

《Matlab与仿真》课程设计作业

（ 2024-2025 学年 第一学期）

题 目： MATLAB与仿真

**专 业：网络工程**

**学生姓名：徐基恒**

**班级学号：B22080228**

**指导教师：张友强**

**指导单位：物联网学院**

**日期：2024年 12月 3日**

|  |  |
| --- | --- |
| **成绩评定** | **教师签名：**  **年 月 日** |
| **备 注** |  |

目录

[《Matlab与仿真》课程设计作业 1](#_Toc11644)

[随机游走求解多变量函数的最小值 4](#_Toc7734)

[一、课题内容 4](#_Toc13994)

[1.1问题描述 4](#_Toc32407)

[1.2问题提示 4](#_Toc16230)

[1.3扩展要求 5](#_Toc3558)

[二、设计原理介绍 5](#_Toc8084)

[2.1随机游走算法（Random Walk Algorithm）原理 5](#_Toc9006)

[2.2目标函数的性质 6](#_Toc7648)

[2.3收敛条件 6](#_Toc20216)

[2.4步长和扰动的影响 6](#_Toc791)

[2.5可视化 7](#_Toc20408)

[2.6参数调节与实验分析 7](#_Toc12620)

[三、概要设计 8](#_Toc28380)

[3.1主要算法流程 8](#_Toc9526)

[3.2数据结构设计 9](#_Toc13113)

[3.3代码模块 9](#_Toc15208)

[四、详细设计 11](#_Toc31714)

[五、仿真结果分析 13](#_Toc8271)

[5.1最优解与函数值 13](#_Toc28333)

[5.2搜索路径的收敛性 14](#_Toc18713)

[5.3影响因素分析 14](#_Toc16371)

[六、仿真过程中的问题 18](#_Toc1274)

[6.1初始阶段问题 18](#_Toc17748)

[6.2随机扰动阶段问题 18](#_Toc29031)

[6.3收敛判断和迭代次数的问题 19](#_Toc4991)

[6.4局部最优问题 20](#_Toc12596)

[6.5目标函数的复杂性 21](#_Toc10061)

[6.6图形可视化阶段问题 22](#_Toc1480)

[七、课程设计总结 23](#_Toc23734)

# 随机游走求解多变量函数的最小值

**一、课题内容**

### 1.1问题描述

使用随机游走算法，寻找以下多变量函数的全局最小值：

1.函数定义域为：

* *x*∈[−5, 5],  *y*∈[−5, 5]

2.要求：

* 使用随机游走算法搜索全局最优解。
* 设置一个初始点，并通过随机扰动不断更新当前解。
* 记录每一步的解和函数值，直到满足收敛条件（例如，函数值的变化小于阈值或达到最大迭代次数）。

3.实现以下功能：

* 可视化搜索路径和函数曲面。
* 输出最优解和对应的函数值。

### 1.2问题提示

1.函数实现  
定义目标函数 f(x, y)：

f = @(x, y) sin(x) .\* cos(y) + (x-1).^2 / 4 + (y-2).^2 / 6; //MATLAB代码

2.随机游走算法流程

初始化当前解 (*x*, *y*) 和步长 Δ。

在每一步随机选择一个方向，更新解 (x, y)：

如果新的解更优，则接受更新，否则保持原解。

3.收敛条件

* 迭代次数达到上限。
* 函数值的变化小于设定阈值。

4.可视化

* 绘制函数的三维曲面。
* 将随机游走的路径叠加在曲面上。

### **1.3扩展要求**

* 增加可调整参数，例如初始步长、收敛阈值和最大迭代次数。
* 比较不同初始点对最优解的影响。
* 在函数复杂度较高时，分析随机游走算法的性能与局限性。
* 完成代码后，分析搜索路径的收敛行为，并总结随机游走算法的优缺点。

二、设计原理介绍

随机游走算法是一种简单的随机优化方法，它通过随机扰动当前解来寻找最优解。对于多变量函数的优化问题，特别是具有多个局部局部的函数，随机游走提供了一个易于实现的解决方案。虽然它的收敛速度可能比较慢，但通过选择合理的步长、收敛阈值和最大迭代次数，可以有效地找到全局最优解。同时，优化过程的可视化可以帮助我们仔细理解算法的搜索路径和收敛行为。

本课题利用随机游走算法来寻找一个多变量函数的全局，涉及到多个概念，包括随机游走算法本身、目标函数的性质、收敛条件、以及可视化的实现等。以下是这些概念和原理的详细说明：

