最优化理论与应用期末报告

中山大学

智能工程学院

姓名: 常毅成

学号: 22354010

邮箱: 2937212699@qq.com

题目: 作业 3

2025 年 6 月 25 日

摘　要

本报告围绕《最优化理论与方法》课程，在完成作业的基础上，深入探讨了多种优化理论及其 MATLAB 实现。报告首先通过**二次多项式拟合**问题，展示了最小二乘原理在数据建模中的应用。随后，详细阐述了**外点罚函数法**和**内点罚函数法**在处理带约束优化问题中的机制，特别是引入 Armijo 线搜索以增强外点罚函数法数值稳定性的实践。针对函数最大化问题，报告运用**遗传算法**进行全局寻优，体现了其在非凸问题中的优势。

报告最后进行了全面的课程总结，系统对比了**二次规划**、**罚函数方法**（外点与内点）、**乘子罚函数法**和**遗传算法**的核心思想、特点及适用场景，强调了基于梯度方法与启发式方法之间的异同。同时，结合个人学习体验，深入剖析了在理论理解、数值稳定性、方法选择与参数调优等方面的学习难点。展望未来，报告还设想了优化理论在机器学习、控制系统、资源调度、图像处理及实验设计等多个现实领域的广泛应用，展现了优化理论作为解决复杂工程与科学问题的强大工具的巨大潜力。

目录

1 二次多项式拟合

1.1 题目重述

1.1.1 问题背景与数学建模分析

本题要求利用 MATLAB 对给定的一组数据进行二次多项式拟合。给定的数据如下表所示：

编号	1	2	3	4	5	6	7	8	9	10	11
x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y	-0.447	1.978	3.28	6.16	7.08	7.34	7.66	9.56	9.48	9.30	11.2

任务是根据上述数据确定一个经验公式 $y = f(x)$，该公式应是一个二次多项式，并输出曲线拟合结果图。

数学建模: 二次多项式拟合的本质是寻找一个形如 $y = ax^2 + bx + c$ 的函数，使得该函数能够最佳地描述给定数据点 (x_i, y_i) 之间的关系。这里的“最佳”通常指的是在最小二乘意义下的最佳，即最小化所有数据点到拟合曲线的垂直距离的平方和。

具体而言，我们需要找到系数 a, b, c，使得以下残差平方和 S 最小：

$$S(a,b,c) = \sum_{i=1}^{n} (y_i - (ax_i^2 + bx_i + c))^2$$

其中 n 是数据点的数量。为了找到使 S 最小的 a, b, c 值，我们可以对 S 分别关于 a, b, c 求偏导，并令偏导数等于零，得到一个线性方程组（即正规方程组）。

$$\begin{cases} \frac{\partial S}{\partial a} = \sum_{i=1}^{n} 2(y_i - (ax_i^2 + bx_i + c))(-x_i^2) = 0 \\ \frac{\partial S}{\partial b} = \sum_{i=1}^{n} 2(y_i - (ax_i^2 + bx_i + c))(-x_i) = 0 \\ \frac{\partial S}{\partial c} = \sum_{i=1}^{n} 2(y_i - (ax_i^2 + bx_i + c))(-1) = 0 \end{cases}$$

整理后得到正规方程组：

$$\begin{pmatrix} \sum x_i^4 & \sum x_i^3 & \sum x_i^2 \\ \sum x_i^3 & \sum x_i^2 & \sum x_i \\ \sum x_i^2 & \sum x_i & n \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sum y_i x_i^2 \\ \sum y_i x_i \\ \sum y_i \end{pmatrix}$$

通过求解这个线性方程组，即可得到最佳拟合二次多项式的系数 a, b, c。

1.2 解题思路

1.2.1 算法原理

本题的核心算法原理是 ** 多项式最小二乘拟合 **。其基本思想与线性最小二乘法相同，都是通过最小化观测值与模型预测值之间的残差平方和来确定模型参数。对于多

项式拟合，即使原始关系是非线性的（如二次方），但待定系数（a, b, c）与自变量的幂次之间是线性的，因此仍然可以归结为线性最小二乘问题。

关键知识点:

1. **最小二乘原理:** 核心思想是选择模型参数，使观测数据点与模型预测值之间的平方误差之和达到最小。

2. **多项式拟合:** 是一种特殊类型的回归分析，用于拟合自变量 x 和因变量 y 之间的非线性关系，通过使用 x 的不同幂次项来构建模型。其一般形式为 $y = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$，其中 m 是多项式的次数。

3. **正规方程组:** 最小二乘问题通过求解正规方程组来获得解析解。这组线性方程组的解就是使得残差平方和最小的模型参数。

4. **数值稳定性:** 尽管正规方程组提供了解析解，但在实际数值计算中，尤其当数据量大或变量之间存在较强相关性时，直接求解正规方程组可能会面临数值稳定性问题（例如，系数矩阵可能接近奇异）。更稳健的方法（如 QR 分解、SVD 分解）通常在内部被用于求解此类问题，MATLAB 的 'polyfit' 函数就是基于这些更稳定的数值算法实现的。

1.2.2 具体解题流程

在 MATLAB 环境中，解决二次多项式拟合问题非常便捷，主要利用内置函数 'polyfit' 和 'polyval'。

1. **数据输入：** 将给定的 x 和 y 数据存储为 MATLAB 中的向量。
   - x 向量：'x = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0];'
   - y 向量：'y = [-0.447, 1.978, 3.28, 6.16, 7.08, 7.34, 7.66, 9.56, 9.48, 9.30, 11.2];'

2. **执行二次多项式拟合：** 使用 'polyfit(x, y, n)' 函数进行拟合。其中 'n' 表示多项式的次数，本题中为 2。该函数会返回拟合多项式的系数，按降幂排列。
   - 调用格式：'coefficients = polyfit(x, y, 2);'
   - 'coefficients' 将是一个包含三个元素的向量，例如 '[a, b, c]'，分别对应 $ax^2 + bx + c$ 中的 a, b, c。

3. **构建经验公式：**根据 'polyfit' 返回的系数，即可写出经验公式 $y = f(x) = \text{coefficients}(1) \cdot x^2 + \text{coefficients}(2) \cdot x + \text{coefficients}(3)$。

4. **生成拟合曲线数据：** 为了绘制平滑的拟合曲线，我们需要在 x 的取值范围内生成更密集的点，并使用 $polyval(coefficients, \text{x\_new})$ 函数计算这些点对应的 y 值。

- 生成新的 x 值范围：x_fit $= linspace(min(x), max(x), 100)$‘ (例如，生成 100 个点)
- 计算对应的 y 值：y_fit $= polyval(coefficients,$ x_fit)

5. **绘制拟合结果图：**使用 MATLAB 的绘图函数（如 ‘plot‘）将原始数据点和拟合曲线绘制在同一张图上，以便直观评估拟合效果。
   - 绘制原始数据点：$plot(x, y,' o')$ （使用圆圈标记）
   - 保持当前图：hold on
   - 绘制拟合曲线：$plot($x_fit, y_fit,$' -')$（使用实线）
   - 添加图例、标题和轴标签：legend(’ 原始数据’, ’ 二次拟合曲线’)title(’ 二次多项式拟合结果’); xlabel(’x’); ylabel(’y’)
   - 关闭 hold on 状态：hold off
6. **结果输出与分析：**
   - 输出拟合得到的系数 a, b, c。
   - 输出完整的经验公式 $y = f(x)$。
   - 对拟合图进行观察，判断拟合曲线是否较好地捕捉了原始数据的趋势。

1.3 解题代码与结果展示

```
% MATLAB 代码：二次多项式拟合

%% 1. 数据准备
% 定义给定的 x 值向量
x = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0];
% 定义给定的 y 值向量
y = [-0.447, 1.978, 3.28, 6.16, 7.08, 7.34, 7.66, 9.56, 9.48, 9.30, 11.2];

%% 2. 执行二次多项式拟合
% 使用 polyfit 函数进行二次（2次）多项式拟合
% polyfit(x, y, n) 返回多项式系数，其中 n 是多项式的次数。
% 对于二次多项式 y = ax^2 + bx + c，返回的 coefficients 将是 [a, b, c]。
coefficients = polyfit(x, y, 2);

% 从拟合结果中提取系数
a = coefficients(1); % 二次项系数
b = coefficients(2); % 一次项系数
c = coefficients(3); % 常数项系数

% 输出拟合得到的经验公式
fprintf('拟合得到的二次经验公式为：y = %.4f * x^2 + %.4f * x + %.4f\n', a, b, c);

%% 3. 生成拟合曲线数据
% 为了绘制平滑的拟合曲线，我们需要生成一系列新的、更密集的 x 值
% linspace(start, end, num) 在指定范围内生成等间距的 num 个点。
x_fit = linspace(min(x), max(x), 100);

% 使用 polyval 函数计算这些新的 x 值对应的 y 值（即拟合曲线上的点）
% polyval(coefficients, x_new) 使用给定系数的多项式计算 x_new 处的值。
y_fit = polyval(coefficients, x_fit);

%% 4. 绘制拟合结果图
% 创建一个新的图形窗口
figure;

% 绘制原始数据点
% 'o' 表示使用圆圈标记数据点
plot(x, y, 'o', 'DisplayName', '原始数据');

% 保持当前图形，以便在同一图中绘制多条曲线
hold on;

% 绘制拟合曲线
% '-' 表示使用实线绘制曲线
plot(x_fit, y_fit, '-', 'LineWidth', 1.5, 'DisplayName', '二次拟合曲线');

% 添加图例、标题和轴标签，使图表更具可读性
legend('show'); % 显示图例
title('二次多项式拟合结果'); % 设置图表标题
xlabel('x 值'); % 设置 x 轴标签
ylabel('y 值'); % 设置 y 轴标签
grid on; % 显示网格

% 释放当前图形的保持状态，防止后续绘图添加到当前图表中
hold off;

%% 5. 结果分析

 S_total = sum((y - mean(y)).^2); % 总平方和
 y_predicted = polyval(coefficients, x); % 预测的y值
 S_residual = sum((y - y_predicted).^2); % 残差平方和
 R_squared = 1 - S_residual / S_total; % 决定系数
 fprintf('决定系数 R-squared: %.4f\n', R_squared);
```

