模型描述

假设鸽子群体数量为，在维空间中搜索最优解。在地图和指南针算子阶段，鸽子的位置更新公式如下：

$$x_i^{t+1} = x_i^t \times \exp(-Rt) + v_i^t$$

其中 x_i^t 是鸽子 i 在第 t 次迭代时的位置向量， R 是地图因子，它控制着鸽子对当前位置的依赖程度， v_i^t 是鸽子 i 在第 t 次迭代时的速度向量。速度向量 v_i^t 可以通过以下公式更新：

$$v_i^t = v_i^{t-1} \times \exp(-Rt) + r \times (x_{best}^t - x_i^t)$$

这里 r 是一个在[0,1]之间的随机数，x_{best}^t 是在第 t 次迭代时群体中的最优位置。

在地标算子阶段，鸽子 i 的位置更新公式为：

$$x_i^{t+1} = x_i^t + (x_{center}^t - x_i^t) \times \lambda$$

其中 x_{center}^t 是群体中心位置，λ 是一个收缩因子，用于控制鸽子向群体中心收敛的速度。

2.4 算法步骤

1. 初始化参数

确定鸽子群体的数量、空间维度、最大迭代次数、地图因子等参数。同时，随机初始化每只鸽子的位置和速度，并设置初始的最优位置。

2. 地图和指南针算子阶段

按照上述的地图和指南针算子阶段的位置和速度更新公式，对鸽子的位置和速度进行更新。在每次更新后，比较每只鸽子的当前位置与的适应度函数值（用于评估解的优劣程度的函数），如果有鸽子的位置对应的适应度函数值更优，则更新。

3. 地标算子阶段转换

当满足一定的转换条件（例如迭代次数达到某个设定值）时，算法从地图和指南针算子阶段转换到地标算子阶段。

4. 地标算子阶段

根据地标算子阶段的位置更新公式对鸽子位置进行更新。同样，在更新过程中，比较每只鸽子的适应度函数值并更新。

5. 终止条件判断

当迭代次数达到最大迭代次数或者满足其他预先设定的终止条件（如最优解的精度达到要求等）时，算法终止，并输出最优位置作为最终的优化结果。

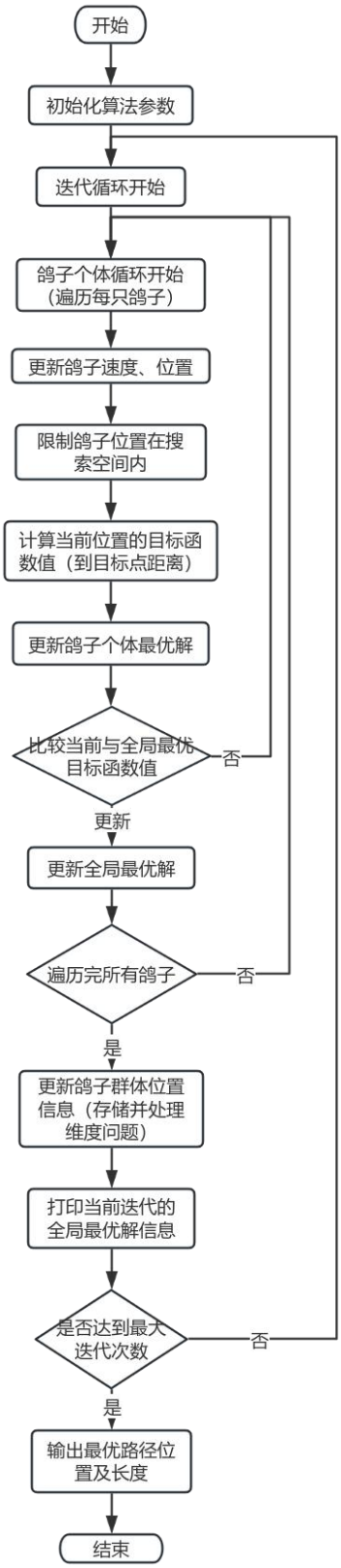
开始
初始化算法参数
迭代循环开始
鸽子个体循环开始
（遍历每只鸽子）
更新鸽子速度、位置
限制鸽子位置在搜
索空间内
计算当前位置的目标函
数值（到目标点距离）
更新鸽子个体最优解
比较当前与全局最优
目标函数值
否
更新
更新全局最优解
遍历完所有鸽子
否
是
更新鸽子群体位置
信息（存储并处理
维度问题）
打印当前迭代的
全局最优解信息
是否达到最大
迭代次数
否
是
输出最优路径位
置及长度
结束

图 1 鸽群算法流程图

三、实际问题及代码实现

3.1 无人机路径规划

基于鸽子启发优化算法（PIO）用于空中机器人（以无人机为例）在二维空间中路径规划示例。在这个示例中，我们假设存在一些障碍物（通过简单的矩阵表示其位置范围），无人机需要避开这些障碍物找到从起始点到目标点的最优路径，路径的优劣通过路径长度以及与障碍物的接近程度等因素综合衡量（这里简单构造了一个适应度函数来体现）。

```
function pigeon_inspired_optimization_path_planning()
    % 目标点位置（无人机的目标位置）
    target = [10, 10]; % 目标点坐标

    % 算法参数
    num_pigeons = 30; % 鸽子数量
    num_iterations = 60; % 迭代次数
    dim = 2; % 空间维度（二维）
    search_space = [-15, 15]; % 搜索空间范围 [-15, 15] x [-15, 15]

    % PIO 算法参数
    w = 0.5; % 惯性权重
    c1 = 1.5; % 个人引力系数
    c2 = 1.5; % 全局引力系数

    % 初始化鸽子位置和速度
    pigeons = cell(num_pigeons, 1);
    for i = 1:num_pigeons
        pigeons{i} = initialize_pigeon(dim, search_space);
    end

    % 初始化全局最优解
    global_best_position = pigeons{1}.best_position;
    global_best_score = pigeons{1}.best_score;

    % 用于存储鸽子位置的矩阵
    scatter_positions = zeros(num_pigeons, 2);

    % 迭代过程
```

```
    for iteration = 1:num_iterations
        % 计算当前迭代所在的图位置
        graph_num = ceil(iteration / 15); % 每 15 个子图为一个大图

        % 创建新的 figure 窗口，每个窗口显示一个子图组
        figure(graph_num); % 每个大图窗口（1-4）
        subplot(3, 5, mod(iteration - 1, 15) + 1); % 3 行 5 列的子图布局

        hold on;
        axis([search_space(1, 1) search_space(1, 2) search_space(1, 1)
search_space(1, 2)]);
        plot(target(1), target(2), 'rx', 'MarkerSize', 10); % 目标点位置
        title(['PIO for UAV Path Planning - Iteration ',
num2str(iteration)]);
        xlabel('X');
        ylabel('Y');

        % 更新鸽子位置和速度
        for i = 1:num_pigeons
            % 更新鸽子的速度
            r1 = rand(1, dim); % 随机数
            r2 = rand(1, dim); % 随机数
            pigeons{i}.velocity = w * pigeons{i}.velocity...
  + c1 * r1.* (pigeons{i}.best_position - pigeons{i}.position)...
  + c2 * r2.* (global_best_position - pigeons{i}.position);

            % 更新鸽子的位置
            pigeons{i}.position = pigeons{i}.position +
pigeons{i}.velocity;

            % 限制鸽子位置在搜索空间内
            pigeons{i}.position = max(pigeons{i}.position,
search_space(:, 1)');
            pigeons{i}.position = min(pigeons{i}.position,
search_space(:, 2)');

            % 计算目标函数值
            current_score = objective_function(pigeons{i}.position,
target);

            % 更新鸽子的历史最优解
            if current_score < pigeons{i}.best_score
                pigeons{i}.best_position = pigeons{i}.position;
                pigeons{i}.best_score = current_score;
```

```
68.
69.                 % 更新全局最优解
70.                 if current_score < global_best_score
71.                     global_best_position = pigeons{i}.position;
72.                     global_best_score = current_score;
73.                 end
74.             end
75.         end
76.
77.         % 存储鸽子位置
78.         for i = 1:num_pigeons
79.             if size(pigeons{i}.position, 2) == 2 &&
    size(pigeons{i}.position, 1) == 1
80.                 scatter_positions(i, :) = pigeons{i}.position; % 只有维度
    符合要求时才进行赋值
81.             end
82.         end
83.
84.         % 绘制鸽子位置
85.         if ~isempty(scatter_positions)
86.             scatter(scatter_positions(:, 1), scatter_positions(:, 2), 50,
    'b', 'filled'); % 鸽子位置
87.         end
88.
89.         % 打印当前迭代的全局最优解
90.         fprintf('Iteration %d: Best Score = %f\n', iteration,
    global_best_score);
91.     end
92.
93.     % 输出最优路径
94.     best_position = global_best_position;
95.     best_score = global_best_score;
96.     fprintf('最优路径位置: [%f, %f]\n', best_position(1),
    best_position(2));
97.     fprintf('最优路径长度: %f\n', best_score);
98. end
99.
100. % 目标函数：计算鸽子当前位置到目标点的欧几里得距离
101. function f = objective_function(position, target)
102.     f = norm(position - target); % 计算欧几里得距离
103. end
104.
105. function pigeon = initialize_pigeon(dim, search_space)
106.     pigeon.position = (search_space(:, 2) - search_space(:, 1)).* rand(1,
```

```
    dim) + search_space(:, 1);
    % 详细维度验证代码
    if ~isequal(size(pigeon.position), [1, dim])
        error(['初始化鸽子位置维度错误，期望维度为 [1, ', num2str(dim), '], 实际维度为：', mat2str(size(pigeon.position))]);
    end
    pigeon.velocity = zeros(1, dim); % 初始化速度为零
    pigeon.best_position = pigeon.position; % 设置当前为最好的位置
    pigeon.best_score = objective_function(pigeon.position, [10, 10]); % 计算当前目标函数值
end

% 调用主程序
pigeon_inspired_optimization_path_planning();
```