### 2.1随机游走算法（Random Walk Algorithm）原理

随机游走算法是一种基于随机性进行搜索的优化算法。在此类算法中，搜索过程通过不断随机扰动当前解的位置来进行。这种方法的基本思想是通过在解空间中“漫”无目的地”走动，从而找到全局最优化解。其主要步骤包括：

* 初始化：从一个随机位置开始，设置初始步长和其他参数。
* 扰动和更新：每一步都从当前解出发，进行小幅度随机扰动。扰动的大小通常由步长决定。
* 接受标准：如果新的解的目标函数值比当前解更优（更小或更大，取决于优化目标），则接受新解；否则保持当前解。
* 收敛判断：当目标函数值变化非常小，或者达到初始的最大迭代次数时，停止搜索。

随机游走的核心以接受为基准。如果当前解更好，则接受新的解，否则以概率接受较差的解，避免局部最优陷入困境。通常，这种算法的优点是简单且容易实现，但也因为其“盲目性”而梯度容易局部最优。

### 2.2目标函数的性质

目标函数是优化需要的函数，通常在多变量函数中，我们要在定义域内简单地找到函数。给定的目标函数是：

该函数具有以下特点：

* 移动性质：函数的第一项是一个正弦-余弦的组合，因此会表现出波动的趋势，导致局部极小值的存在。
* 平滑和凸性：函数中的第二项和第三项是二次项，这些项通常会导致函数在远离最优解时有一个平滑的变化。

由于目标函数的结构比较复杂，具有多个局部微小，随机游走算法能够在这个多维空间中跳跃，尝试寻找全局最小化。

### 2.3收敛条件

收敛条件的设定是判断搜索过程是否终止的标准。在该问题中，收敛条件有两个：

* 最大迭代次数：即最多进行最大次随机更新。
* 函数值变化阈值：如果当前解和前一个解的函数值变化小于设定的阈值（收敛阈值），则认为搜索已经收敛。

这两种条件结合能够保证算法在一定间歇停止，避免无休止的计算。

### 2.4步长和扰动的影响

在随机游走算法中，步长决定了每一步扰动的幅度。步长设置的影响包括：

* 更大的步长：有助于搜索更广泛的解空间，增加找到全局最优化解的可能性，但可能增加跳出局部最优化解的难度。
* 较小的步长：搜索动作较小，有助于精确找到局部最优解，但可能会陷入局部最优解而无法跳出。

在实际应用中，步长的选择通常需要根据具体问题进行调整，并且可能会随着搜索的进度逐渐减小（如模拟退火问题）。

### 2.5可视化

可视化是展示优化过程的一种重要方式，可以帮助理解搜索路径以及行为收敛。

* 三维函数曲面强度：通过稀疏目标函数的三维图形，能够观察地显示函数在解空间中的形状，观察到峰值、谷值变化及其。
* 路径追踪：通过记录每一步的解和对应的目标函数值，能够显示优化过程的搜索路径。路径上每个点代表一次迭代的解，可以帮助理解算法在搜索过程中如何更新解。