图 1: 题目一代码

```
>> task1
拟合得到的二次经验公式为: y = -9.8108 * x^2 + 20.1293 * x + -0.0317
决定系数 R-squared: 0.9671
```

图 2: 题目一结果

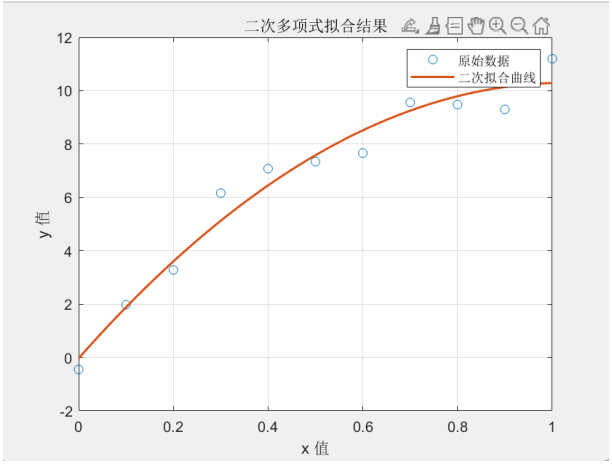

图 3: 二次多项式拟合结果

2 外点罚函数法

2.1 题目重述

2.1.1 问题背景与数学建模分析

本题要求编写 MATLAB 程序，利用外点罚函数法（Exterior Penalty Function Method）求解如下带等式约束的优化问题，要求精度为 10^{-8}，初始点为 $(10, -5)$。

$$\min f(x) = x_1^2 + 2x_2^2$$
$$\text{s.t.} \quad x_1 - x_2 = 0$$

数学建模: 这是一个典型的带等式约束的非线性优化问题。外点罚函数法的核心思想是将约束条件转换为惩罚项，并将其添加到原目标函数中，从而将原问题转化为一系列无约束优化问题进行求解。

对于等式约束 $h(x) = 0$，常用的罚函数形式为 $P(x) = (h(x))^2$。因此，针对本题，约束条件为 $h(x) = x_1 - x_2 = 0$。增广目标函数（或称罚函数）$F(x, c_k)$ 定义为:

$$F(x, c_k) = f(x) + c_k(h(x))^2 = x_1^2 + 2x_2^2 + c_k(x_1 - x_2)^2$$

其中 c_k 是罚因子（penalty parameter）。在迭代过程中，c_k 逐渐增大并趋向于无穷大。当 $c_k \to \infty$ 时，无约束优化问题 $\min F(x, c_k)$ 的最优解 $x^*(c_k)$ 将收敛到原约束优化问题的最优解 x^*。罚因子平方项的选择确保了罚函数的可微性，这对于基于梯度的无约束优化方法至关重要。

2.2 解题思路

2.2.1 算法原理

外点罚函数法的基本原理是通过惩罚违反约束的可行性来强制迭代点向可行域靠近。每次迭代，算法会解决一个无约束子问题 $\min F(x, c_k)$，其解作为下一次迭代的起始点。随着罚因子 c_k 的增大，违反约束的惩罚项在目标函数中的权重越来越大，使得算法被迫在越来越接近可行域的区域内寻找最优解。最终，当 c_k 足够大时，无约束问题的解将无限逼近原约束问题的最优解。

本算法内部循环求解无约束子问题时，采用 ** 梯度下降法 **，并结合 **Armijo 线搜索 ** 来动态调整步长。梯度下降法是一种一阶优化算法，通过沿着目标函数梯度的负方向移动来逐步逼近局部最小值。对于函数 $F(x)$，其更新规则为 $x^{new} = x^{current} - \alpha \nabla F(x^{current})$，其中 α 是学习率（步长）。

**Armijo 线搜索 ** 是一种非精确线搜索方法，旨在找到一个合适的步长 α，使得在迭代方向上，目标函数值有足够的下降。它避免了固定步长可能导致的震荡或发散问题，尤其是在罚因子 c_k 增大导致增广目标函数曲率急剧变化时，线搜索能够提供更强的数值稳定性。

关键知识点:

1. **罚函数构造:** 将约束问题转化为无约束问题，通过添加惩罚项。
2. **罚因子更新策略:** 罚因子 c_k 递增趋于无穷，以加强对约束违反的惩罚。通常采用 $c_{k+1} = \beta c_k$ 的形式，其中 $\beta > 1$。
3. **无约束优化子问题求解:** 每个外层迭代都需要解决一个无约束优化问题。本题选用梯度下降法。
4. **梯度下降法:** 通过计算目标函数的梯度，并沿负梯度方向移动来迭代逼近局部最小值。
5. **Armijo 线搜索:** 动态调整梯度下降步长，确保每次迭代有足够的下降量，并提高数值稳定性，避免因固定步长过大而导致的数值溢出或发散。
6. **收敛判据:** 外点罚函数法通常以外层迭代解的收敛性（如两次迭代解的范数差小于阈值）和约束违反程度（如 $|h(x)|$ 小于阈值）作为停止条件。

2.2.2 具体解题流程

1. **初始化：**
   - **初始点：** $x^{(0)} = (10, -5)$。
   - **初始罚因子：** $c_0 = 1$ (可以根据问题调整)。

- **罚因子增长率：** $\beta = 5$ (通常取 $5 \sim 10$ 之间的值)。
- **整体收敛精度：** $\epsilon = 10^{-8}$。
- **最大外层迭代次数：** max_outer_iter $= 500$。
- **内层（梯度下降）收敛容差：** inner_tolerance $= 1e-6$ (可以略宽松于 ϵ)。
- **最大内层（梯度下降）迭代次数：** $max_inner_iter = 5000$ (为线搜索预留足够迭代次数)。
- **Armijo 线搜索参数：**
  - 初始尝试步长：$\alpha_{init} = 1$。
  - 步长衰减因子：$\rho = 0.5$ $(0 < \rho < 1)$。
  - Armijo 条件参数：$\sigma = 0.0001$ (很小的正数，通常 $0 < \sigma < 1$)。
  - 最大线搜索迭代次数：max_line_search_iter $= 100$。

2. **外层循环（$k = 0, 1, 2, \ldots$）：**

(a) **构造增广目标函数 $F(x, c_k)$：** $F(x_1, x_2, c_k) = x_1^2 + 2x_2^2 + c_k(x_1 - x_2)^2$。

(b) **计算增广目标函数的梯度 $\nabla F(x, c_k)$：**

$$\frac{\partial F}{\partial x_1} = 2x_1 + 2c_k(x_1 - x_2) \frac{\partial F}{\partial x_2} = 4x_2 - 2c_k(x_1 - x_2)$$

(c) **求解无约束子问题：**从当前点 $x^{(k)}$ 开始，使用梯度下降法结合 Armijo 线搜索求解 $\min F(x, c_k)$，得到近似解 x^*_{inner}。

- **内层循环（梯度下降）：**

i. 计算当前点 $x^{current}$ 处的梯度 $\nabla F(x^{current}, c_k)$ 及其范数平方 $||\nabla F||_2^2$。

ii. **Armijo 线搜索：**
  - 初始化步长 $\alpha = \alpha_{init}$。
  - 循环：
    * 计算候选新点 $x^{new} = x^{current} - \alpha \nabla F(x^{current}, c_k)$。
    * 计算 $F(x^{new})$ 和 $F(x^{current})$。
    * **检查 Armijo 条件：**如果 $F(x^{new}) \leq F(x^{current}) - \sigma\alpha||\nabla F(x^{current})||_2^2$，且 $F(x^{new})$ 不是 'NaN' 或 'Inf'，则接受当前 α。
    * 否则，将 α 衰减：$\alpha = \alpha \cdot \rho$，并继续尝试，直到满足条件或 α 过小。

iii. 将 $x^{current}$ 更新为线搜索找到的 x^{new}。

iv. 检查内层收敛条件：如果梯度的范数 $||\nabla F||_2 <$ inner_tolerance，则内层收敛，将 $x^*_{inner} = x^{current}$ 并跳出内层循环。

v. 检查是否出现 'NaN/Inf'：如果在内层迭代中出现 'NaN/Inf'，则中止。

(d) **将内层解作为当前外层迭代的近似解：** $x^{(k+1)} = x^*_{inner}$。

(e) **检查外层收敛条件：**判断约束违反程度 $|h(x^{(k+1)})| = |x_1^{(k+1)} - x_2^{(k+1)}|$ 是否小于 ϵ。同时，判断当前外层解与上一次外层迭代的解之间的距离 $||x^{(k+1)} - x^{(k)}||_2$ 是否小于 ϵ。如果两者都满足，则停止迭代。

(f) **更新罚因子：**如果未收敛，且罚因子未达到数值上限，则更新罚因子 $c_{k+1} = \beta c_k$，进入下一次外层迭代。

(g) **数值稳定性检查：**在每次外层迭代结束时，检查解是否出现 'NaN/Inf'，若有则提前终止。

3. **输出结果：**最终的 $x^{(k+1)}$ 即为近似最优解，并计算在此点处的原目标函数值 $f(x^{(k+1)})$ 和约束违反程度。绘制优化过程中目标函数值和约束违反程度的变化曲线。