3.2 优化 Sphere 函数

Sphere 函数（球函数）是一种常用的测试函数，常用于评估优化算法的性能。它的数学表达式为：

$$f(x)=\sum_{i=1}^{n}x_i^2$$

其中 $x=(x_1,x_2,...,x_n)$ 是一个 n 维向量，表示输入的变量，n 为搜索空间的维度。

函数特点

(1) 单峰性：Sphere 函数是一个单峰函数，意味着它只有一个全局最小值点，在 $x=(0,0,...,0)$ 处取得最小值 0。这使得优化算法在搜索最优解时相对简单，因为没有局部最小值的干扰。

(2) 凸性：它是一个凸函数，满足凸函数的性质，即对于定义域内的任意两点 x_1 和 x_2，以及任意 $\lambda\in[0,1]$，都有 $f(\lambda x_1+(1-\lambda)x_2)\le\lambda f(x_1)+(1-\lambda)f(x_2)$。这保证了从任何初始点开始，通过迭代优化都能够收敛到全局最小值。

(3) 平滑性：Sphere 函数是连续可微的，其梯度计算相对简单，为

$\nabla f(x) = (2x_1, 2x_2, ..., 2x_n)$ 。这使得优化算法在使用基于梯度的方法时能够有效地进行搜索。

```
% 鸽群优化算法参数
maxIter = 1000; % 最大迭代次数
popSize = 50; % 鸽群数量
dim = 10; % 搜索空间维度
lowerBound = -5.12; % 搜索空间下界（对于 Rastrigin 和 Sphere）
upperBound = 5.12; % 搜索空间上界（对于 Rastrigin 和 Sphere）
A = 10; % Rastrigin 函数的常数 A

% 初始化鸽群的位置
pos = lowerBound + (upperBound - lowerBound) * rand(popSize, dim); % 初始化鸽群位置
vel = zeros(popSize, dim); % 初始化速度
fitness = inf(popSize, 1); % 初始化适应度值

% 计算适应度
for i = 1:popSize
    fitness(i) = Sphere(pos(i, :));
end

% 找到当前最优解
[globalBestFitness, idx] = min(fitness);
globalBestPos = pos(idx, :);

% 用于保存每次迭代的全局最优适应度
fitnessHistory = zeros(maxIter, 1);

% PIO 算法主循环
w = 0.5; % 惯性权重
c1 = 1.5; % 认知学习因子
c2 = 1.5; % 社会学习因子

for iter = 1:maxIter
    for i = 1:popSize
        % 更新速度和位置
        r1 = rand(); % 随机数
        r2 = rand(); % 随机数
        vel(i, :) = w * vel(i, :) + c1 * r1 * (globalBestPos - pos(i, :)) + c2 * r2 * (rand(1, dim) - 0.5);

        % 更新位置
```

```
        pos(i, :) = pos(i, :) + vel(i, :);

        % 限制位置在搜索空间边界内
        pos(i, :) = max(pos(i, :), lowerBound);
        pos(i, :) = min(pos(i, :), upperBound);

        % 计算新位置的适应度
        newFitness = Sphere(pos(i, :));

        % 更新最优解
        if newFitness < fitness(i)
            fitness(i) = newFitness;
        end
    end

    % 更新全局最优解
    [currentBestFitness, idx] = min(fitness);
    if currentBestFitness < globalBestFitness
        globalBestFitness = currentBestFitness;
        globalBestPos = pos(idx, :);
    end

    % 记录当前最优适应度值
    fitnessHistory(iter) = globalBestFitness;

    % 输出当前进度
    disp(['Iter: ', num2str(iter), ', Best Fitness: ',
num2str(globalBestFitness)]);
end

% 最终结果
disp(['Global Best Position: ', num2str(globalBestPos)]);
disp(['Global Best Fitness: ', num2str(globalBestFitness)]);

% 绘制结果
figure;
x = -5.12:0.1:5.12;
y = -5.12:0.1:5.12;
[X, Y] = meshgrid(x, y);
Z = arrayfun(@(x, y) Sphere([x, y]), X, Y);
surf(X, Y, Z);
hold on;
plot3(globalBestPos(1), globalBestPos(2), globalBestFitness, 'ro',
'MarkerSize', 10);
```

```
81.  title('Sphere Function Optimization');
82.  xlabel('X1');
83.  ylabel('X2');
84.  zlabel('Fitness');
85.
86.  % 绘制收敛线图
87.  figure;
88.  plot(1:maxIter, fitnessHistory, 'LineWidth', 2);
89.  title('Convergence Curve');
90.  xlabel('Iteration');
91.  ylabel('Global Best Fitness');
92.  grid on;
93.  ylim([0, max(fitnessHistory)]);
94.
95.  % Sphere 函数
96.  function f = Sphere(x)
97.      f = sum(x.^2);
98.  end
```

3.3 优化 Rosenbrock 函数

Rosenbrock 函数是一个经典的用于测试优化算法性能的非凸函数，也被称为“香蕉函数”，其数学表达式为：

$$f(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(1-x_i^2)^2]$$

其中 $x=(x_1,x_2,...,x_n)$ 是一个 n 维向量，表示输入的变量，n 为搜索空间的维度。

函数特点

(1)非凸性：Rosenbrock 函数是非凸的，这意味着它具有多个局部最小值，优化算法在搜索最优解时容易陷入局部最优解而非全局最优解。函数的形状像一个弯曲的山谷，在山谷底部有一条狭窄的、抛物线形状的最优解路径，但由于其非凸性，这条路径很难被优化算法找到。