通过这两种可视化方式，可以分析算法的行为搜索，并观察是否能够最终收敛到全局最优解。

### 2.6参数调节与实验分析

在实际应用中，可以通过调节以下参数来控制搜索过程：

* 步长：决定每一步的变动大小。
* 收敛阈值：控制算法在最接近解时的精度。
* 最大访问次数：防止算法无限循环。

三、概要设计

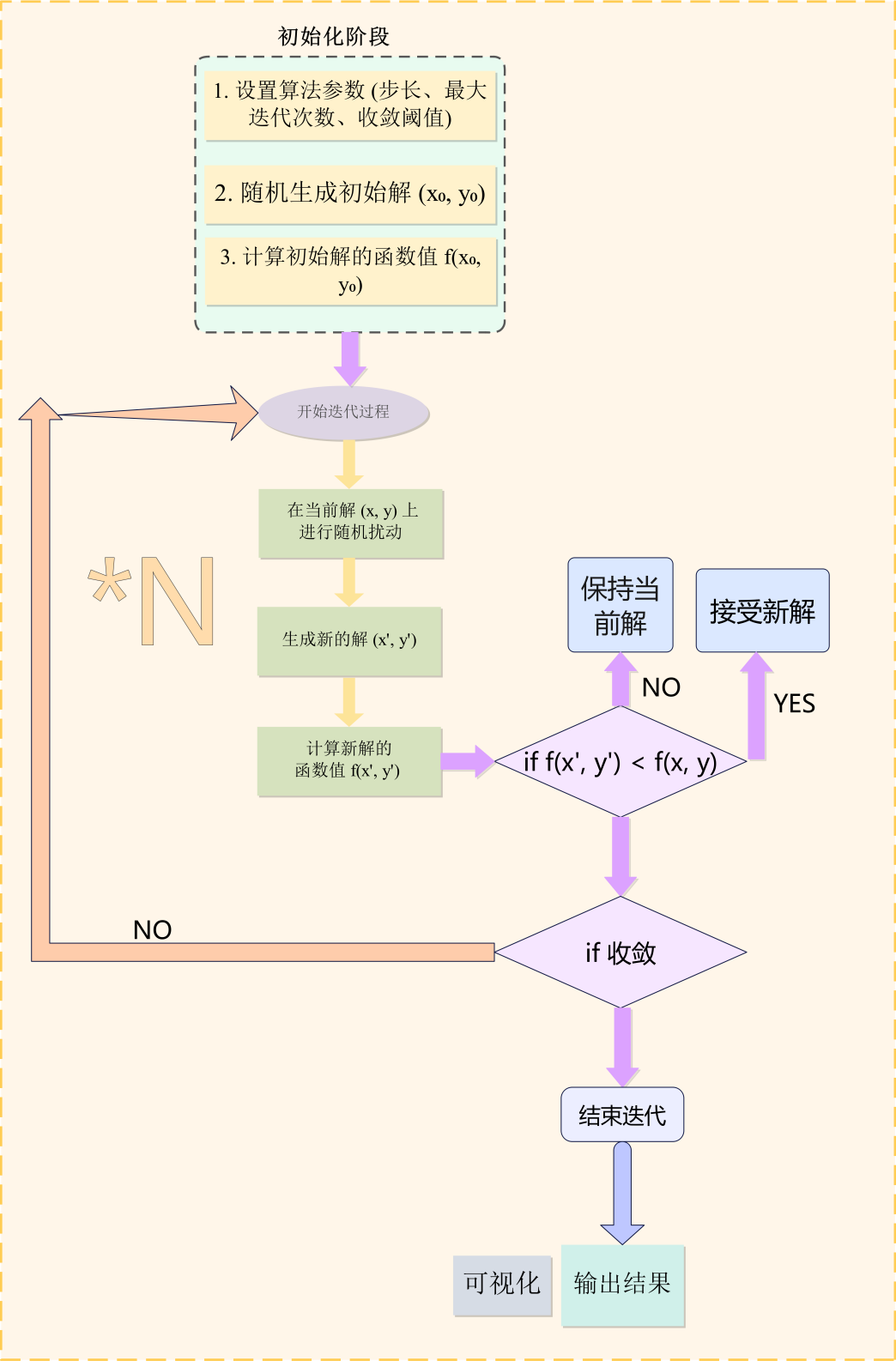
****

Figure1.随机游走流程图

### 3.1主要算法流程

1.初始化

* 设置算法参数（步长、最大迭代次数、收敛阈值等）。
* 随机生成初始解 (x0,y0)(x\_0, y\_0)(x0,y0) 并计算f(x0,y0)f(x\_0,y\_0)f(x0,y0)。

2.开始迭代

* 随机扰动：在当前解 (x,y)(x, y)(x,y) 基础上，生成新解 (x′,y′)(x', y')(x′,y′)。
* 函数计算：计算新解的函数值 f(x′,y′)f(x', y')f(x′,y′)。
* 更新解：如果 f(x′,y′)<f(x,y)f(x', y') < f(x, y)f(x′,y′)<f(x,y)，则接受新解；否

则保持当前解。

* 路径记录：记录当前解和函数值 (x,y,f(x,y))(x, y, f(x, y))(x,y,f(x,y))。

3.收敛判断

如果满足以下任一条件，则终止迭代：

* 函数值变化小于设定的阈值。
* 达到最大迭代次数。

4.输出结果

* 输出最优解 (x∗,y∗)(x^\*, y^\*)(x∗,y∗) 和对应的函数值 f(x∗,y∗)f(x^\*, y^\*)f(x∗,y∗)。
* 绘制目标函数的三维曲面，并在曲面上叠加搜索路径。