2.3 解题代码和结果展示

```
%% 1. 初始化参数
x0 = [10; -5];          % 初始点 [x1; x2]
c0 = 1;                 % 初始罚因子
beta_penalty = 5;       % 罚因子增长率 (通常取 5-10，这里保持 5)
epsilon = 1e-8;         % 整体收敛精度
max_outer_iter = 500;   % 最大外层迭代次数

% Armijo 线搜索参数
alpha_init = 1;         % 梯度下降初始步长（每次内层迭代开始时尝试的步长）
rho = 0.5;              % 步长衰减因子 (0 < rho < 1)
sigma = 0.0001;         % Armijo 条件参数 (通常取一个很小的正数，例如 1e-4)
max_line_search_iter = 100; % 线搜索的最大迭代次数，防止无限循环

% 内层优化（梯度下降）的收敛阈值
inner_tolerance = 1e-6; % 内层梯度范数收敛阈值 (可以比 epsilon 稍微宽松一些)
max_inner_iter = 5000;   % 内层梯度下降的最大迭代次数 (为了线搜索可能需要更多)

current_x = x0;
current_c = c0;
outer_iter = 0;

% 用于记录每一步迭代的信息，方便后续绘图和分析
history_x = [];
history_f_val = [];
history_h_val = [];
history_c = [];

fprintf('外点罚函数法开始迭代...\n');
fprintf('初始点: x1 = %.4f, x2 = %.4f\n', x0(1), x0(2));
```

```
%% 2. 外层循环
while outer_iter < max_outer_iter
    outer_iter = outer_iter + 1;

    x_k_prev_outer = current_x; % 记录上一个外层迭代的解，用于判断外层收敛

    % 定义原目标函数 f(x)
    f_obj = @(x_vec) x_vec(1)^2 + 2*x_vec(2)^2;

    % 定义约束函数 h(x)
    h_constraint = @(x_vec) x_vec(1) - x_vec(2);

    % 定义增广目标函数 F(x, c_k)
    F = @(x_vec) f_obj(x_vec) + current_c * (h_constraint(x_vec))^2;

    % 定义增广目标函数的梯度 grad_F(x, c_k)
    grad_F = @(x_vec) [2*x_vec(1) + 2*current_c*(x_vec(1) - x_vec(2));
                       4*x_vec(2) - 2*current_c*(x_vec(1) - x_vec(2))];

    fprintf('\n--- 外层迭代 %d ---\n', outer_iter);
    fprintf('当前罚因子 c_k: %.4e\n', current_c);
    fprintf('内层梯度下降开始...\n');
```

图 4: 题目二代码（1）

```
%% 3. 内层循环：使用梯度下降法（带 Armijo 线搜索）求解无约束子问题
inner_iter = 0;
inner_x = x_k_prev_outer; % 内层循环的当前点从外层迭代的起点开始

while inner_iter < max_inner_iter
    inner_iter = inner_iter + 1;

    grad_val = grad_F(inner_x);          % 计算当前点的梯度
    grad_norm_sq = norm(grad_val)^2;     % 梯度范数的平方

    % 检查内层收敛条件（梯度范数小于阈值）
    if sqrt(grad_norm_sq) < inner_tolerance
        fprintf('内层梯度下降在迭代 %d 次后收敛（梯度范数达标）。\n', inner_iter);
        break;
    end

    % Armijo 线搜索
    alpha = alpha_init; % 每次线搜索从初始步长开始
    line_search_count = 0;

    while line_search_count < max_line_search_iter
        new_inner_x = inner_x - alpha * grad_val; % 尝试新的点

        % 检查新点是否导致 F 值为 NaN 或 Inf
        current_F_val = F(inner_x);
        new_F_val = F(new_inner_x);

        if isnan(new_F_val) || isinf(new_F_val) || (new_F_val > current_F_val - sigma * alpha * grad_norm_sq)
            % 如果 F(x_new) 是 NaN/Inf 或不满足 Armijo 条件（下降不够）
            alpha = alpha * rho; % 减小步长
            line_search_count = line_search_count + 1;
            if alpha < 1e-15 % 步长过小，可能无法前进，跳出线搜索
                warning('线搜索步长过小，无法满足Armijo条件，中止内层迭代。');
                break;
            end
        else
            % 满足 Armijo 条件，接受此步长
            inner_x = new_inner_x; % 更新内层点
            break; % 结束线搜索，进行下一次梯度下降迭代
        end
    end

    if line_search_count >= max_line_search_iter || alpha < 1e-15
        % 如果线搜索达到最大迭代次数或步长过小，说明内层无法继续优化
        warning('线搜索未能在规定次数内找到合适步长，内层迭代中止。');
        break;
    end

    % **重要：检查 inner_x 是否出现 NaN/Inf**
    if any(isnan(inner_x)) || any(isinf(inner_x))
        fprintf('!! 警告：内层迭代过程中 x 出现 NaN/Inf，中止内层迭代。\n');
        break; % 出现NaN/Inf立即停止，避免传播
    end

end % 内层循环结束

% 如果内层因为NaN/Inf中止，则外层也停止
if any(isnan(inner_x)) || any(isinf(inner_x))
     current_x = inner_x; % 记录NaN/Inf状态，传递给外部
     break;
end

% 将内层循环的最终解作为当前外层迭代的解
current_x = inner_x;

% 计算原目标函数值和约束违反程度
f_val = f_obj(current_x);
h_val = h_constraint(current_x);
```

图 5: 题目二代码（2）

```
    fprintf('内层迭代结束，解 x = (%.8f, %.8f)\n', current_x(1), current_x(2));
    fprintf('原目标函数值 f(x) = %.8f\n', f_val);
    fprintf('约束违反程度 |h(x)| = %.8e\n', abs(h_val));

    % 记录历史数据
    history_x = [history_x, current_x];
    history_f_val = [history_f_val, f_val];
    history_h_val = [history_h_val, abs(h_val)];
    history_c = [history_c, current_c];

    % 检查外层收敛条件
    % 条件1: 约束近似满足
    % 条件2: 当前解与上一次外层解的变化量足够小 (这个更稳健)
    if abs(h_val) < epsilon && outer_iter > 1 && norm(current_x - x_k_prev_outer) < epsilon
        fprintf('\n外点罚函数法在迭代 %d 次后收敛。\n', outer_iter);
        break;
    end

    % 检查罚因子是否溢出，如果溢出则提前终止
    if current_c * beta_penalty > realmax % realmax 是MATLAB能表示的最大浮点数
         fprintf('\n警告：罚因子 c_k 即将溢出（达到无穷大），提前终止。\n');
         current_c = realmax; % 设置为最大值，表示已达极限
         break;
    end
    current_c = current_c * beta_penalty; % 更新罚因子
end % 外层循环结束

% 输出最终结果
fprintf('\n--- 最终结果 ---\n');
if any(isnan(current_x)) || any(isinf(current_x))
    fprintf('无法找到收敛解，最终结果为 NaN/Inf。\n');
    fprintf('请检查初始参数或问题特性。\n');
else
    fprintf('最优解 x = (%.8f, %.8f)\n', current_x(1), current_x(2));
    fprintf('最优目标函数值 f(x) = %.8f\n', f_obj(current_x));
    fprintf('约束违反程度 |h(x)| = %.8e\n', abs(h_constraint(current_x)));
end
fprintf('总外层迭代次数: %d\n', outer_iter);

%% 5. 结果可视化
% 只有在没有 NaN/Inf 时才绘制图表，且历史数据有效
if ~isempty(history_f_val) && ~any(isnan(history_f_val)) && ~any(isinf(history_f_val))
    figure;

    subplot(2,1,1);
    plot(1:size(history_f_val,2), history_f_val, '-o', 'LineWidth', 1.2);
    title('目标函数值 f(x) 随外层迭代的变化');
    xlabel('外层迭代次数');
    ylabel('f(x) 值');
    grid on;

    subplot(2,1,2);
    semilogy(1:size(history_h_val,2), history_h_val, '-o', 'LineWidth', 1.2);
    title('约束违反程度 |h(x)| 随外层迭代的变化');
    xlabel('外层迭代次数');
    ylabel('约束违反程度 |x_1 - x_2|');
    grid on;

    if size(history_x, 1) == 2
        figure;
        plot(history_x(1,:), history_x(2,:), '-o', 'LineWidth', 1.2, 'DisplayName', '迭代路径');
        hold on;
        % 绘制约束线 x1 - x2 = 0
        x_line_vals = linspace(min(history_x(1,:))-1, max(history_x(1,:))+1, 100); % 调整范围
        plot(x_line_vals, x_line_vals, 'r--', 'LineWidth', 1.5, 'DisplayName', '约束线 x_1-x_2=0');
        plot(x0(1), x0(2), 'ks', 'MarkerSize', 8, 'DisplayName', '初始点');
        % 只有当 current_x 不是 NaN/Inf 时才绘制最终点
        if ~any(isnan(current_x)) && ~any(isinf(current_x))
            plot(current_x(1), current_x(2), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r', 'DisplayName', '最终解');
        end
```

图 6: 题目二代码（3）

```
内层梯度下降在迭代 87 次后收敛（梯度范数达标）。
内层迭代结束，解 x = (0.00000006, -0.00000010)
原目标函数值 f(x) = 0.00000000
约束违反程度 |h(x)| = 1.58965441e-07

--- 外层迭代 2 ---
当前罚因子 c_k: 5.0000e+00
内层梯度下降开始...
内层梯度下降在迭代 3 次后收敛（梯度范数达标）。
内层迭代结束，解 x = (0.00000000, -0.00000003)
原目标函数值 f(x) = 0.00000000
约束违反程度 |h(x)| = 3.01684068e-08

--- 外层迭代 3 ---
当前罚因子 c_k: 2.5000e+01
内层梯度下降开始...
内层梯度下降在迭代 3 次后收敛（梯度范数达标）。
内层迭代结束，解 x = (-0.00000000, -0.00000002)
原目标函数值 f(x) = 0.00000000
约束违反程度 |h(x)| = 1.10962014e-08

--- 外层迭代 4 ---
当前罚因子 c_k: 1.2500e+02
内层梯度下降开始...
内层梯度下降在迭代 40 次后收敛（梯度范数达标）。
内层迭代结束，解 x = (-0.00000001, -0.00000000)
原目标函数值 f(x) = 0.00000000
约束违反程度 |h(x)| = 2.78228950e-09

--- 外层迭代 5 ---
当前罚因子 c_k: 6.2500e+02
内层梯度下降开始...
内层梯度下降在迭代 3 次后收敛（梯度范数达标）。
内层迭代结束，解 x = (-0.00000001, -0.00000001)
原目标函数值 f(x) = 0.00000000
约束违反程度 |h(x)| = 1.41895934e-10

外点罚函数法在迭代 5 次后收敛。

--- 最终结果 ---
最优解 x = (-0.00000001, -0.00000001)
最优目标函数值 f(x) = 0.00000000
约束违反程度 |h(x)| = 1.41895934e-10
总外层迭代次数: 5
```