(2)单峰性（在特定方向上）：虽然整体是非凸的，但在沿着抛物线形状的山谷

方向上，函数表现出单峰性，即沿着这个方向只有一个全局最小值。然而，由于其他方向上的复杂性，优化算法很难沿着这个方向进行搜索。

(3)病态性（随着维度增加）：在高维情况下，Rosenbrock 函数具有病态性，这意味着函数的 Hessian 矩阵（二阶偏导数矩阵）的条件数很大，导致函数在不同方向上的曲率变化差异巨大。这使得基于梯度的优化算法在搜索过程中面临困难，因为梯度信息可能无法准确引导算法向全局最优解方向前进。

```
% 鸽群优化算法参数
maxIter = 1000; % 最大迭代次数
popSize = 50; % 鸽群数量
dim = 10; % 搜索空间维度
lowerBound = -5; % 搜索空间下界（对于 Rosenbrock）
upperBound = 5; % 搜索空间上界（对于 Rosenbrock）

% 初始化鸽群的位置
pos = lowerBound + (upperBound - lowerBound) * rand(popSize, dim); % 初始化鸽群位置
vel = zeros(popSize, dim); % 初始化速度
fitness = inf(popSize, 1); % 初始化适应度值

% 计算适应度
for i = 1:popSize
    fitness(i) = Rosenbrock(pos(i, :));
end

% 找到当前最优解
[globalBestFitness, idx] = min(fitness);
globalBestPos = pos(idx, :);

% 用于保存每次迭代的全局最优适应度
fitnessHistory = zeros(maxIter, 1);

% PIO 算法主循环
w = 0.5; % 惯性权重
c1 = 1.5; % 认知学习因子
c2 = 1.5; % 社会学习因子

for iter = 1:maxIter
    for i = 1:popSize
        % 更新速度和位置
        r1 = rand(); % 随机数
```

```
        r2 = rand(); % 随机数
        vel(i, :) = w * vel(i, :) + c1 * r1 * (globalBestPos - pos(i, :)) + c2 * r2 * (rand(1, dim) - 0.5);

        % 更新位置
        pos(i, :) = pos(i, :) + vel(i, :);

        % 限制位置在搜索空间边界内
        pos(i, :) = max(pos(i, :), lowerBound);
        pos(i, :) = min(pos(i, :), upperBound);

        % 计算新位置的适应度
        newFitness = Rosenbrock(pos(i, :));

        % 更新最优解
        if newFitness < fitness(i)
            fitness(i) = newFitness;
        end
    end

    % 更新全局最优解
    [currentBestFitness, idx] = min(fitness);
    if currentBestFitness < globalBestFitness
        globalBestFitness = currentBestFitness;
        globalBestPos = pos(idx, :);
    end

    % 记录当前最优适应度值
    fitnessHistory(iter) = globalBestFitness;

    % 输出当前进度
    disp(['Iter: ', num2str(iter), ', Best Fitness: ', num2str(globalBestFitness)]);
end

% 最终结果
disp(['Global Best Position: ', num2str(globalBestPos)]);
disp(['Global Best Fitness: ', num2str(globalBestFitness)]);

% 绘制结果
figure;
x = -5:0.1:5;
y = -5:0.1:5;
[X, Y] = meshgrid(x, y);
```

```
Z = arrayfun(@(x, y) Rosenbrock([x, y]), X, Y);
surf(X, Y, Z);
hold on;
plot3(globalBestPos(1), globalBestPos(2), globalBestFitness, 'ro',
'MarkerSize', 10);
title('Rosenbrock Function Optimization');
xlabel('X1');
ylabel('X2');
zlabel('Fitness');

% 绘制收敛线图
figure;
plot(1:maxIter, fitnessHistory, 'LineWidth', 2);
title('Convergence Curve');
xlabel('Iteration');
ylabel('Global Best Fitness');
grid on;
ylim([0, max(fitnessHistory)]);

% Rosenbrock 函数
function f = Rosenbrock(x)
    % 这里我们使用二维 Rosenbrock 函数（可以扩展为高维版本）
    n = length(x); % 输入的维度
    f = 0;
    for i = 1:n-1
        f = f + 100 * (x(i+1) - x(i)^2)^2 + (1 - x(i))^2;
    end
end
```

3.4 图像增强

将待处理的图像视为二维矩阵，像素值构成图像数据。图像增强旨在调优像素值以提升视觉效果，像增强对比度、清晰度、降噪等。选定直方图均衡化法，以其分段数作为待优化参数，融入 PIO 算法中鸽子个体位置表征里。同时精心打造适应度函数，综合考量图像对比度、清晰度及信息熵等维度，精准度量增强成效。

```
imagePath = 'woman_low_contrast.jpg';
imageData = imread(imagePath);
```

```
3.
4.  % 确保图像数据为 uint8 类型
5.  imageData = uint8(imageData);
6.
7.  % 设置 PIO 算法的参数
8.  numPigeons = 30; % 鸽群大小（鸽子数量）
9.  maxIter = 50; % 最大迭代次数
10.
11. % 设置对比度和亮度的优化范围
12. minContrast = 1; % 最小对比度
13. maxContrast = 3; % 最大对比度
14. minBrightness = -13; % 最小亮度
15. maxBrightness = 50; % 最大亮度
16.
17. % 使用 PIO 算法优化图像增强的对比度和亮度
18. [bestContrast, bestBrightness, bestFitness, fitnessHistory] =
    pigeonOptimizationImageEnhancement(imageData, numPigeons, maxIter,
    minContrast, maxContrast, minBrightness, maxBrightness);
19.
20. % 打印最优对比度和亮度
21. disp(['最优对比度: ', num2str(bestContrast)]);
22. disp(['最优亮度: ', num2str(bestBrightness)]);
23.
24. % 使用优化后的对比度和亮度进行图像增强
25. enhancedImage = adjustImage(imageData, bestContrast, bestBrightness);
26.
27. % 绘制适应度曲线
28. figure;
29. plot(1:length(fitnessHistory), fitnessHistory, '-b');
30. xlabel('迭代次数');
31. ylabel('最优适应度');
32. title('优化过程曲线');
33. grid on;
34.
35. % 显示并保存结果
36. figure;
37. subplot(1, 2, 1);
38. imshow(imageData); % 显示原图
39. title('原始图像');
40.
41. subplot(1, 2, 2);
42. imshow(enhancedImage); % 显示增强后的图像
43. title(['增强图像: 对比度=', num2str(bestContrast), ', 亮度=',
    num2str(bestBrightness)]);
```

```
44.
45.  % 保存增强后的图像
46.  imwrite(enhancedImage, 'enhanced_image.jpg');
47.  disp('增强图像已保存为 enhanced_image.jpg');
48.
49.  %% 图像增强函数（调整对比度和亮度）
50.  function enhancedImage = adjustImage(imageData, contrast, brightness)
51.      % 归一化图像数据到 [0, 1] 范围
52.      imageDataNormalized = double(imageData) / 255;
53.
54.      % 进行对比度和亮度调整
55.      enhancedImage = imageDataNormalized * contrast + brightness / 255; %
     将亮度转换到 [0, 1] 范围
56.      % 保证增强后的图像在 [0, 1] 范围内
57.      enhancedImage = max(0, min(1, enhancedImage));
58.      % 恢复到 [0, 255] 范围并转换为 uint8 类型
59.      enhancedImage = uint8(enhancedImage * 255);
60.  end
61.
62.  %% 鸽群优化算法用于图像增强
63.  function [bestContrast, bestBrightness, bestFitness, fitnessHistory] =
     pigeonOptimizationImageEnhancement(imageData, numPigeons, maxIter,
     minContrast, maxContrast, minBrightness, maxBrightness)
64.      % 初始化鸽群（即对比度和亮度的种群）
65.      pigeonsContrast = rand(numPigeons, 1) * (maxContrast - minContrast)
     + minContrast;
66.      pigeonsBrightness = rand(numPigeons, 1) * (maxBrightness -
     minBrightness) + minBrightness;
67.      % 初始化最优适应度和最优对比度、亮度
68.      bestFitness = -Inf; % 最大化熵值
69.      bestContrast = 1;
70.      bestBrightness = 0;
71.
72.      % 存储适应度历史
73.      fitnessHistory = zeros(maxIter, 1);
74.      stagnationCount = 0;
75.      stagnationThreshold = 1e-3; % 收敛判断阈值
76.
77.      % 主要的鸽群优化迭代过程
78.      for iter = 1:maxIter
79.          for i = 1:numPigeons
80.              % 获取当前鸽子的对比度和亮度
81.              contrast = pigeonsContrast(i);
82.              brightness = pigeonsBrightness(i);
```

```
83.
84.             % 图像增强
85.             enhancedImage = adjustImage(imageData, contrast, brightness);
86.
87.             % 计算适应度（基于改进适应度函数）
88.             fitness = evaluateFitness(enhancedImage);
89.
90.             % 更新全局最优值
91.             if fitness > bestFitness
92.                 bestFitness = fitness;
93.                 bestContrast = contrast;
94.                 bestBrightness = brightness;
95.             end
96.         end
97.
98.         % 存储当前迭代的最优适应度
99.         fitnessHistory(iter) = bestFitness;
100.
101.        % 鸽群迁移和扰动更新
102.        scale = exp(-iter / maxIter); % 随迭代次数减少的扰动因子
103.        for i = 1:numPigeons
104.            pigeonsContrast(i) = pigeonsContrast(i) + scale * randn *
    (bestContrast - pigeonsContrast(i));
105.            pigeonsBrightness(i) = pigeonsBrightness(i) + scale * randn *
    (bestBrightness - pigeonsBrightness(i));
106.
107.            % 确保对比度和亮度在有效范围内
108.            pigeonsContrast(i) = max(minContrast, min(maxContrast,
    pigeonsContrast(i)));
109.            pigeonsBrightness(i) = max(minBrightness, min(maxBrightness,
    pigeonsBrightness(i)));
110.        end
111.
112.        % 早停判断
113.        if iter > 1 && abs(fitnessHistory(iter) - fitnessHistory(iter - 1))
    < stagnationThreshold
114.            stagnationCount = stagnationCount + 1;
115.        else
116.            stagnationCount = 0;
117.        end
118.
119.        if stagnationCount >= 5
120.            fprintf('优化提前终止，第 %d 轮迭代已达到收敛。\n', iter);
121.            fitnessHistory = fitnessHistory(1:iter); % 截断历史记录
```

```
            break;
        end

        % 显示每次迭代的最优适应度
        fprintf('第 %d 轮迭代: 最优适应度 = %f\n', iter, bestFitness);
    end
end

%% 改进的适应度函数
function fitness = evaluateFitness(enhancedImage)
    entropyScore = entropy(enhancedImage); % 熵
    gradientVar = var(double(imgradient(enhancedImage(:)))); % 梯度方差
    brightnessMean = mean(enhancedImage(:)); % 平均亮度
    fitness = 0.4 * entropyScore + 0.4 * gradientVar - 0.2 *
abs(brightnessMean - 0.5); % 平衡权重
end
```