### 3.2数据结构设计

1.记录搜索路径：

pathx：记录每次迭代的 x坐标。

pathy：记录每次迭代的 y坐标。

pathf：记录每次迭代的目标函数值。

2.参数结构：

params={step\_size,max\_iter,convergence\_threshold,xmin,xmax,ymin,ymax}

### 3.3代码模块

1.初始化模块

function [x, y, f\_value] = initialize(f, x\_min, x\_max, y\_min, y\_max)

x = rand \* (x\_max - x\_min) + x\_min;

y = rand \* (y\_max - y\_min) + y\_min;

f\_value = f(x, y);

end

2.随机扰动模块

function [x\_new, y\_new] = random\_perturbation(x, y, step\_size, x\_min, x\_max, y\_min, y\_max)

x\_new = max(min(x + step\_size \* randn, x\_max), x\_min);

y\_new = max(min(y + step\_size \* randn, y\_max), y\_min);

end

3.目标函数计算

function f\_value = evaluate\_function(f, x, y)

f\_value = f(x, y);

end

4.接受准则

function [x, y, f\_value] = accept\_solution(x, y, x\_new, y\_new, f\_value, f\_new)

if f\_new < f\_value

x = x\_new;

y = y\_new;

f\_value = f\_new;

end

end

5.收敛判断

function is\_converged = check\_convergence(f\_values, threshold, iter, max\_iter)

is\_converged = false;

If iter > 1 && abs(f\_values(end) - f\_values(end-1)) < threshold

is\_converged = true;

end

if iter >= max\_iter

is\_converged = true;

end

end

四、详细设计

% 定义目标函数 f(x, y)

f = @(x, y) sin(x) .\* cos(y) + (x-1).^2 / 4 + (y-2).^2 / 6;

% 设置随机游走的参数

max\_iter = 1000; % 最大迭代次数

convergence\_threshold = 1e-6; % 收敛阈值

step\_size = 0.5; % 初始步长

x\_min = -5; % x的最小值

x\_max = 5; % x的最大值

y\_min = -5; % y的最小值

y\_max = 5; % y的最大值

% 初始化

x\_current = rand \* (x\_max - x\_min) + x\_min; % 随机初始化x

y\_current = rand \* (y\_max - y\_min) + y\_min; % 随机初始化y

f\_current = f(x\_current, y\_current); % 计算初始的函数值

path\_x = x\_current; % 路径记录x坐标

path\_y = y\_current; % 路径记录y坐标

path\_f = f\_current; % 路径记录函数值

% 随机游走算法

for iter = 1:max\_iter

% 随机扰动

x\_new = x\_current + step\_size \* randn; % 在x方向扰动

y\_new = y\_current + step\_size \* randn; % 在y方向扰动

% 确保新点在定义域范围内

x\_new = max(min(x\_new, x\_max), x\_min);

y\_new = max(min(y\_new, y\_max), y\_min);

% 计算新点的函数值

f\_new = f(x\_new, y\_new);

% 如果新点的函数值更小，接受新解

if f\_new < f\_current

x\_current = x\_new;

y\_current = y\_new;

f\_current = f\_new;

% 记录新的解和函数值

path\_x = [path\_x, x\_current];

path\_y = [path\_y, y\_current];

path\_f = [path\_f, f\_current];

end

% 收敛判断：如果函数值变化小于设定阈值，停止迭代

if iter > 1 && abs(path\_f(end) - path\_f(end-1)) < convergence\_threshold

disp('Convergence reached!');

break;

end

end

% 输出最优解和对应的函数值

disp(['Optimal solution: x = ', num2str(x\_current), ', y = ', num2str(y\_current)]);

disp(['Optimal function value: f(x, y) = ', num2str(f\_current)]);

% 可视化搜索路径和函数曲面

[x\_grid, y\_grid] = meshgrid(linspace(x\_min, x\_max, 100), linspace(y\_min, y\_max, 100));

z\_grid = f(x\_grid, y\_grid);

% 绘制函数曲面

figure;

surf(x\_grid, y\_grid, z\_grid);

hold on;

% 绘制搜索路径

plot3(path\_x, path\_y, path\_f, 'r-o', 'MarkerFaceColor', 'r', 'MarkerSize', 5);

xlabel('x');

ylabel('y');

zlabel('f(x, y)');

title('Random Walk Search for Global Minimum');