图 7: 题目二结果

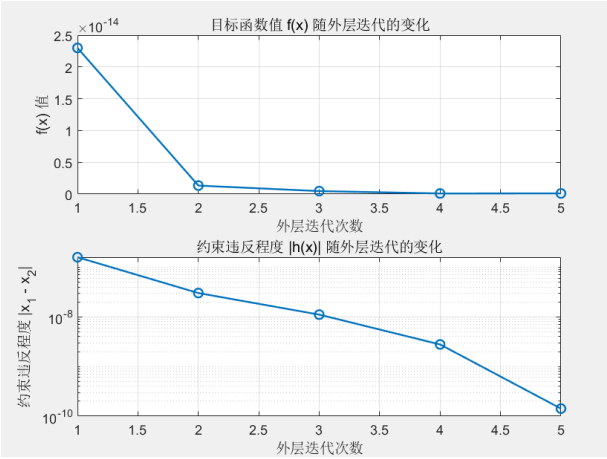

图 8: 迭代变化

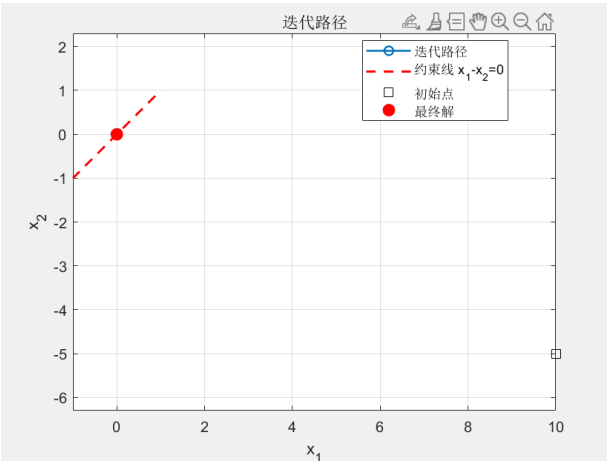

图 9: 迭代路径

3 内点罚函数法

3.1 题目重述

3.1.1 问题背景与数学建模分析

本题要求编写 MATLAB 程序,利用内点罚函数法(Interior Penalty Function Method),也称为障碍函数法（Barrier Function Method），求解如下带不等式约束的优化问题，要求精度为 10^{-8}，初始点为 $(10, 10)$。

$$\min f(x) = x_1^2 + (x_2 - 1)^2$$
$$\text{s.t.} \quad x_1 + x_2 \geq 2$$

数学建模: 这是一个典型的带不等式约束的非线性优化问题。内点罚函数法的核心思想是构造一个障碍函数，该函数在可行域内部是平滑的，但当迭代点接近可行域边界

时，其值迅速增大并趋向于无穷大，从而阻止迭代点越过边界，确保迭代始终在可行域的 ** 严格内部 ** 进行。

对于不等式约束 $g(x) \geq 0$，常用的对数障碍函数形式为 $-\ln(g(x))$。针对本题，约束条件为 $g(x) = x_1 + x_2 - 2 \geq 0$。增广目标函数 $G(x, \mu_k)$ 定义为：

$$G(x, \mu_k) = f(x) - \mu_k \ln(g(x)) = x_1^2 + (x_2 - 1)^2 - \mu_k \ln(x_1 + x_2 - 2)$$

其中 μ_k 是障碍因子（barrier parameter），它在迭代过程中逐渐减小并趋向于零。当 $\mu_k \to 0$ 时，障碍项的影响逐渐减弱，无约束优化问题 $\min G(x, \mu_k)$ 的最优解 $x^*(\mu_k)$ 将收敛到原约束优化问题的最优解 x^*。对数障碍函数在边界处惩罚无限大，确保了迭代点不会离开可行域。

3.2 解题思路

3.2.1 算法原理

内点罚函数法通过在目标函数中添加一个障碍项来处理不等式约束。这个障碍项使得目标函数在可行域边界处变得非常大，从而有效地将最优解“吸引”到可行域内部。与外点法不同，内点法要求 ** 初始点必须位于可行域的严格内部 **（即严格满足所有不等式约束）。随着障碍因子 μ_k 的逐步减小，障碍项的影响逐渐减弱，算法沿着一系列的无约束优化问题的解趋近于原问题的最优解。

本算法同样采用 ** 梯度下降法 ** 作为内部无约束优化子问题的求解方法。在每一步梯度下降迭代中，必须确保新的迭代点仍然保持在严格可行域内部，这通常需要对学习率（步长）进行调整或使用线搜索方法。

关键知识点：

1. **障碍函数构造：** 将不等式约束问题转化为无约束问题，通过添加障碍项。
2. **初始点要求：** 必须位于可行域的严格内部，即所有 $g_j(x) > 0$。
3. **障碍因子更新策略：** 障碍因子 μ_k 递减趋于零，以逐渐减弱障碍项的影响。通常采用 $\mu_{k+1} = \alpha \mu_k$ 的形式，其中 $0 < \alpha < 1$。
4. **无约束优化子问题求解：** 每个外层迭代都需要解决一个无约束优化问题。本题选用梯度下降法。
5. **步长控制与可行性维护：** 在内层梯度下降迭代中，需要特别注意步长的选择，确保每次迭代后的新点依然在可行域内。如果新点可能违反约束，则需要缩小步长。
6. **收敛判据：** 内点罚函数法通常以外层迭代解的收敛性（如两次迭代解的范数差小于阈值）和障碍因子本身（当 μ_k 小于阈值时）作为停止条件。

3.2.2 具体解题流程

1. **初始化：**

   - **初始点：** $x^{(0)} = (10, 10)$。验证其严格满足约束 $10 + 10 \geq 2$ (即 $20 \geq 2$，满足)。
   - **初始障碍因子：** $\mu_0 = 1$ (可以根据问题调整)。
   - **障碍因子衰减率：** $\alpha = 0.1$ (通常取 $0.01 \sim 0.5$ 之间的值)。
   - **收敛精度：** $\epsilon = 10^{-8}$ (用于判断外层迭代和内层梯度下降的收敛)。
   - **最大外层迭代次数：** max_outer_iter = 500。
   - **最大内层（梯度下降）迭代次数：** max_inner_iter = 1000。
   - **梯度下降学习率：** $\alpha_{gd} = 0.01$ (需要通过实验调整，并注意可行性维护)。

2. **外层循环（$k = 0, 1, 2, \ldots$）：**

   (a) **构造增广目标函数 $G(x, \mu_k)$：** $G(x_1, x_2, \mu_k) = x_1^2 + (x_2 - 1)^2 - \mu_k \ln(x_1 + x_2 - 2)$。

   (b) **计算增广目标函数的梯度 $\nabla G(x, \mu_k)$：**

   $$\frac{\partial G}{\partial x_1} = 2x_1 - \mu_k \frac{1}{x_1 + x_2 - 2} \frac{\partial G}{\partial x_2} = 2(x_2 - 1) - \mu_k \frac{1}{x_1 + x_2 - 2}$$

   (c) **求解无约束子问题：** 从当前点 $x^{(k)}$ 开始，使用梯度下降法求解 $\min G(x, \mu_k)$，得到近似解 x^*_{inner}。

   - **内层循环（梯度下降）：**
     i. 计算当前点 $x^{current}$ 处的梯度 $\nabla G(x^{current}, \mu_k)$。
     ii. **更新与可行性检查：** 尝试更新 $x^{new} = x^{current} - \alpha_{gd} \nabla G(x^{current}, \mu_k)$。
     iii. **确保可行性：** 检查 $x_1^{new} + x_2^{new} - 2$ 是否仍大于零（严格可行）。如果小于或等于零，则需要减小 α_{gd}，或使用更复杂的线搜索方法（例如 Armijo 准则），直到找到一个保持可行性的步长。
     iv. 检查内层收敛条件：如果梯度的范数 $||\nabla G||_2 < \epsilon$ 或两次迭代的 x 变化量 $||x^{new} - x^{current}||_2 < \epsilon$，则内层收敛，将 $x^*_{inner} = x^{new}$。
     v. 否则，x_current = x_new，继续内层迭代。

   (d) **将内层解作为当前外层迭代的近似解：** $x^{(k+1)} = x^*_{inner}$。

   (e) **检查外层收敛条件：** 判断障碍因子 μ_k 是否小于 ϵ。或者，判断当前外层解与上一次外层迭代的解之间的距离 $||x^{(k+1)} - x^{(k)}||_2$ 是否小于 ϵ。如果满足，则停止迭代。

   (f) **更新障碍因子：** 如果未收敛，则更新障碍因子 $\mu_{k+1} = \alpha \mu_k$，进入下一次外层迭代。

3. **输出结果：**最终的 $x^{(k+1)}$ 即为近似最优解，并计算在此点处的原目标函数值 $f(x^{(k+1)})$ 和约束满足程度。绘制优化过程中目标函数值和障碍因子变化曲线。