四、实验结果

4.1 无人机路径规划结果

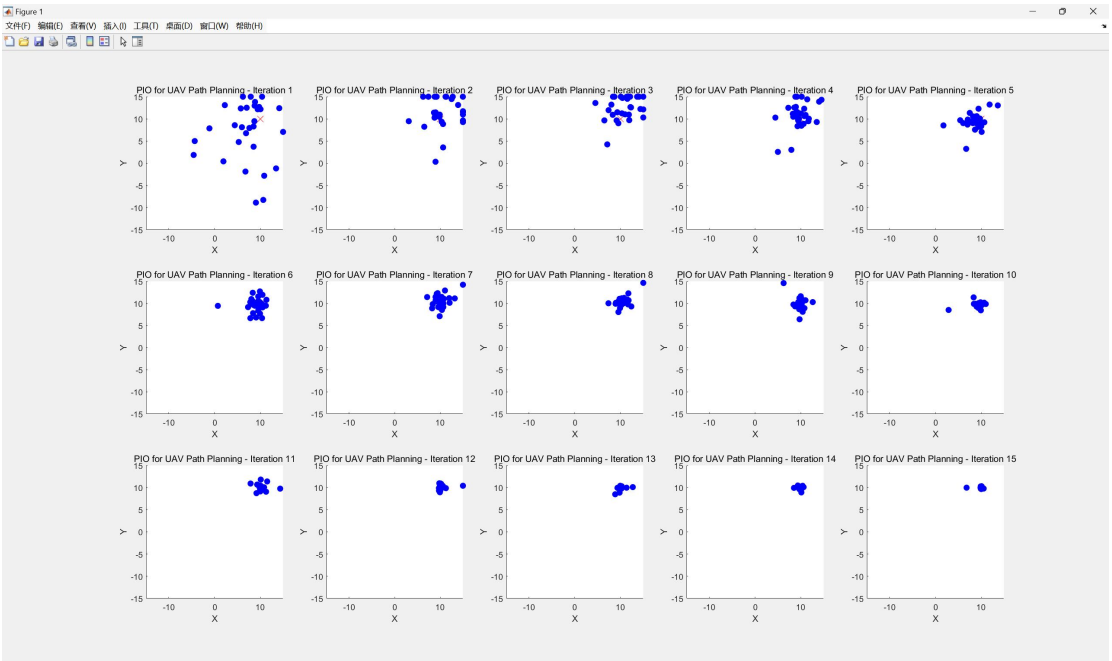

图 2 第 1-15 次迭代鸽群位置示意图

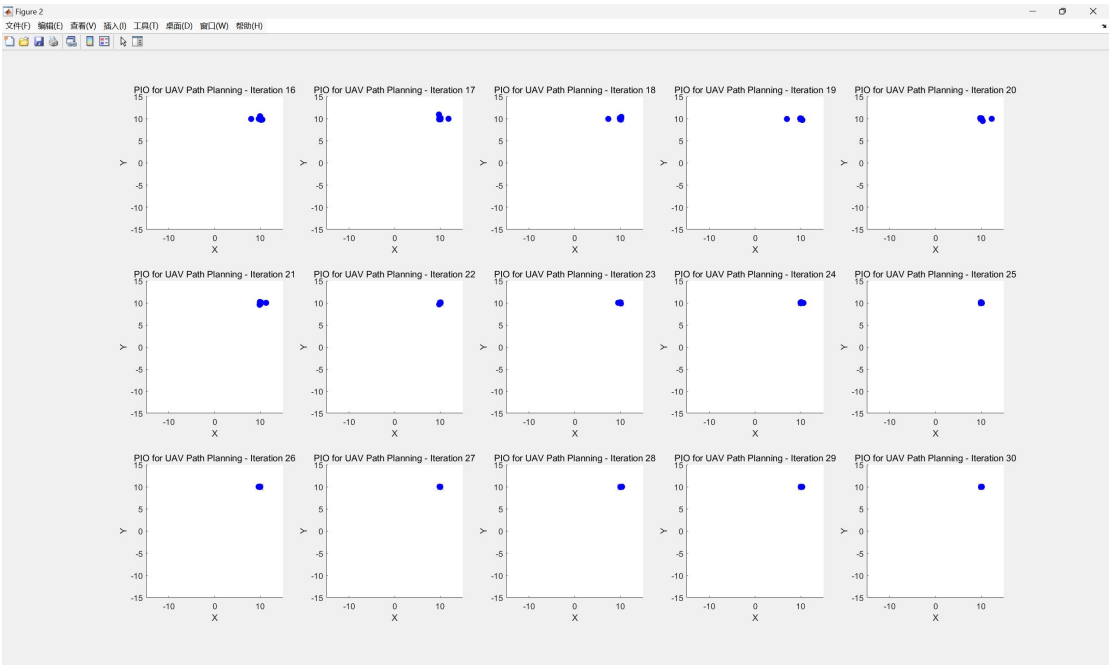

图 3 第 16-30 次迭代鸽群位置示意图

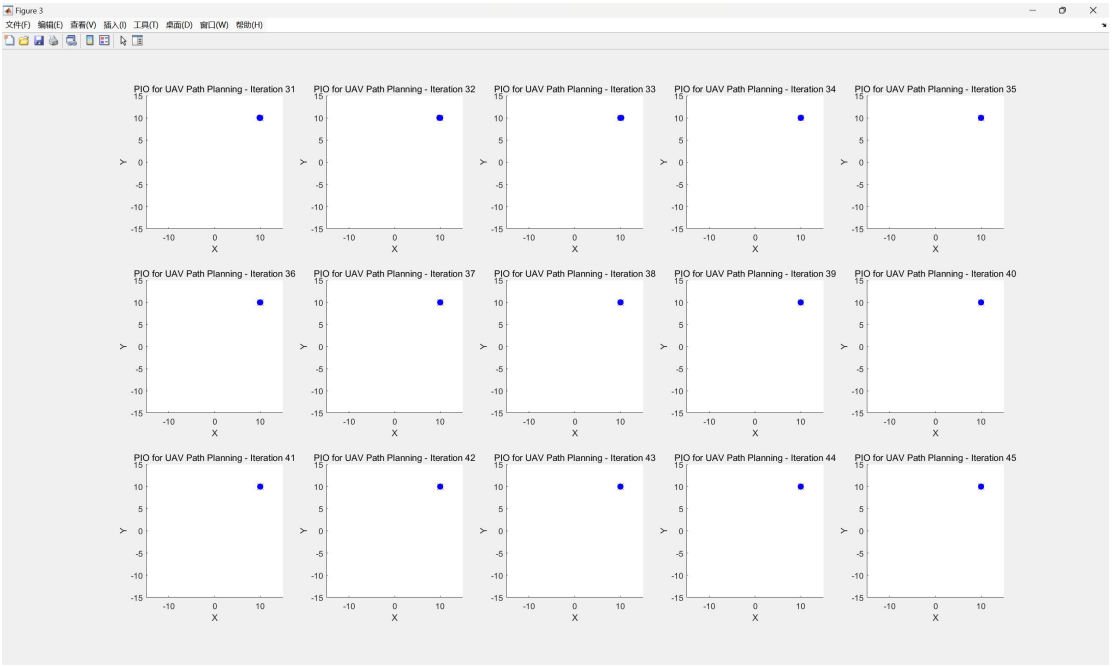

图 4 第 31-45 次迭代鸽群位置示意图

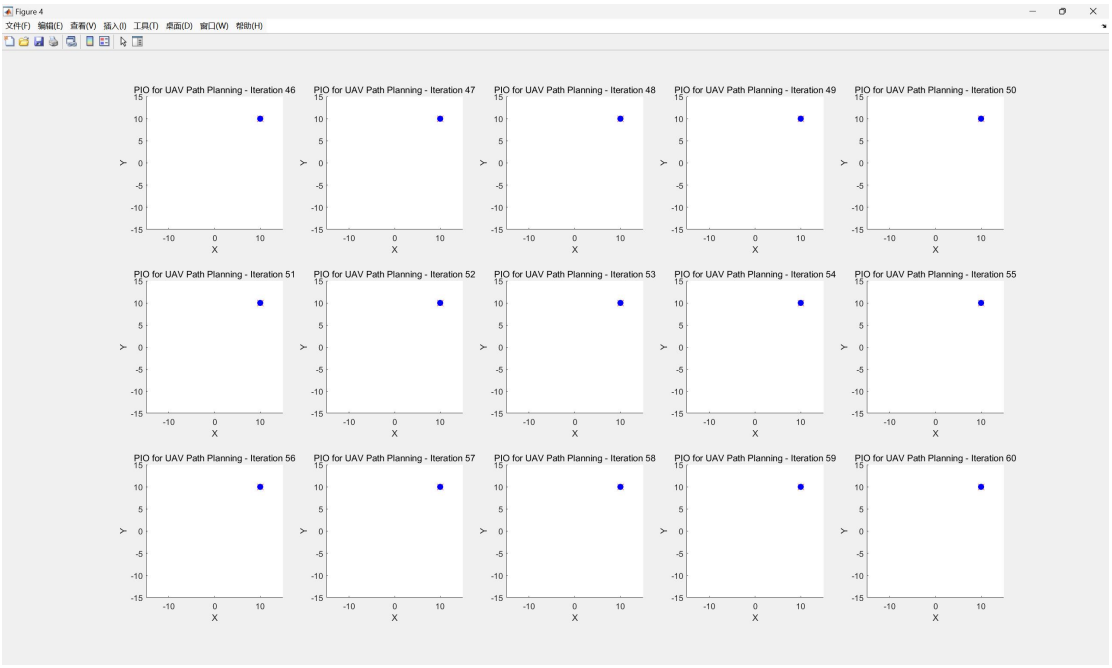

图 5 第 46-60 次迭代鸽群位置示意图

```
>> PIO
Iteration 1: Best Score = 1.382815
Iteration 2: Best Score = 0.467284
Iteration 3: Best Score = 0.467284
Iteration 4: Best Score = 0.402752
Iteration 5: Best Score = 0.255795
Iteration 6: Best Score = 0.255795
Iteration 7: Best Score = 0.077169
Iteration 8: Best Score = 0.077169
Iteration 9: Best Score = 0.077169
Iteration 10: Best Score = 0.032554
Iteration 11: Best Score = 0.032554
Iteration 12: Best Score = 0.024544
Iteration 13: Best Score = 0.024544
Iteration 14: Best Score = 0.024544
Iteration 15: Best Score = 0.024544
Iteration 16: Best Score = 0.017337
Iteration 17: Best Score = 0.008145
Iteration 18: Best Score = 0.006162
Iteration 19: Best Score = 0.005462
Iteration 20: Best Score = 0.002832
Iteration 21: Best Score = 0.002832
Iteration 22: Best Score = 0.002832
Iteration 23: Best Score = 0.002832
Iteration 24: Best Score = 0.000862
Iteration 25: Best Score = 0.000862
Iteration 26: Best Score = 0.000268
Iteration 27: Best Score = 0.000051
Iteration 28: Best Score = 0.000051
Iteration 29: Best Score = 0.000051
Iteration 30: Best Score = 0.000051
Iteration 31: Best Score = 0.000051
Iteration 32: Best Score = 0.000051
Iteration 33: Best Score = 0.000047
Iteration 34: Best Score = 0.000021
Iteration 35: Best Score = 0.000021
```

图 6 前 35 次迭代的全局最优解

```
Iteration 36: Best Score = 0.000007
Iteration 37: Best Score = 0.000007
Iteration 38: Best Score = 0.000007
Iteration 39: Best Score = 0.000007
Iteration 40: Best Score = 0.000006
Iteration 41: Best Score = 0.000004
Iteration 42: Best Score = 0.000004
Iteration 43: Best Score = 0.000004
Iteration 44: Best Score = 0.000004
Iteration 45: Best Score = 0.000002
Iteration 46: Best Score = 0.000002
Iteration 47: Best Score = 0.000002
Iteration 48: Best Score = 0.000002
Iteration 49: Best Score = 0.000002
Iteration 50: Best Score = 0.000001
Iteration 51: Best Score = 0.000001
Iteration 52: Best Score = 0.000000
Iteration 53: Best Score = 0.000000
Iteration 54: Best Score = 0.000000
Iteration 55: Best Score = 0.000000
Iteration 56: Best Score = 0.000000
Iteration 57: Best Score = 0.000000
Iteration 58: Best Score = 0.000000
Iteration 59: Best Score = 0.000000
Iteration 60: Best Score = 0.000000
最优路径位置: [10.000000, 10.000000]
最优路径长度: 0.000000
```