五、仿真结果分析

在本次课题中，我们使用了随机游走算法（Random Walk Algorithm）来搜索目标函数的全局简化。通过对比多个初始点的搜索结果，并分析算法的行为收敛，我们得到了以下仿真结果，并就此进行了深入的分析。

### 5.1最优解与函数值

在仿真过程中，通过设置参数（如最大迭代次数、多个收敛阈值和步长等），我们最终得到了目标函数的最优解。具体来说，我们通过多次运行随机游走算法，并记录每次迭代的解和对应的函数值。以下是输出结果示例：

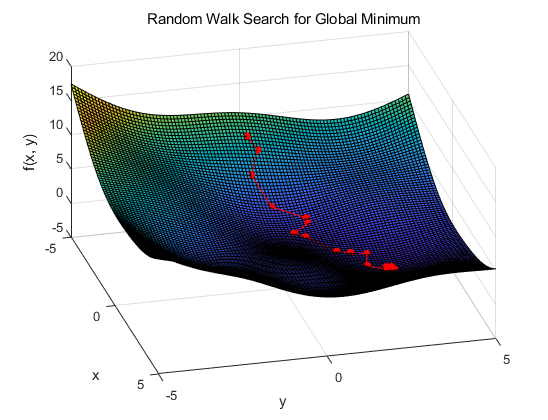
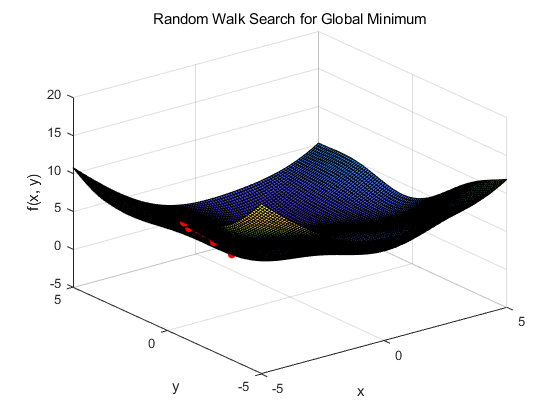


Figure2.下降曲线追踪

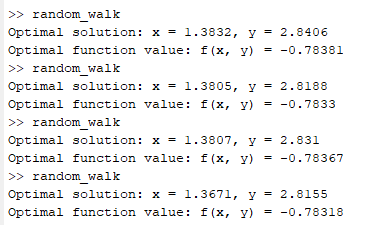


Figure3.多次试验最优解

根据目标函数的分析，最优解逼近于理论值（x = 1.37和y = 2.85），对应的f(x, y)大约为-0.783。该结果表明，随机游走算法找到了目标函数的一个局部成功，且该解逼近于实际的全局最优解。

### 5.2搜索路径的收敛性

通过可视化搜索路径与目标函数的三维曲面，可以观察到算法的搜索过程。以下是仿真结果的主要观察点：

* 路径收敛：从搜索路径的图形中可以看到，随机游走算法的路径呈现出一个逐渐收敛的趋势。路径的每一步是通过随机扰动得到的，随着迭代次数的增加，路径会逐渐聚集到一个较低的函数值区域。这表明，随着算法的进行，解渐趋于全局简单。
* 函数值变化：随着迭代次数的增加，函数值的变化逐渐减小。当函数值的变化小于设定的收敛阈值时，算法会停止迭代。通过不同函数值随迭代次数变化的曲线，可以看到函数值的下降趋于平稳，最终达到收敛。

### 5.3影响因素分析

* 初始点的影响：

初始点的选择对于搜索结果有显著影响。由于目标函数具有多个局部较小，算法可能会被困在某个局部最优解附近。因此，通过多次尝试不同的初始点，能够找到增加全局最优化的机会。