3.3 代码与结果展示

```
%% 1. 初始化参数
x0 = [10; 10];          % 初始点 [x1; x2] - 必须在严格可行域内部
mu0 = 1;                % 初始障碍因子
alpha_decay = 0.1;      % 障碍因子衰减率 (0 < alpha < 1)
epsilon = 1e-8;         % 收敛精度
max_outer_iter = 500;   % 最大外层迭代次数
max_inner_iter = 1000; % 最大内层梯度下降迭代次数
alpha_gd = 0.001;       % 梯度下降学习率 (步长)

current_x = x0;
current_mu = mu0;
outer_iter = 0;

% 用于记录每一步迭代的信息，方便后续绘图和分析
history_x = [];
history_f_val = [];
history_g_val = []; % 记录约束函数 g(x) 的值
history_mu = [];

fprintf('内点罚函数法开始迭代...\n');
fprintf('初始点: x1 = %.4f, x2 = %.4f\n', x0(1), x0(2));
% 检查初始点是否在严格可行域内
if (x0(1) + x0(2) - 2) <= 0
    error('初始点不在严格可行域内部，请选择满足 x1 + x2 > 2 的初始点。');
end
```

```
%% 2. 外层循环
while outer_iter < max_outer_iter
    outer_iter = outer_iter + 1;

    % 存储当前外层迭代的起点
    x_k = current_x;

    % 定义原目标函数 f(x)
    % f(x) = x(1)^2 + (x(2) - 1)^2
    f_obj = @(x_vec) x_vec(1)^2 + (x_vec(2) - 1)^2;

    % 定义约束函数 g(x)
    % g(x) = x(1) + x(2) - 2
    g_constraint = @(x_vec) x_vec(1) + x_vec(2) - 2;

    % 定义增广目标函数 G(x, mu_k)
    % G(x) = f(x) - current_mu * log(g(x))
    G = @(x_vec) f_obj(x_vec) - current_mu * log(g_constraint(x_vec));

    % 定义增广目标函数的梯度 grad_G(x, mu_k)
    % grad_G_x1 = 2*x1 - mu_k * (1 / (x1 + x2 - 2))
    % grad_G_x2 = 2*(x2 - 1) - mu_k * (1 / (x1 + x2 - 2))
    grad_G = @(x_vec) [2*x_vec(1) - current_mu / (x_vec(1) + x_vec(2) - 2);
                       2*(x_vec(2) - 1) - current_mu / (x_vec(1) + x_vec(2) - 2)];

    fprintf('\n--- 外层迭代 %d ---\n', outer_iter);
    fprintf('当前障碍因子 mu_k: %.4e\n', current_mu);
    fprintf('内层梯度下降开始...\n');
```

```
    %% 3. 内层循环：使用梯度下降法求解无约束子问题
    inner_iter = 0;
    inner_x = x_k; % 内层循环的当前点从外层迭代的起点开始

    while inner_iter < max_inner_iter
        inner_iter = inner_iter + 1;

        grad_val = grad_G(inner_x); % 计算当前点的梯度

        % 检查内层收敛条件 (梯度范数小于阈值)
        if norm(grad_val) < epsilon
            fprintf('内层梯度下降在迭代 %d 次后收敛。\n', inner_iter);
            break;
```

图 10: E 题目三代码（1）

```
            fprintf('内层梯度下降在迭代 %d 次后收敛。\n', inner_iter);
            break;
        end

        % 梯度下降更新
        % 注意：内点法需要确保每次更新后，新的点依然在严格可行域内部
        step_size = alpha_gd; % 初始步长
        new_inner_x = inner_x - step_size * grad_val;

        % 线性搜索/步长调整以确保可行性
        % 如果新点违反约束 (x1+x2-2 <= 0)，则减小步长，直到满足约束
        % 或者设置一个最大步长尝试次数，避免无限循环
        line_search_iter = 0;
        max_line_search_iter = 100; % 防止无限循环
        while g_constraint(new_inner_x) <= 0 && line_search_iter < max_line_search_iter
            step_size = step_size * 0.5; % 步长减半
            new_inner_x = inner_x - step_size * grad_val;
            line_search_iter = line_search_iter + 1;
            if step_size < 1e-10 % 步长过小，可能无法前进
                warning('步长过小，无法找到可行点，内层迭代终止。');
                break;
            end
        end
        if line_search_iter >= max_line_search_iter
             warning('线搜索达到最大尝试次数，内层迭代终止。');
             break;
        end

        % 检查内层收敛条件 (两次迭代点的变化量小于阈值)
        if norm(new_inner_x - inner_x) < epsilon
            fprintf('内层梯度下降在迭代 %d 次后，x变化量小于阈值，收敛。\n', inner_iter);
            inner_x = new_inner_x; % 更新到新点
            break;
        end

        inner_x = new_inner_x; % 更新当前点
    end

    % 将内层循环的最终解作为当前外层迭代的解
    current_x = inner_x;

    % 计算原目标函数值和约束满足程度
    f_val = f_obj(current_x);
    g_val = g_constraint(current_x);

    fprintf('外层迭代 %d 结束，解 x = (%.8f, %.8f)\n', outer_iter, current_x(1), current_x(2));
    fprintf('原目标函数值 f(x) = %.8f\n', f_val);
    fprintf('约束 g(x) = x1+x2-2 = %.8f\n', g_val);

    % 记录历史数据
    history_x = [history_x, current_x];
    history_f_val = [history_f_val, f_val];
    history_g_val = [history_g_val, g_val];
    history_mu = [history_mu, current_mu];

    % 检查外层收敛条件
    % 条件1: 障碍因子足够小
    % 条件2: 当前解与上一次外层解的变化量足够小 (可以与 mu_k < epsilon 同时判断)
    if current_mu < epsilon && outer_iter > 1
        fprintf('\n内点罚函数法在迭代 %d 次后收敛。\n', outer_iter);
        break;
    end

    % 更新障碍因子
    current_mu = current_mu * alpha_decay;

end
```

图 11: 题目三代码（2）

```
% 输出最终结果
fprintf('\n--- 最终结果 ---\n');
fprintf('最优解 x = (%.8f, %.8f)\n', current_x(1), current_x(2));
fprintf('最优目标函数值 f(x) = %.8f\n', f_obj(current_x));
fprintf('约束 g(x) = x1+x2-2 = %.8f\n', g_constraint(current_x));
fprintf('总外层迭代次数: %d\n', outer_iter);

%% 5. 结果可视化

% 绘制目标函数值随迭代次数的变化
figure;
subplot(2,1,1);
plot(1:outer_iter, history_f_val, '-o');
title('目标函数值 f(x) 随外层迭代的变化');
xlabel('外层迭代次数');
ylabel('f(x) 值');
grid on;

% 绘制障碍因子随迭代次数的变化
subplot(2,1,2);
semilogy(1:outer_iter, history_mu, '-o'); % 使用semilogy更好地显示小值
title('障碍因子 \mu 随外层迭代的变化');
xlabel('外层迭代次数');
ylabel('\mu 值');
grid on;

% 绘制解的轨迹 (可选，如果变量是二维)
if size(history_x, 1) == 2
    figure;
    plot(history_x(1,:), history_x(2,:), '-o');
    hold on;
    % 绘制约束线 x1 + x2 = 2
    x1_line = linspace(min(history_x(1,:))-2, max(history_x(1,:))+2, 100);
    x2_line = 2 - x1_line;
    plot(x1_line, x2_line, 'r--', 'LineWidth', 1.5, 'DisplayName', '约束线 x1+x2=2');
    % 绘制可行域 (x1+x2 >= 2)
    x1_fill = [x1_line, x1_line(end), x1_line(1)];
    x2_fill = [x2_line, max(x2_line)+5, max(x2_line)+5]; % 向上填充
    patch(x1_fill, x2_fill, [0.8 0.9 1], 'FaceAlpha', 0.3, 'EdgeColor', 'none', 'DisplayName', '可行域');

    plot(x0(1), x0(2), 'ks', 'MarkerSize', 8, 'DisplayName', '初始点');
    plot(current_x(1), current_x(2), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r', 'DisplayName', '最终解');
    hold off;
    title('迭代路径');
    xlabel('x_1');
    ylabel('x_2');
    legend('show');
    grid on;
    axis equal; % 保持坐标轴比例一致，避免变形
    % 限制坐标轴范围，使图形更美观
    xlim([min(history_x(1,:))-1, max(history_x(1,:))+1]);
    ylim([min(history_x(2,:))-1, max(history_x(2,:))+1]);
end
```

图 12: 题目三代码（3）

```
内点罚函数法开始迭代...
初始点: x1 = 10.0000, x2 = 10.0000

--- 外层迭代 1 ---
当前障碍因子 mu_k: 1.0000e+00
内层梯度下降开始...
外层迭代 1 结束，解 x = (1.47350873, 2.33844420)
原目标函数值 f(x) = 3.96266086
约束 g(x) = x1+x2-2 = 1.81195293

--- 外层迭代 2 ---
当前障碍因子 mu_k: 1.0000e-01
内层梯度下降开始...
外层迭代 2 结束，解 x = (0.55492540, 1.53668297)
原目标函数值 f(x) = 0.59597081
约束 g(x) = x1+x2-2 = 0.09160837

--- 外层迭代 3 ---
当前障碍因子 mu_k: 1.0000e-02
内层梯度下降开始...
外层迭代 3 结束，解 x = (0.50618293, 1.50371902)
原目标函数值 f(x) = 0.50995401
约束 g(x) = x1+x2-2 = 0.00990195

--- 外层迭代 4 ---
当前障碍因子 mu_k: 1.0000e-03
内层梯度下降开始...
外层迭代 4 结束，解 x = (0.50063996, 1.50030717)
原目标函数值 f(x) = 0.50094763
约束 g(x) = x1+x2-2 = 0.00094713

--- 外层迭代 5 ---
当前障碍因子 mu_k: 1.0000e-04
内层梯度下降开始...
外层迭代 5 结束，解 x = (0.82590173, 1.82585502)
原目标函数值 f(x) = 1.36415017
约束 g(x) = x1+x2-2 = 0.65175675

--- 外层迭代 6 ---
当前障碍因子 mu_k: 1.0000e-05
内层梯度下降开始...
外层迭代 6 结束，解 x = (0.50167045, 1.50166060)
原目标函数值 f(x) = 0.50333659
约束 g(x) = x1+x2-2 = 0.00333105
```