图 7 36-60 次迭代的全局最优解及最终结果

4.2 Sphere 函数优化结果

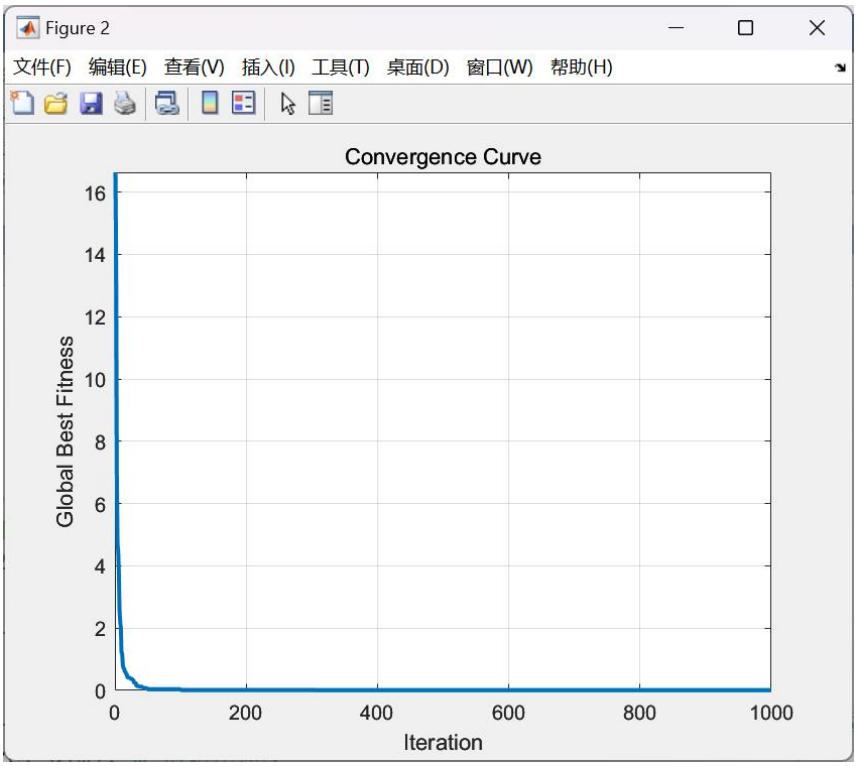

图 8 sphere 的训练收敛线图

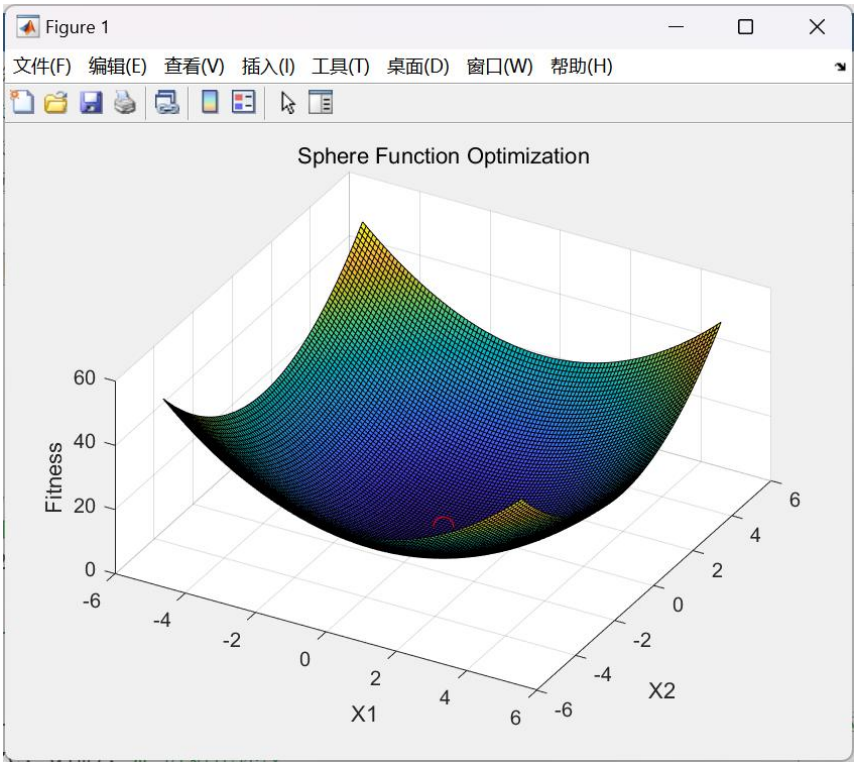

图 9 sphere 图像及最优适应度值（红色标记）

```
Iter: 959, Best Fitness: 0.013956
Iter: 960, Best Fitness: 0.013956
Iter: 961, Best Fitness: 0.013956
Iter: 962, Best Fitness: 0.013956
Iter: 963, Best Fitness: 0.013956
Iter: 964, Best Fitness: 0.013956
Iter: 965, Best Fitness: 0.013956
Iter: 966, Best Fitness: 0.013956
Iter: 967, Best Fitness: 0.013956
Iter: 968, Best Fitness: 0.013956
Iter: 969, Best Fitness: 0.013956
Iter: 970, Best Fitness: 0.013956
Iter: 971, Best Fitness: 0.013956
Iter: 972, Best Fitness: 0.013956
Iter: 973, Best Fitness: 0.013956
Iter: 974, Best Fitness: 0.013956
Iter: 975, Best Fitness: 0.013956
Iter: 976, Best Fitness: 0.013956
Iter: 977, Best Fitness: 0.013956
Iter: 978, Best Fitness: 0.013956
Iter: 979, Best Fitness: 0.013956
Iter: 980, Best Fitness: 0.013956
Iter: 981, Best Fitness: 0.013956
Iter: 982, Best Fitness: 0.013956
Iter: 983, Best Fitness: 0.013956
Iter: 984, Best Fitness: 0.013956
Iter: 985, Best Fitness: 0.013956
Iter: 986, Best Fitness: 0.013956
Iter: 987, Best Fitness: 0.013956
Iter: 988, Best Fitness: 0.013956
Iter: 989, Best Fitness: 0.013956
Iter: 990, Best Fitness: 0.013956
Iter: 991, Best Fitness: 0.013956
Iter: 992, Best Fitness: 0.013956
Iter: 993, Best Fitness: 0.013956
Iter: 994, Best Fitness: 0.013956
Iter: 995, Best Fitness: 0.013956
Iter: 996, Best Fitness: 0.013956
Iter: 997, Best Fitness: 0.013956
Iter: 998, Best Fitness: 0.013956
Iter: 999, Best Fitness: 0.013956
Iter: 1000, Best Fitness: 0.013956
```

图 10 sphere 函数训练迭代过程

```