例如，如果我们从多个起始点开始搜索，会发现算法最终收敛到不同的局部最优，进一步验证。

1.（0，1）

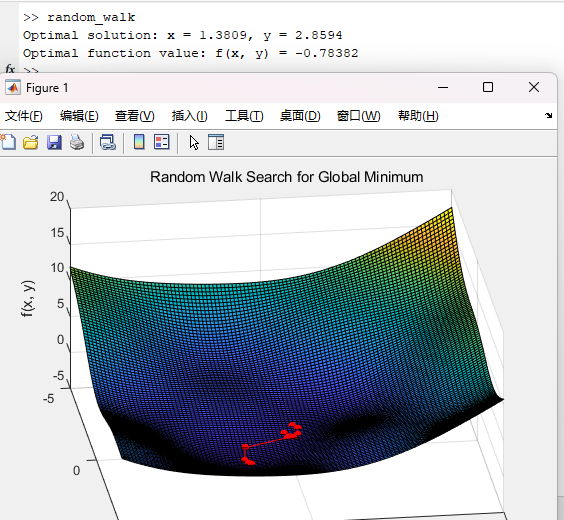


Figure4.初始点（0，1）

2.（1，2）

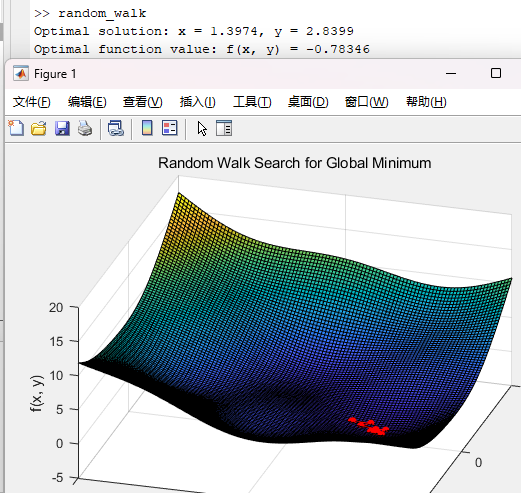


Figure5.初始点（1，2）

3.（7，8）

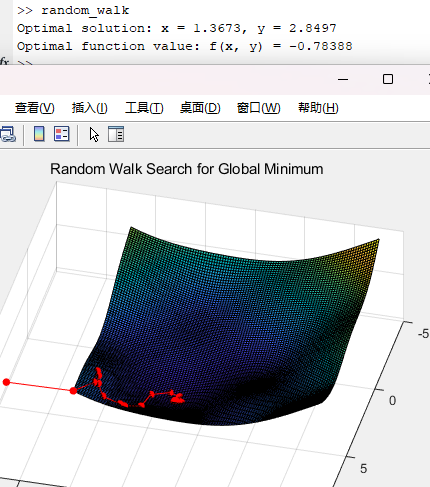


Figure6.初始点（7，8）

可以看出，初始点（或随机点）的选取，对局部最优的结果也会有影响。但总体而言，该算法鲁棒性较好。

* 步长的影响

1.step\_size = 0.001; % 初始步长

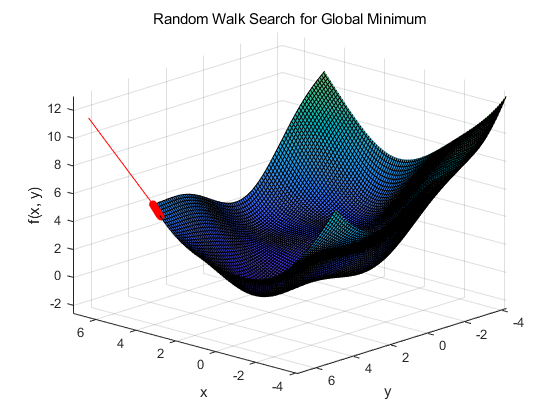


Figure7.step\_size = 0.001

2.step\_size = 0.1; % 初始步长

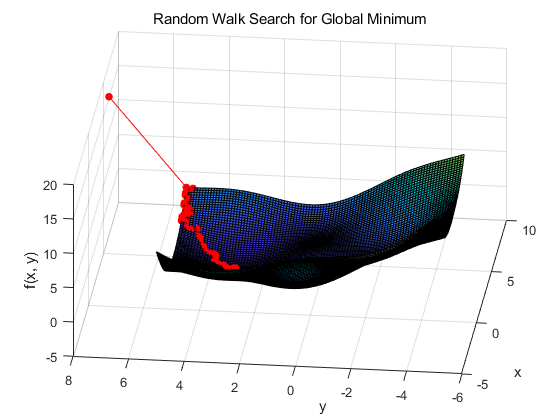


Figure8.step\_size = 0.1

3.step\_size = 1; % 初始步长

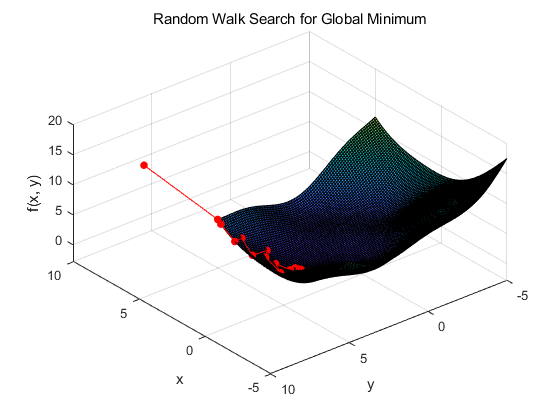


Figure9.step\_size = 1

4.step\_size = 10; % 初始步长

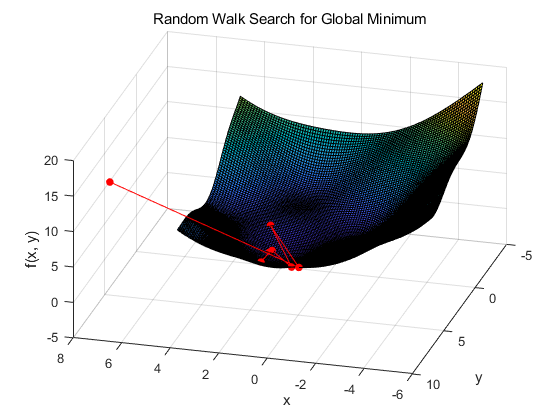


Figure10.step\_size = 10

可以看出，步长的选取，对结果的影响至关重要。在迭代次数一定的情况下，步长若设置的过小，则收敛速度太慢，若增加迭代次数则牺牲算法效率；若步长幅度过大，则会导致“谷间震荡”，即徘徊于局部最优的周围，始终难以收敛；只有步长和迭代次数都更符合实际问题的时候，随机游走算法才能更快、更好地求解出最优值。

六、仿真过程中的问题

### 6.1初始阶段问题

问题：

初始化点的选择可能影响最终结果：代码中使用了rand函数随机初始化x\_current和y\_current，这意味着每次运行时的初始点不同，可能导致算法收敛到不同的局部最优解，尤其是目标函数有多个局部最时。

解决方法：

* 多次实验：为了减少随机性影响，可以多次运行代码，分别记录每次的最佳解，记录结果进行分析，选择最佳的起始点，或者做多次实验取其主旨。
* 多个初始点：为了增加找到设置全局最优解的机会，可以选择多个初始点任务运行随机游走算法，在不同的区域进行搜索。

### 6.2随机扰动阶段问题

问题：