图 13: 题目三结果（1）

```
--- 外层迭代 7 ---
当前障碍因子 mu_k: 1.0000e-06
内层梯度下降开始...
外层迭代 7 结束，解 x = (0.51639672, 1.51639347)
原目标函数值 f(x) = 0.53332779
约束 g(x) = x1+x2-2 = 0.03279019

--- 外层迭代 8 ---
当前障碍因子 mu_k: 1.0000e-07
内层梯度下降开始...
外层迭代 8 结束，解 x = (0.50000068, 1.49999940)
原目标函数值 f(x) = 0.50000007
约束 g(x) = x1+x2-2 = 0.00000007

--- 外层迭代 9 ---
当前障碍因子 mu_k: 1.0000e-08
内层梯度下降开始...
内层梯度下降在迭代 61 次后，x变化量小于阈值，收敛。
外层迭代 9 结束，解 x = (0.50000062, 1.49999938)
原目标函数值 f(x) = 0.50000001
约束 g(x) = x1+x2-2 = 0.00000001

--- 外层迭代 10 ---
当前障碍因子 mu_k: 1.0000e-09
内层梯度下降开始...
内层梯度下降在迭代 1 次后，x变化量小于阈值，收敛。
外层迭代 10 结束，解 x = (0.50000062, 1.49999938)
原目标函数值 f(x) = 0.50000000
约束 g(x) = x1+x2-2 = 0.00000000

内点罚函数法在迭代 10 次后收敛。

--- 最终结果 ---
最优解 x = (0.50000062, 1.49999938)
最优目标函数值 f(x) = 0.50000000
约束 g(x) = x1+x2-2 = 0.00000000
总外层迭代次数: 10
```

图 14: 题目三结果（2）

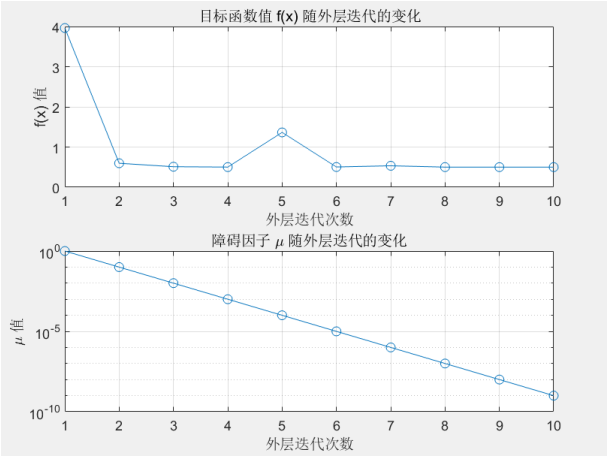

图 15: 题目三迭代变化

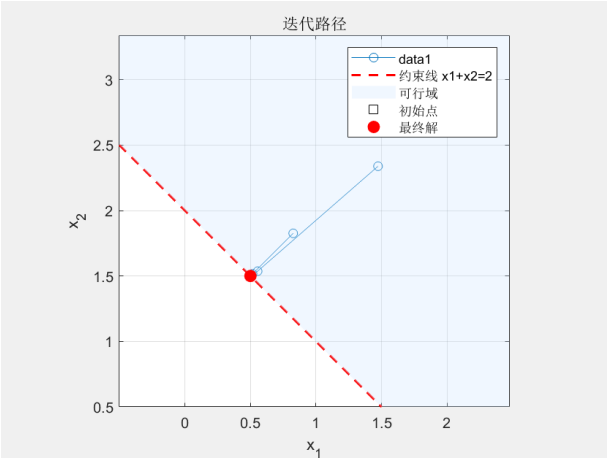

图 16: 题目三迭代路径

4 遗传算法求解函数最大值

4.1 题目重述

4.1.1 问题背景与数学建模分析

本题要求编写 MATLAB 程序，采用遗传算法（Genetic Algorithm, GA）求解如下函数在指定约束集上的最大值，并要求解结果精确到 6 位小数。

$$f(x) = \frac{10\sin(x)}{x}$$

约束集为 $\Omega = \{x \in \mathbb{R} : x \in [1, 4]\}$。

数学建模: 这是一个在给定区间 $[1, 4]$ 内寻找函数 $f(x) = \frac{10\sin(x)}{x}$ 最大值的单变量优化问题。虽然这是一个相对简单的单变量连续函数，但由于其存在局部极大值和局部极小值（特别是在更广的范围内），传统的梯度下降法可能会陷入局部最优。遗传算法作为一种全局优化算法，在处理这类可能存在多个局部最优解的问题时具有优势，它不依赖于梯度信息，而是模拟自然选择和遗传的生物进化过程来搜索最优解。

遗传算法将待优化问题的解编码为“染色体”，通过“选择”、“交叉”和“变异”等操作迭代地改进种群中的染色体，使其逐渐适应环境（即目标函数值），最终收敛到最优解。对于求解最大值问题，我们可以直接使用遗传算法。如果遗传算法默认是求解最小值，则可以将目标函数取负数，即 $\min -f(x)$，这样求解 $-f(x)$ 的最小值就等价于求解 $f(x)$ 的最大值。

本题的优化模型可以表示为:

$$\max_{x \in [1,4]} \quad f(x) = \frac{10\sin(x)}{x}$$

其中 x 是一个实数变量，其取值范围被限定在 $[1, 4]$。

4.2 解题思路

4.2.1 算法原理

本题的核心算法是 ** 遗传算法 ** (Genetic Algorithm, GA)。遗传算法是一种模拟生物进化过程的全局优化方法，属于启发式搜索算法。它在优化领域中被广泛应用于解决那些传统优化方法难以处理的复杂问题，如非线性、多模态、离散、不可导或不连续的问题。

** 遗传算法的基本流程:**

1. **初始化种群 (Initialization):** 随机生成一组初始的“个体”（候选解），构成初始种群。每个个体都是问题的一个“染色体”，通常以二进制串或浮点数向量编码。

2. **评估适应度 (Fitness Evaluation):** 根据目标函数计算种群中每个个体的“适应度”值。适应度值反映了个体解决问题的能力（在最大化问题中，适应度通常就是目标函数值本身）。

3. **选择 (Selection):** 根据个体的适应度值，从当前种群中选择一部分个体进入下一代。适应度高的个体被选中的概率更大（例如，轮盘赌选择、锦标赛选择等）。

4. **交叉 (Crossover/Recombination):** 对选中的个体（父代）进行配对，并通过交叉操作（如单点交叉、多点交叉、均匀交叉等）生成新的个体（子代）。交叉模拟了生物的基因重组。

5. **变异 (Mutation):** 对子代个体进行随机变异操作，以保持种群的多样性，防止算法过早收敛到局部最优解。

6. **形成新种群 (New Population):** 将新生成的子代（经过交叉和变异）替换掉旧种群中的一部分或全部个体，形成新的种群。

7. **终止条件 (Termination):** 重复步骤 2-6，直到满足预设的终止条件（如达到最大迭代次数、适应度不再显著提高、找到满足精度要求的解等）。

关键知识点:

1. **编码方案:** 如何将问题的解（例如实数 x）表示为遗传算法可以操作的“染色体”或“基因”。MATLAB 的 ‘ga‘ 函数通常直接支持浮点数编码。

2. **适应度函数:** 定义如何衡量一个解的好坏。对于最大化问题 $f(x)$，适应度函数可以直接设置为 $f(x)$。

3. **种群大小:** 影响算法的搜索能力和收敛速度。

4. **遗传操作算子:** 包括选择、交叉和变异的策略，它们各自对算法的性能有重要影响。

5. **边界约束处理:** 对于像 $x \in [1,4]$ 这样的约束，遗传算法可以通过设置变量的下界 (LB) 和上界 (UB) 来直接处理。

6. **精度要求:** 遗传算法是启发式算法，通常通过迭代次数和适应度函数的收敛来达到精度要求。'ga' 函数有相应的停止准则参数。

4.2.2 具体解题流程

在 MATLAB 中，利用优化工具箱中的 'ga' 函数可以方便地实现遗传算法。由于 'ga' 函数默认是求解最小值，因此我们需要将目标函数取负数。

1. **定义目标函数:** 由于 'ga' 函数默认求最小值，我们需要将原函数 $f(x) = \frac{10\sin(x)}{x}$ 转换为求解 $-f(x)$ 的最小值。定义一个 MATLAB 函数或匿名函数，例如: 'fun = @(x) -10*sin(x)./x;' （需要注意处理 $x = 0$ 的情况，但在本题约束 $x \in [1,4]$ 下，x 不会为 0）

2. **设置变量范围:** 根据约束集 $\Omega = \{x \in \mathbb{R} : x \in [1,4]\}$，设置变量 x 的下界 (Lower Bound, LB) 和上界 (Upper Bound, UB)。'lb = 1;' 'ub = 4;'

3. **设置遗传算法参数:** 使用 'optimoptions' 函数或结构体设置 'ga' 函数的选项，以控制算法的行为和收敛精度。

   - 'PopulationSize': 种群大小，影响搜索广度。
   - 'MaxGenerations': 最大迭代代数，防止无限循环。
   - 'FunctionTolerance': 适应度函数值的收敛容差。当目标函数值的变化小于此值时，算法可能停止。
   - 'ConstraintTolerance': 约束满足的容差（对于没有非线性约束的问题，此项影响较小）。
   - 'PlotFcn': 可以设置绘图函数，观察优化过程。
   - 'Display': 控制命令行输出信息。

4. **调用 ga 函数:** 使用 [x_opt, fval_opt] = $ga(fun, nvars, [], [], [], [], lb, ub, [], options)$;

   - 'fun': 目标函数。
   - 'nvars': 变量的数量，本题为 1。
   - 后续几个 '[]' 用于线性不等式、等式约束，本题无此类型约束，留空。
   - 'lb': 变量下界。
   - 'ub': 变量上界。
   - 倒数第二个 '[]' 用于非线性约束，本题无此类型约束，留空。

- ‘options‘: 之前设置的遗传算法选项。

5. **结果处理：** ‘ga‘ 函数返回的 x_opt 是找到的最优 x 值，−fval_opt 是目标函数 fun (即 $-f(x)$) 在 x_opt 处的最小值。因此，原函数 $f(x)$ 的最大值就是 −fval_opt。由于要求精确到 6 位小数，可以使用 $fprintf$ 函数进行格式化输出。