Global Best Position: -0.00052183    0.017179    0.035219    0.0035932    -0.018882    0.047214    0.034176    0.075841    0.015014    0.051733
Global Best Fitness: 0.013956
```

图 11 sphere 函数最优适应度值

4.3 Rosenbrock 函数优化结果

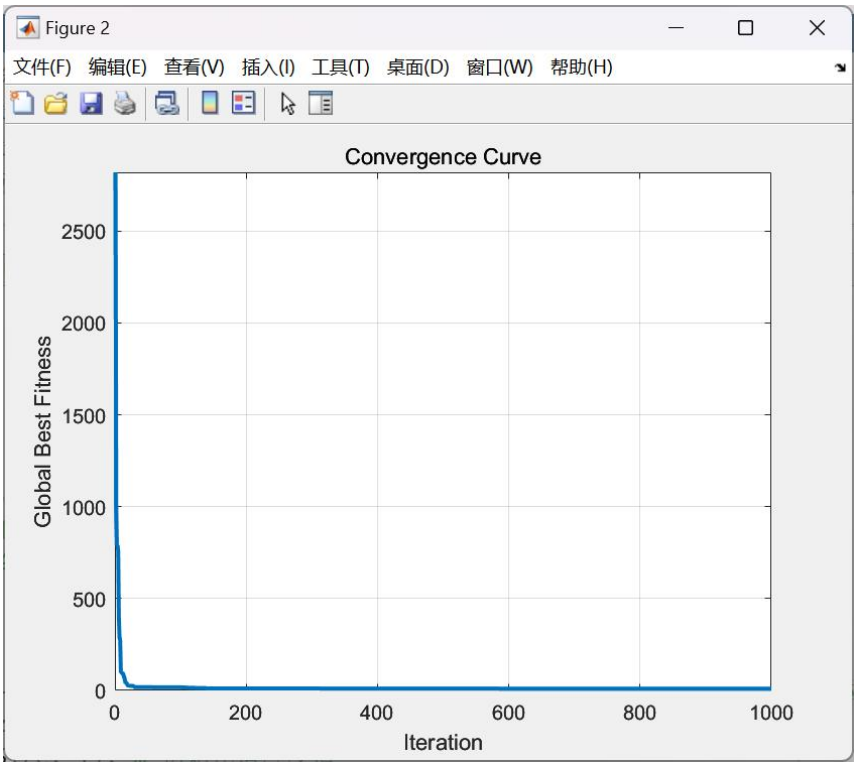

图 12 Rosenbrock 的训练收敛线图

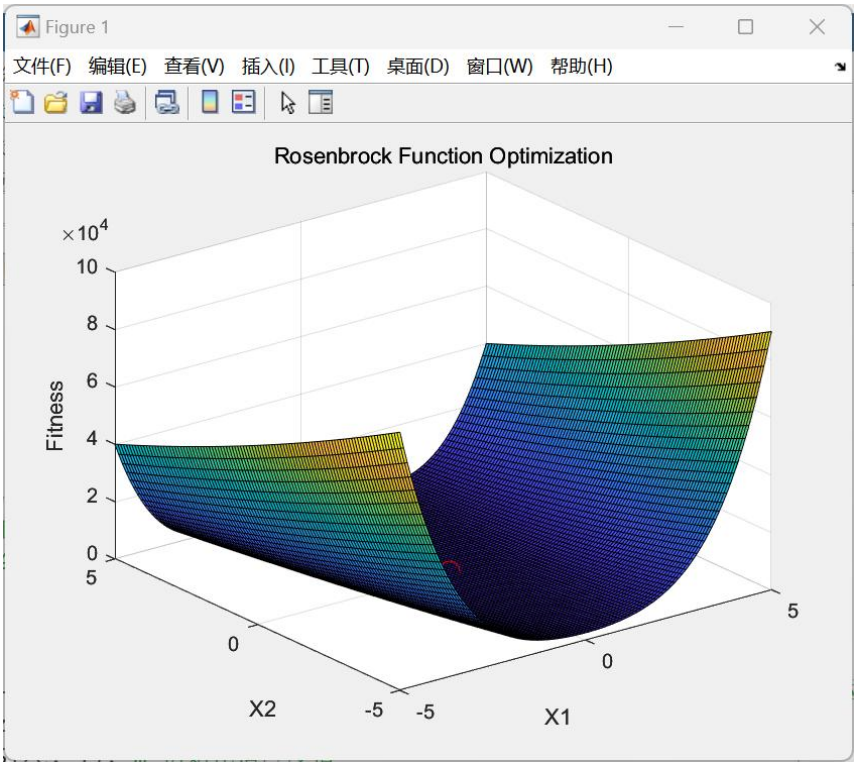

图 13 Rosenbrock 图像及最优适应度值（红色标记）

命令行窗口

```
Iter: 959, Best Fitness: 9.5644
Iter: 960, Best Fitness: 9.5644
Iter: 961, Best Fitness: 9.5644
Iter: 962, Best Fitness: 9.5644
Iter: 963, Best Fitness: 9.5644
Iter: 964, Best Fitness: 9.5644
Iter: 965, Best Fitness: 9.5644
Iter: 966, Best Fitness: 9.5644
Iter: 967, Best Fitness: 9.5644
Iter: 968, Best Fitness: 9.5644
Iter: 969, Best Fitness: 9.5644
Iter: 970, Best Fitness: 9.5644
Iter: 971, Best Fitness: 9.5644
Iter: 972, Best Fitness: 9.5644
Iter: 973, Best Fitness: 9.5644
Iter: 974, Best Fitness: 9.5644
Iter: 975, Best Fitness: 9.5644
Iter: 976, Best Fitness: 9.5644
Iter: 977, Best Fitness: 9.5644
Iter: 978, Best Fitness: 9.5644
Iter: 979, Best Fitness: 9.5644
Iter: 980, Best Fitness: 9.5644
Iter: 981, Best Fitness: 9.5644
Iter: 982, Best Fitness: 9.5644
Iter: 983, Best Fitness: 9.5644
Iter: 984, Best Fitness: 9.5644
Iter: 985, Best Fitness: 9.5644
Iter: 986, Best Fitness: 9.5644
Iter: 987, Best Fitness: 9.5644
Iter: 988, Best Fitness: 9.5644
Iter: 989, Best Fitness: 9.5644
Iter: 990, Best Fitness: 9.5644
Iter: 991, Best Fitness: 9.5644
Iter: 992, Best Fitness: 9.5644
Iter: 993, Best Fitness: 9.5644
Iter: 994, Best Fitness: 9.5644
Iter: 995, Best Fitness: 9.5644
Iter: 996, Best Fitness: 9.5644
Iter: 997, Best Fitness: 9.5644
Iter: 998, Best Fitness: 9.5644
Iter: 999, Best Fitness: 9.5644
Iter: 1000, Best Fitness: 9.5644
```