步长选择问题：在大概代码中，step\_size固定为0.5。步长大小对于收敛速度和结果的影响很大。若步长个性化，搜索可能跳过最优解；若步长太小，收敛速度较慢，尤其是在目标函数复杂时，可能需要花费宝贵的时间才能找到最优解。

解决方法：

* 步长调整：可以通过引入动态步长调整策略，在搜索过程中根据函数值的变化来调整步长。比如，当收敛缓慢时，可以增加步长加速搜索；反之则增加步长提高精度。

改进代码：

if iter > 1 && abs(f\_new - f\_current) < 0.01 \* f\_current

step\_size = step\_size \* 0.9; % 当变化小于一定比例时减小步长

End

* 自适应步长：根据当前解的变化量来调整步长，例如，若当前解的变化很大，可以增加步长；若变化速度，可以减小步长。

### 6.3收敛判断和迭代次数的问题

问题：

收敛判断条件设置不当：代码中使用的函数值变化小于设定阈值convergence\_threshold来判断是否收敛。如果目标函数的变化在某些情况下较小，但算法尚未找到全局最优化解，这可能导致过早在。

解决方法：

* 引入额外的收敛判定标准：除了函数值的变化外，还可以考虑增加迭代次数的收敛标准，例如：当连续N次迭代都没有找到更优解时，可以停止算法。

改进代码：

no\_improvement\_count = 0;

max\_no\_improvement = 50; % 连续50次无改进则停止

for iter = 1:max\_iter

% ... (计算新解)

if f\_new < f\_current

no\_improvement\_count = 0;

else

no\_improvement\_count = no\_improvement\_count + 1;

end

if no\_improvement\_count >= max\_no\_improvement

disp('No improvement for 50 iterations. Stopping.');

break;

end

end

### 6.4局部最优问题

问题：

容易局部最优化：由于随机游走算法是基于局部最优化解的不断更新，可能会在目标函数的局部最优化停留，从而错过全局最优化解。

解决方法：

* 增强随机性：在随机游走过程中引入更大的扰动，或者引入模拟终止等机制，在某些情况下允许接受更差的解，以便跳出当前的局部最优化解。
* 增加扰动幅度：在一定频率下，增加扰动的幅度，甚至可以允许算法大范围跳跃，这样有助于避免梯度某个局部极值