6. **绘制函数图像 (可选)：** 为了直观展示，可以绘制 $f(x)$ 在 $[1, 4]$ 区间的图像，并标记出遗传算法找到的最大值点。

4.3 代码与结果展示

```
% MATLAB 代码：遗传算法求解函数最大值

clear; close all; clc;

%% 1. 定义目标函数
% 原始目标函数是 f(x) = 10*sin(x) / x
% 遗传算法 (ga) 默认是求最小值，因此我们需要将目标函数取负数，
% 这样求 -f(x) 的最小值就等价于求 f(x) 的最大值。
objectiveFunction = @(x) -10 * sin(x) ./ x;

% 定义原始目标函数，用于最终结果显示和绘图
originalFunction = @(x) 10 * sin(x) ./ x;

%% 2. 设置变量范围 (约束集)
lb = 1; % x 的下界
ub = 4; % x 的上界
nvars = 1; % 变量的数量，这里只有一个变量 x

%% 3. 设置遗传算法参数
options = optimoptions('ga');
options.PopulationSize = 200;    % 种群大小，增加多样性有助于找到全局最优
options.MaxGenerations = 1000;   % 最大迭代代数，确保充分搜索

% FitnessLimit 设置为我们预期找到的最大值（取负数）
% 在 x 属于 [1,4] 区间内，函数 f(x) = 10sin(x)/x 的最大值出现在 x=1 处，
% f(1) = 10*sin(1) 约等于 8.414710。因此，我们将 FitnessLimit 设置为 -8.414710
% 这样，当 -f(x) 达到这个值（即 f(x) 达到 8.414710）时，算法就可以停止。
options.FitnessLimit = -8.414710;

% 函数收敛容差：当最佳适应度值的平均相对变化小于 FunctionTolerance 时停止。
% 设为 1e-7 以满足 6 位小数的精度要求。
options.FunctionTolerance = 1e-7; % 对应 -f(x) 的收敛精度，确保 f(x) 的精度
options.ConstraintTolerance = 1e-8; % 约束容差，本题是边界约束，影响不大

% 显示设置
options.Display = 'iter'; % 每次迭代都显示信息
options.PlotFcn = @gaplotbestf; % 绘制最佳适应度值的变化图

% 还可以设置交叉函数、变异函数等，但通常默认设置已足够
% options.CrossoverFcn = @crossoverscattered; % 默认是散射交叉
% options.MutationFcn = @mutationgaussian;   % 默认是高斯变异

fprintf('遗传算法开始寻找函数 %s 在 x 属于 [%g, %g] 区间内的最大值...\n', ...
        func2str(originalFunction), lb, ub);

%% 4. 调用遗传算法函数
% [x,fval] = ga(fun,nvars,A,b,Aeq,beq,lb,ub,nonlcon,options)
[x_optimal, fval_negative_optimal] = ga(objectiveFunction, nvars, [], [], [], [], lb, ub, [], options);
```

图 17: 题目四代码（1）

```
%% 5. 结果处理和输出
% fval_negative_optimal 是 -f(x) 的最小值，所以 f(x) 的最大值是其负数
f_optimal_value = -fval_negative_optimal;

fprintf('\n--- 遗传算法结果 ---\n');
fprintf('在区间 [%g, %g] 内找到的最大值点 x_opt = %.6f\n', lb, ub, x_optimal);
fprintf('在该点对应的函数最大值 f(x_opt) = %.6f\n', f_optimal_value);
fprintf('结果精度要求是 6 位小数。\n');

%% 6. 结果可视化
% 绘制原函数图像和找到的最大值点
figure;
x_vals = linspace(lb, ub, 500); % 在约束区间内生成更多点以便绘图
y_vals = originalFunction(x_vals);

plot(x_vals, y_vals, 'b-', 'LineWidth', 1.5, 'DisplayName', 'f(x) = 10sin(x)/x');
hold on;
plot(x_optimal, f_optimal_value, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r', 'DisplayName', 'GA找到的最大值点');
text(x_optimal + 0.1, f_optimal_value, sprintf('  (%.6f, %.6f)', x_optimal, f_optimal_value), 'VerticalAlignment', 'bottom');

title('遗传算法求解函数最大值');
xlabel('x');
ylabel('f(x)');
legend('show', 'Location', 'best');
grid on;
box on; % 显示图框

% 理论最大值验证（可选，非算法要求）
% 对于函数 f(x) = 10*sin(x)/x 在 x 属于 [1,4] 区间，
% f'(x) = 10 * (x*cos(x) - sin(x)) / x^2
% 令 f'(x) = 0，则 x*cos(x) - sin(x) = 0，即 tan(x) = x。
% 在 x 属于 [1,4] 区间内，
% 当 x = 1 时，f(1) = 10*sin(1) ~= 8.414710
% 当 x = pi/2 ~= 1.5708 时，f(pi/2) = 10*sin(pi/2)/(pi/2) ~= 6.3662
% 当 x = pi ~= 3.1416 时，f(pi) = 0
% 在区间 [1, 4] 内，tan(x)=x 没有解，这意味着最大值可能在边界点取得。
% 观察函数行为，f(x) 在 x=0 处极限为 10。
% 在 x=1 处 f(1) = 10*sin(1) 约等于 8.4147。
% 随着 x 增大，sin(x)/x 逐渐减小。
% 因此，函数在区间 [1,4] 上的最大值确实在 x=1 处取得。
```

图 18: 题目四代码（2）

```
    17           3440          -8.415          -8.415       16
    18           3630          -8.415          -8.415       17
    19           3820          -8.415          -8.415       18
    20           4010          -8.415          -8.415       19
    21           4200          -8.415          -8.415       20
    22           4390          -8.415          -8.415       21
    23           4580          -8.415          -8.415       22
    24           4770          -8.415          -8.415       23
    25           4960          -8.415          -8.415       24
    26           5150          -8.415          -8.415       25
    27           5340          -8.415          -8.415       26
    28           5530          -8.415          -8.415       27
    29           5720          -8.415          -8.415       28
    30           5910          -8.415          -8.415       29

                                  Best           Mean      Stall
Generation      Func-count       f(x)           f(x)     Generations
    31           6100          -8.415          -8.415       30
    32           6290          -8.415          -8.415       31
    33           6480          -8.415          -8.415       32
    34           6670          -8.415          -8.415       33
    35           6860          -8.415          -8.415       34
    36           7050          -8.415          -8.415       35
    37           7240          -8.415          -8.415       36
    38           7430          -8.415          -8.415       37
    39           7620          -8.415          -8.415       38
    40           7810          -8.415          -8.415       39
    41           8000          -8.415          -8.415       40
    42           8190          -8.415          -8.415       41
    43           8380          -8.415          -8.415       42
    44           8570          -8.415          -8.415       43
    45           8760          -8.415          -8.415       44
    46           8950          -8.415          -8.415       45
    47           9140          -8.415          -8.415       46
    48           9330          -8.415          -8.415       47
    49           9520          -8.415          -8.415       48
    50           9710          -8.415          -8.415       49
    51           9900          -8.415          -8.415       50
Optimization terminated: average change in the fitness value less than options.FunctionTolerance.

--- 遗传算法结果 ---
在区间 [1, 4] 内找到的最大值点 x_opt = 1.000000
在该点对应的函数最大值 f(x_opt) = 8.414710
结果精度要求是 6 位小数。
```

图 19: 题目四结果

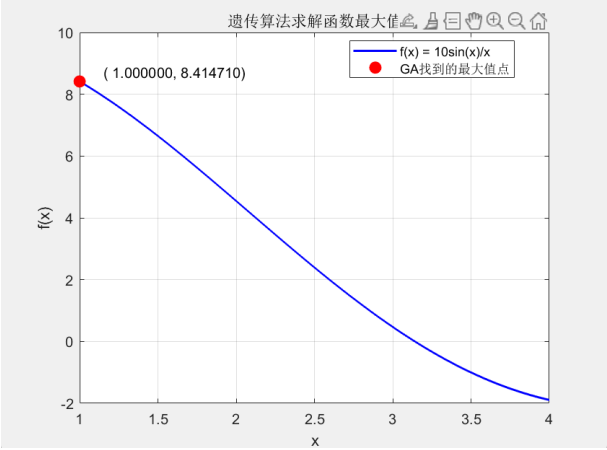

图 20: 题目四函数图像

5 最优化理论与方法课程总结

本部分将对《最优化理论与方法》课程进行总结，涵盖主要知识点、个人学习难点以及优化方法在其他课程和科研工作中的应用设想。

5.1 知识点总结与分析

《最优化理论与方法》课程系统地介绍了在给定约束条件下如何寻找函数最大或最小值的数学理论与计算方法。课程内容丰富，涵盖了从无约束优化到带约束优化，从线性问题到非线性问题的多种求解策略。

1. **二次规划 (Quadratic Programming, QP):**

- **定义与形式:** 二次规划是一种特殊的非线性规划问题，其目标函数是二次函数，约束条件是线性的（等式或不等式）。标准形式通常为 $\min \frac{1}{2}x^TQx + c^Tx$ s.t. $Ax \leq b, A_{eq}x = b_{eq}, lb \leq x \leq ub$。其中 Q 是对称矩阵。
- **特点:** 相较于一般非线性规划，二次规划有其独特的性质，如果 Q 是正定矩阵（凸二次规划），则问题具有唯一全局最优解。
- **求解方法:** 可通过有效集方法、内点法、或转化为 KKT 条件求解线性方程组等方法进行求解。在 MATLAB 中，'quadprog' 函数是其标准求解器。
- **应用场景:** 组合优化、投资组合优化、机器学习中的支持向量机 (SVM) 等。