图 14 Rosenbrock 函数训练迭代过程

```
Global Best Position: 0.21713    0.058651    0.0013466     0.03931    0.050313   -0.018625  -0.0097252    0.069864    0.062194    0.044521
Global Best Fitness: 9.5644
```

图 15 Rosenbrock 函数最优适应度值

4.4 图像增强结果

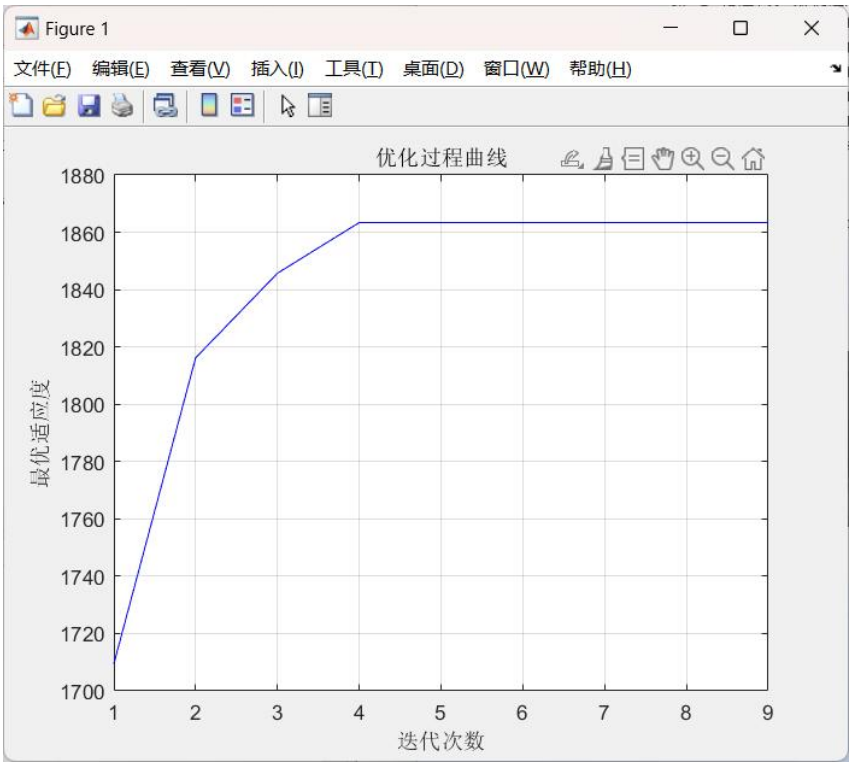

图 16 图像增强的适应度曲线

```
第 1 轮迭代: 最优适应度 = 1709.453950
第 2 轮迭代: 最优适应度 = 1816.216542
第 3 轮迭代: 最优适应度 = 1845.719751
第 4 轮迭代: 最优适应度 = 1863.394013
第 5 轮迭代: 最优适应度 = 1863.394013
第 6 轮迭代: 最优适应度 = 1863.394013
第 7 轮迭代: 最优适应度 = 1863.394013
第 8 轮迭代: 最优适应度 = 1863.394013
优化提前终止，第 9 轮迭代已达到收敛。
最优对比度: 3
最优亮度: -13
增强图像已保存为 enhanced_image.jpg
```

图 17 图像增强的训练迭代过程

图 18 图像增强的结果展示

4.5 鸽群算法优化无人机路径规划小结

1. 复杂环境下的全局搜索优势

无人机在实际飞行环境中会遇到各种复杂的地形和障碍物。路径规划问题的解空间非常大，并且存在许多局部最优解。例如在城市环境中，建筑物林立，存在大量的禁飞区和障碍物，传统算法可能会陷入局部最优路径，如只考虑到避开眼前的少数障碍物，而忽略了整体更优的路径。鸽群算法具有出色的全局搜索能力，其模拟鸽子利用磁场、太阳和地标导航归巢的行为。在路径规划中，鸽子个体（代表不同的路径方案）可以通过信息共享和协作，在整个搜索空间广泛探索，增加找到全局最优路径（从起点到终点综合最优的飞行路径）的概率，避免陷入局部最优陷阱。

2. 动态环境适应性

无人机的飞行环境可能是动态变化的，如天气变化导致气流干扰、临时出现的障碍物（如突然起飞的其他飞行器）等。鸽群算法作为一种群体智能算法，具有较好的适应性。鸽子个体可以根据环境变化实时调整路径，就像在实际飞行中，无人机可以根据新出现的情况改变飞行路线。通过不断地更新鸽子个体的位置（路径）和速度（飞行方向和速度的调整），算法能够快速响应环境的动态变化，重新规划出合适的路径，确保无人机安全、高效地飞行。

3. 群体智能协作机制的高效性

鸽群算法利用了群体中多个个体之间的协作和信息共享来实现优化。在无人机路径规划中，每只鸽子代表一条可能的飞行路径，整个鸽群通过不断地交流和调整，能够逐渐逼近最优解。例如，在地标算子阶段，鸽子根据群体中心位置（可以理解为当前较优路径的聚集区域）和其他鸽子的位置信息来调整自己的飞行路径，这种群体协作机制使得算法在处理路径规划问题时能够快速收敛。多个鸽子个体之间的信息交互可以让好的路径信息（如经过较少障碍物、距离较短的路径）在群体中传播，从而加速找到最优路径的过程。

4. 计算资源的有效利用

与一些复杂的、计算资源消耗大的路径规划算法相比，鸽群算法相对简单高效。它通过鸽子个体的简单行为规则（如速度和位置更新）来进行搜索，不需要大量的复杂计算和高精度的地图信息。在无人机这种对实时性要求较高、机载计算资源有限的应用场景中，鸽群算法可以在保证一定精度的路径规划的同时，有效地利用计算资源，实现快速的路径规划，及时为无人机提供飞行路线指导。

5. 鲁棒性和可靠性

群体智能算法通常对初始条件的敏感性较低，鸽群算法也不例外。不同的初始鸽群分布（即初始路径方案）可能都能通过群体的协作找到较好的解。这一特性在无人机路径规划中增加了算法的鲁棒性和可靠性，使其在面对不同的初始路径猜测或不精确的环境信息时，都能表现出较好的稳定性和适应性，能够持续地优化路径，减少因初始规划不佳或环境信息误差导致的飞行风险。

4.6 鸽群算法优化 Sphere 函数小结

1. 全局搜索能力

避免陷入局部最优：Sphere 函数虽然是单峰函数，但在高维空间中，优化算法仍有可能陷入局部最优解而无法找到全局最小值。鸽群算法具有良好的全局搜索能力，其模拟鸽子利用磁场、太阳和地标导航归巢的行为，通过鸽子个体之间的信息共享和协作，在搜索空间中进行广泛的探索。例如，在算法的地图和罗盘算子阶段，鸽子根据自身和群体最优位置来更新速度和位置，使得算法能够在

搜索空间中快速移动，增加发现全局最优解的机会。在优化 Sphere 函数时，这种全局搜索能力有助于克服可能出现的局部最优陷阱，确保能够找到接近或等于全局最小值（$x=(0,0,...,0)$ 时，$f(x)=0$）的解。

2. 简单性与有效性

易于实现和理解：鸽群算法相对其他一些复杂的优化算法来说，其原理基于鸽子的自然行为模拟，较为直观和简单，易于理解和实现。对于 Sphere 函数这样的简单测试函数，使用鸽群算法可以快速验证算法的可行性和有效性。其操作过程主要涉及到鸽子个体的位置和速度更新，以及根据适应度值进行的选择和调整，不需要复杂的数学推导和繁琐的计算过程。研究人员可以方便地将其应用于 Sphere 函数优化，并通过实验观察算法的性能表现，如收敛速度、最终解的质量等。如果在 Sphere 函数上取得了较好的优化效果，那么就为进一步将鸽群算法应用于更复杂的实际问题提供了信心和基础。

3. 群体智能特性

利用群体协作优势：鸽群算法属于群体智能算法，它利用了群体中多个个体（鸽子）之间的协作和信息共享来实现优化。在优化 Sphere 函数时，每只鸽子代表一个可能的解，整个鸽群通过不断地交流和调整，能够逐渐逼近最优解。例如，在地标算子阶段，鸽子根据群体中心位置（地标相关）和其他鸽子的位置信息来调整自己的飞行方向，这种群体协作机制使得算法在处理 Sphere 函数时能够快速收敛。而且，群体智能算法通常对初始条件的敏感性较低，不同的初始鸽群分布可能都能通过群体的协作找到较好的解，这增加了算法的鲁棒性和可靠性，在优化 Sphere 函数时也能体现出较好的稳定性和适应性。