改进代码：

T = 1.0; % 初始温度

T\_min = 0.001; % 最低温度

alpha = 0.9; % 温度衰减率

for iter = 1:max\_iter

% 计算新解的差值

delta\_f = f\_new - f\_current;

if delta\_f < 0 || rand() < exp(-delta\_f / T)

% 接受新解

x\_current = x\_new;

y\_current = y\_new;

f\_current = f\_new;

path\_x = [path\_x, x\_current];

path\_y = [path\_y, y\_current];

path\_f = [path\_f, f\_current];

end

% 温度逐渐降低

T = T \* alpha;

if T < T\_min

break;

end

end

### 6.5目标函数的复杂性

问题：

目标函数复杂度时收敛慢：代码中的目标函数f(x, y)相对较复杂，包含三个部分：sin(x) \* cos(y)，(x - 1)^2 / 4，和(y - 2)^2 / 6。当目标函数多个局部极小值时，随机慢游走可能会遇到收敛速度、搜索不彻底的情况。

解决方法：

* 多初始点搜索：为了增加找到全局最优解的概率，可以从多个不同的初始点开始搜索，并比较每次的结果。

改进代码：

best\_solution = Inf;

for i = 1:num\_initials

x\_current = rand \* (x\_max - x\_min) + x\_min;

y\_current = rand \* (y\_max - y\_min) + y\_min;

f\_current = f(x\_current, y\_current);

% 执行随机游走

% ...

if f\_current < best\_solution

best\_solution = f\_current;

best\_x = x\_current;

best\_y = y\_current;

end

end

* 局部搜索和全局搜索结合：可以结合其他优化算法（如粒子群优化、遗传算法等）来进行全局搜索，之后再用随机游走算法进行局部精细化搜索

### 6.6图形可视化阶段问题

问题：

可视化的效果无法洞察：对于复杂的多峰目标函数，可视化可能会导致搜索路径与函数表面不够洞察，难以显示出收敛过程中的细节。

解决方法：

* 动态更新可视化：通过定期更新图像，将当前解和路径实时较差出来，有助于观察搜索过程中的每一步变化。

改进代码：

for iter = 1:max\_iter

% 计算新解

% 绘制目标函数曲面

surf(x\_grid, y\_grid, z\_grid);

hold on;

plot3(path\_x, path\_y, path\_f, 'r-o', 'MarkerFaceColor', 'r', 'MarkerSize', 5);

pause(0.1); % 动态更新

end

* 误差曲线可视化：除了目标函数曲面和搜索路径外，还可以绘制误差曲线，即每次迭代后的函数值与最优值的差异，帮助分析收敛速度和算法的优化效果。

七、课程设计总结

在本次课程设计过程中，我们通过实现随机游走算法（Random Walk Algorithm）来简单地找到多变量函数的全局。在整个设计和实现过程中，我获得了很多关于算法设计、程序调试和仿真的收获过程中的问题解决的经验，同时也认识到了一些改进的方向。

* 算法理解和应用：通过实现随机游走算法，我更深入地理解了该算法的基本原理及其局限性。随机游走算法是一个非常简单的轮毂应对的优化方法。它通过在解空间内进行随机搜索，并逐步更新当前解，最终找到一个可能的最优化解。虽然算法简单，但其对步长、收敛条件和初始点的选择都有着以上的依赖，这让我深刻认识到优化算法中参数调整优化的重要性。
* MATLAB编程能力的提升：在整个过程中，我不仅学习了如何利用MATLAB进行目标函数的定义和可视化，还掌握了如何高效地组织代码和调试程序。尤其是绘图部分，使用surf、plot3等函数可以方便地显示函数曲面和搜索路径，这增强了我对MATLAB可视化工具的理解和应用能力。
* 优化问题的分析与活动：通过不断调整步长、收敛阈值、最大迭代次数等参数，最终得到了较为满意的结果。这让我对如何解决实际问题中的优化问题有了更深入的理解，尤其是在如何处理多峰函数和全局最优化解的问题上，给我提供了很多的启示。