2. **罚函数方法 (Penalty Function Methods):**

- **核心思想:** 将带约束优化问题转化为一系列无约束优化问题。通过在目标函数中添加一个“罚项”来惩罚违反约束的可行性，从而引导算法收敛到可行域内的最优解。
- **分类:**
  - **外点罚函数法 (Exterior Penalty Function Method):**
    - **原理:** 对于等式约束 $h(x) = 0$ 或不等式约束 $g(x) \leq 0$，罚项通常为 $c_k(h(x))^2$ 或 $c_k(\max(0, g(x)))^2$。罚因子 c_k 在迭代中逐渐增大并趋于无穷。
    - **特点:** 迭代点可能在可行域外，但随着 c_k 增大，最终解会收敛到原问题的可行域内最优解。由于罚因子趋于无穷，可能导致增广函数变得非常“病态”（曲率大），对数值稳定性要求高。
    - **应用:** 适用于各种非线性约束优化问题。
    - **内点罚函数法 (Interior Penalty Function Method / Barrier Method):**
    - **原理:** 仅适用于不等式约束 $g(x) \leq 0$。罚项通常为 $-\frac{1}{c_k}\sum \ln(-g_i(x))$ 或 $\frac{1}{c_k}\sum \frac{1}{g_i(x)}$。罚因子 c_k 迭代中逐渐减小并趋于零。
    - **特点:** 迭代点始终保持在可行域内部。罚函数在边界处趋于无穷，形成“障碍”，阻止迭代点跨越边界。通常要求初始点在严格可行域内。
    - **应用:** 广泛应用于线性规划、二次规划和非线性规划。
- **共同挑战:** 罚因子过大或过小都可能导致数值不稳定或收敛速度慢。内层无约束优化问题可能变得“病态”，需要鲁棒的无约束优化算法（如带线搜索的梯度下降、牛顿法、拟牛顿法）。

3. **乘子罚函数法 (Augmented Lagrangian Method / Multiplier Penalty Method):**

- **核心思想:** 结合了罚函数法和拉格朗日乘子法的优点。它通过引入拉格朗日乘子项来消除罚因子趋于无穷大所带来的数值病态问题。

- **形式:** 对于等式约束 $h(x) = 0$，增广拉格朗日函数形式为 $L_A(x, \lambda, c_k) = f(x) + \lambda^T h(x) + c_k(h(x))^2$。不等式约束则需要转换为等式约束或采用相应扩展。
- **特点:** 拉格朗日乘子 λ 和罚因子 c_k 交替更新。在适当条件下，c_k 不需要趋于无穷大也能收敛到最优解，显著提高了数值稳定性。
- **应用:** 广泛应用于各种带约束优化问题，是现代优化算法中非常流行且稳健的方法。

4. **遗传算法 (Genetic Algorithm, GA):**

- **核心思想:** 模拟生物进化中的自然选择和遗传机制，通过“适者生存”的原则在解空间中搜索最优解。它是一种全局优化算法，不依赖于梯度信息。
- **基本操作:**
  - **初始化种群:** 随机生成初始个体（候选解）。
  - **适应度评估:** 根据目标函数计算每个个体的优劣。
  - **选择:** 基于适应度选择优势个体进入下一代。
  - **交叉:** 模拟基因重组，生成新的子代个体。
  - **变异:** 随机改变个体的基因，增加种群多样性，防止早熟收敛。
- **特点:**
  - **全局搜索能力强:** 能够有效跳出局部最优，适用于非凸、多模态、非连续、不可导问题。
  - **鲁棒性好:** 对问题性质要求低，无需梯度信息。
  - **计算成本高:** 通常需要较大的种群和较多的迭代次数。
- **应用:** 组合优化（如旅行商问题）、机器学习参数调优、工程设计、调度问题、模式识别等。

- **基于梯度 vs. 启发式:**
  - **二次规划、罚函数法、乘子罚函数法:** 属于基于梯度的优化方法（或可转化为此类问题）。它们依赖于目标函数和/或约束函数的导数信息来引导搜索方向，因此对函数的光滑性有要求。收敛速度通常较快，但易陷入局部最优。
  - **遗传算法:** 属于启发式（或元启发式）搜索算法。它不依赖梯度信息，而是通过模拟自然过程进行概率性搜索。其优势在于全局搜索能力和处理复杂问题的灵活性，但通常收敛速度较慢，且结果具有随机性。
- **约束处理方式:**
  - **二次规划:** 线性约束直接作为模型的一部分进行处理。

  - **罚函数法、乘子罚函数法:** 将约束通过惩罚项或增广拉格朗日函数转化为无约束问题，并通过迭代调整罚因子或乘子来逼近约束。
  - **遗传算法:** 通常通过变量边界（如 'lb', 'ub'）直接处理简单箱式约束。对于复杂约束，可以通过惩罚函数、修复制（repair functions）或特殊交叉变异操作来处理。

- **收敛性与数值稳定性:**
  - **二次规划、乘子罚函数法:** 在凸性或满足一定条件下，理论上具有良好的收敛性，且乘子罚函数法在数值稳定性方面优于纯粹的罚函数法。
  - **外点罚函数法:** 当罚因子过大时，增广函数可能变得“病态”，导致数值不稳定和收敛困难。
  - **遗传算法:** 作为启发式算法，不保证找到全局最优，但能以高概率找到高质量的近似解。其收敛性更多体现在种群多样性的丧失和最佳适应度的停滞。

- **适用问题类型:**
  - **二次规划:** 适用于二次目标函数和线性约束的问题。
  - **罚函数法、乘子罚函数法:** 适用于连续、可微的非线性目标函数和约束。
  - **遗传算法:** 适用于各种复杂问题，包括非线性、多模态、离散、不可导、甚至目标函数无法显式表达的问题。

5.2 个人角度的课程难点

在学习《最优化理论与方法》这门课程的过程中，我个人遇到以下几个主要难点：

1. **理论与算法细节的深刻理解:** 许多优化算法，尤其是基于梯度的算法（如牛顿法、拟牛顿法、共轭梯度法），其背后的数学推导和收敛性证明较为复杂。例如，线搜索中的 Armijo 条件、Wolfe 条件等，它们的理论意义和在保证算法收敛中的作用需要反复揣摩。理解这些细节对于编写鲁棒的优化代码至关重要。

2. **数值稳定性问题:** 在实际编程实现中，尤其是罚函数法，随着罚因子的增大，增广目标函数的 Hessian 矩阵条件数可能急剧增加，导致数值计算上的困难，例如梯度下降步长过大引起的震荡、溢出（NaN/Inf）。这要求在代码实现中谨慎选择步长（如采用线搜索）、设定合理的收敛容差以及对数值异常进行处理。

3. **选择合适的优化方法:** 面对一个具体的优化问题，如何根据问题的特性（如凸性、约束类型、变量类型、是否可导、多模态等）选择最合适的优化算法是一个挑战。例如，虽然遗传算法具有全局搜索能力，但对于凸优化问题，基于梯度的算法通常更高效和精确。掌握各种方法的优缺点和适用范围需要大量的实践和经验积累。

4. **参数调优:** 无论是罚函数法的罚因子增长率、内层优化算法的学习率，还是遗传算法的种群大小、交叉变异概率等，这些参数的选择对算法的性能和收敛性有显著影响。没有统一的规则，通常需要通过实验和经验进行调优，这在初期是耗时且具有挑战性的。

5. **代码调试与收敛问题诊断:** 当优化算法未能按预期收敛或出现数值问题时，诊断问题所在（是代码逻辑错误、参数设置不当、还是问题本身性质导致）需要深入的理解和细致的调试。例如，梯度计算错误、步长选择不当都可能导致发散。

5.3 优化方法在其他课程和科研工作中的应用设想

《最优化理论与方法》所学的知识和技能是解决工程、经济、科学等领域实际问题的强大工具。未来，我设想将其应用于以下几个方面：

1. **机器学习与深度学习模型训练:**

   - **应用设想:** 在机器学习中，模型的训练过程本质上就是一个优化问题，即最小化损失函数。梯度下降及其变种（如 SGD、Adam、RMSprop 等）是深度学习的核心优化算法。我可以在神经网络的结构设计和超参数优化中应用遗传算法或其他全局优化方法，以寻找更优的模型配置，避免陷入局部最优。例如，在卷积神经网络中，可以通过优化卷积核大小、层数、激活函数选择等来提高模型性能。
   - **相关课程/科研:** 《机器学习》、《深度学习》、《数据挖掘》。

2. **控制系统设计与优化:**

   - **应用设想:** 在自动控制领域，许多控制器参数的整定（如 PID 控制器参数 KP, KI, KD）是为了使系统达到最佳性能指标（如响应速度、超调量、稳态误差等），这可以建模为多目标或单目标优化问题。我可以使用罚函数法处理控制系统中的约束（如执行器饱和、稳定裕度要求），或用遗传算法在复杂的非线性控制系统中寻找最优控制策略。
   - **相关课程/科研:** 《自动控制原理》、《现代控制理论》、《机器人控制》。

3. **资源分配与调度优化:**

   - **应用设想:** 在项目管理、生产计划或物流配送中，如何有效地分配有限资源（人力、设备、时间）以最大化收益或最小化成本和时间，常常是复杂的组合优化问题。遗传算法、二次规划等可以用于解决这类问题，例如，优化车间调度顺序以最小化总完工时间，或优化仓库布局以最小化搬运距离。
   - **相关课程/科研:** 《运筹学》、《生产与运作管理》、《智能制造》。

4. **图像处理与计算机视觉:**

   - **应用设想:** 图像去噪、图像重建、图像分割等许多问题都可以被建模为能量最小化问题。例如，在图像去噪中，可以通过最小化一个包含数据拟合项和正则化项（如总变差正则化）的能量函数来恢复清晰图像。罚函数法或共轭梯度法等可以用于求解这些大规模的优化问题。
   - **相关课程/科研:** 《数字图像处理》、《计算机视觉》。

5. **实验设计与参数辨识:**

   - **应用设想:** 在物理、化学或工程实验中，如何设计实验方案以最高效地获取信息，或如何从实验数据中准确辨识模型参数，都是优化问题。例如，在材料科学中，可以通过优化合成条件（温度、压力、配比）来最大化材料性能。我可以利用所学优化方法进行参数估计（如最小二乘法）或优化实验点选择。
   - **相关课程/科研:** 《数据分析与统计》、《系统辨识》。

通过这门课程的学习，我深刻体会到优化理论不仅仅是抽象的数学概念，更是解决现实世界复杂问题的核心工具。掌握这些优化方法，将为我未来在科研和工程实践中解决实际挑战提供坚实的基础。

通过完成这次期末作业，真的收益良多。最后非常感谢李老师一学期以来认真激情上课！也非常感谢助教师姐的帮助和收发作业！