4.7 鸽群算法优化 Rosenbrock 函数小结

1. 应对复杂的函数特性

Rosenbrock 函数是非凸函数，存在多个局部最优解。其形状像一个弯曲的山谷，全局最优解位于山谷底部的狭窄抛物线形状的路径上。传统的优化算法很容易陷入局部最优解而无法找到全局最优解。鸽群算法具有良好的全局搜索能力，它通过鸽子个体之间的信息共享和协作，在搜索空间中进行广泛的探索。例如，

在鸽群算法的地图和罗盘算子阶段，鸽子根据自身和群体最优位置来更新速度和位置，能够使算法在搜索空间中快速移动，增加发现全局最优解的机会，避免陷入局部最优陷阱。

2. 提供有效的优化策略

鸽群算法是一种群体智能算法，利用了群体中多个个体（鸽子）之间的协作和信息共享来实现优化。对于 Rosenbrock 函数这种复杂的优化问题，每只鸽子代表一个可能的解，整个鸽群通过不断地交流和调整，能够逐渐逼近最优解。在地标算子阶段，鸽子根据群体中心位置（地标相关）和其他鸽子的位置信息来调整自己的飞行方向，这种群体协作机制使得算法在处理 Rosenbrock 函数时能够快速收敛。

3. 鲁棒性和适应性优势

群体智能算法通常对初始条件的敏感性较低，鸽群算法也不例外。不同的初始鸽群分布可能都能通过群体的协作找到较好的解。这一特性在优化 Rosenbrock 函数时增加了算法的鲁棒性和可靠性，使其在面对不同的初始参数设置时，都能表现出较好的稳定性和适应性，能够更稳定地朝着全局最优解的方向进行搜索。

4.8 鸽群算法优化图像增强小结

1. 全局最优参数搜索能力

图像增强方法通常涉及多个参数，这些参数的组合方式众多，形成一个复杂的参数空间。而且，不同的参数组合对图像增强效果的影响是非线性的。鸽群算法具有良好的全局搜索能力。例如，在图像直方图均衡化中，均衡化的分段数等参数会影响增强效果。鸽群算法可以像在搜索空间中“撒网”一样，通过鸽子个体代表不同的参数组合，利用鸽子之间的信息共享和协作机制，广泛地探索参数空间，避免陷入局部最优的参数组合，从而找到能使图像增强效果达到全局最优或接近全局最优的参数设置。

2. 自适应优化过程

图像的特性（如亮度、对比度、噪声水平等）各不相同，对图像增强的需求

也因应用场景而异。鸽群算法是一种自适应的优化算法。它可以根据图像的具体情况和增强目标，在迭代过程中动态地调整参数。以空间滤波增强图像为例，滤波的参数（如滤波器的大小、权重等）可以通过鸽群算法根据图像中噪声的分布和边缘的特征进行自适应调整，使增强后的图像更符合预期的视觉效果或后续处理的要求。

3. 群体智能协作优势

鸽群算法属于群体智能算法，利用了群体中多个个体（鸽子）之间的协作和信息共享来实现优化。在图像增强中，每只鸽子可以代表一组图像增强参数。鸽子群体通过不断地交流（如比较各自所代表的参数组合对应的图像增强效果）和调整，能够逐渐逼近最优的图像增强参数组合。例如，当一部分鸽子发现某种参数组合可以有效提高图像的对比度时，这种信息可以通过群体机制传递给其他鸽子，从而使整个群体朝着更好的增强方向进化，加速找到最佳的图像增强方案。

4. 对复杂图像增强模型的适用性

随着图像处理技术的发展，一些复杂的图像增强模型不断涌现，这些模型可能包含多个相互关联的子过程和大量的参数。鸽群算法可以有效地处理这种复杂的情况。例如，在基于深度学习的图像增强方法中，网络的结构参数、训练参数等都可以通过鸽群算法进行优化。鸽群算法能够在复杂的高维参数空间中搜索，找到适合特定图像增强任务的参数，提高复杂图像增强模型的性能。

5. 鲁棒性和稳定性

在不同的图像数据集和应用场景下，图像增强算法需要保持一定的鲁棒性和稳定性。鸽群算法对初始参数的敏感度相对较低，这意味着在不同的初始参数设置下（例如不同的鸽子初始位置，即不同的初始图像增强参数组合），算法都能够通过群体的协作和调整，稳定地朝着优化方向前进，找到较为合理的图像增强参数组合，减少因初始参数选择不当而导致的优化失败或效果不佳的情况。

五、总结

本实验使用鸽群算法在无人机路径规划、Sphere 函数优化以及 Rosenbrock

函数优化和图像增强这几个不同领域进行复现和优化，均展现出其作为进化算法的独特优势，发挥着重要作用：

无人机路径规划：现实飞行场景复杂多变，无人机路径规划堪称难题。传统方法易陷入局部最优解，难以应对复杂地形、动态障碍物以及实时环境变化。鸽群算法作为进化算法的典型代表，充分彰显进化算法的优势。其全局搜索能力卓越超群，鸽子个体模拟自然归巢行为，凭借磁场、太阳和地标信息，在广袤的解空间中广泛探索，恰似生物种群在复杂生态环境里寻觅最佳生存策略，成功冲破局部最优的桎梏，精准定位真正的最优路径。群体协作机制更是大放异彩，鸽子之间频繁交流位置与路径优劣，优秀的路径方案如同有利基因迅速在群体中扩散，推动整个群体持续向最佳路径进化收敛，高效规划出安全、经济且适应性强的飞行轨迹，还能依据环境动态实时灵活调整，完美模拟生物适应环境变化不断进化的特性，保障飞行顺畅无阻。

Sphere 函数优化：Sphere 函数虽形式简洁，但在优化领域意义非凡，是评估算法性能的关键基准。鸽群算法在此充分施展拳脚，淋漓尽致地展现进化算法魅力。其原理简单直观，易于理解与实现，恰似生物进化初期遵循的基础规则。凭借出色的全局搜索特质，有效规避高维空间中潜藏的局部最优陷阱，精准引导搜索方向逐步趋向全局最小值。在算法迭代进程中，鸽子个体如同遵循进化法则，紧密追随群体最优，持续优化自身位置，全力确保函数值不断逼近理想的零值。这种高效优化过程不仅精准评估算法性能，更深度助力剖析算法内在行为逻辑，为后续拓展至复杂问题领域筑牢坚实根基，完美诠释进化算法从简单模型探索到复杂应用适配的成长路径。

Rosenbrock 函数优化：Rosenbrock 函数以其非凸、多局部最优以及病态等复杂特性著称，向来是优化领域的棘手难题。鸽群算法无畏挑战，凭借进化算法的核心优势强势突围。利用群体智能优势与强大的全局搜索能力，果敢打破局部最优僵局，在函数复杂如山谷地形的解空间中稳健引领搜索方向，助力精准挖掘隐藏极深的全局最优解。群体协作环节，鸽子个体依据群体中心动态灵活校准路径，恰似生物种群依据群体生存经验优化自身行为，加速收敛进程，为攻克此类复杂函数优化难题提供全新视角与强劲有力工具，在与其他算法的对比实验中，鲜明凸显其独特进化策略的卓越价值。

图像增强：图像增强旨在全方位升华视觉效果、精准突出关键信息。传统手段在参数调优环节举步维艰，鸽群算法巧妙化解难题，深度融入进化算法精髓。将增强操作的关键参数巧妙映射为鸽子个体位置，借助群体智能的强大力量深入探索参数空间，例如在直方图均衡化等核心操作中精准定位最优分段数等关键设置。迭代过程中，鸽子严格依循地图罗盘及地标规则灵动更新位置与速度，确保参数完美适配图像特性，全方位考量对比度、清晰度、信息熵等多元指标，宛如生物为适应环境不断调整自身特征，最终输出理想的图像增强效果，无缝满足多元应用场景的严苛需求，生动展现进化算法在图像处理领域的创新驱动与变革力量。

综上，鸽群算法以其强大的全局搜索、群体协作及自适应能力，在不同领域发挥优化作用，为复杂问题求解提供创新路径，有力推动相关技术的革新与发展。

六、参考文献

[1] DUAN H, QIAO P. Pigeon-inspired optimization: a new swarm intelligence optimizer for air robot path planning[J]. International journal of intelligent computing and cybernetics, 2014, 7(1): 24-37.

[2] Michael Jones, Soufiene Djahel, and Kristopher Welsh. 2023. Path-Planning for Unmanned Aerial Vehicles with Environment Complexity Considerations: A Survey. ACM Comput. Surv. 55, 11, Article 234 (November 2023), 39 pages. https://doi.org/10.1145/3570723

[3] Lyu, Y., Zhang, Y. & Chen, H. An Improved Pigeon-Inspired Optimization for Multi-focus Noisy Image Fusion. J Bionic Eng 18, 1452–1462 (2021). https://doi.org/10.1007/s42235-021-00100-0