《系统仿真与 matlab》综合试题

题 目：森林救火问题

编 号：13

姓 名 秦明远

班 级 人工智能 2202 班

学 号 U202214966

联系方式 15623089585

得分项	创新性	工作量	代码可读性	报告写作	总分
分数					

《系统仿真与 matlab》综合试题

要求： 1 所有程序均要求用 matlab 完成；

2 程序要有相应的注释，条理性好；
3 调试通过，没有错误；
4 程序每个人独立完成；
5 要有研究报告，研究报告要求有目录，封面用提供的统一封面；
6 题目从所给的题目列表中挑选，也可以自己选择，如果自己选择，应对所给题目进行详细描述，题目形式可以多样，但应体现系统仿真的基本概念及仿真策略和方法，要求用 matlab 实现该系统；原则上是从所给题目列表中选取；
7 特别提醒：程序每个人自己独立完成，如果发现有两个人的程序或研究报告完全相同，将视同考试作弊，与考场上的考试作弊同等对待；一旦发现有从网上下载的程序，也将视同考试作弊，与考场上的考试作弊同等对待。

评分标准： 1 创新性；

2 程序工作量；
3 程序可读性；
4 报告质量；
5 平时作业；
6 实验成绩。

提交材料： 1 研究报告纸质版（包括：封面、目录、试题建模过程、试题实现中的关键难点、程序运行指南、程序运行实例分析）；
2 研究报告单独成一个 word 文档，命名为：姓名＋班级＋学号；
3 实验报告纸质版；
4 实验报告单独成一个 word 文档，命名为：姓名＋学号，一个班放在一个文件夹内；
5 编写的源程序电子版，2024 年 12 月 31 日前发送到 wxpwinnie@163.com；
6 全体同学提交的纸质材料，由一位（或多位）同学在 2024 年 12 月 31 日之前（2024 年 12 月 31 日是最后一天）交到南一楼中 527 室。

1 引言

森林失火了，消防站接到报警后派多少消防队员前去救火呢?派的队员越多，森林的损失越小，但是救援的开支会越大，所以需要综合考虑森林损失费和救援费与消防队员人数之间的关系，以总费用最小来决定派出队员的数目。

2 模型建立

2.1 问题分析

损失费通常正比于森林烧毁的面积，而烧毁面积与失火、灭火(指火被扑灭)的时间有关，灭火时间又取决于消防队员数目，队员越多灭火越快．救援费除与消防队员人数有关外，也与灭火时间长短有关。记失火时刻为$t=0$，开始救火时刻为$t=t_1$，灭火时刻为$t=t_2$。设在时刻t森林烧毁面积为$B(t)$，则造成损失的森林烧毁面积为$B(t_2)$．建模要对函数$B(t)$的形式作出合理的简单假设．

研究$\frac{dB}{dt}$比$B(t)$更为直接和方便。$\frac{dB}{dt}$是单位时间烧毁面积，表示火势蔓延的程度．在消防队员到达之前，即$0\le t\le t_1$火势越来越大，即$\frac{dB}{dt}$随t的增加而增加；开始救火以后，即$t_1\le t\le t_2$．如果消防队员救火能力足够强，火势会越来越小，即$\frac{dB}{dt}$应减小，并且当$t=t_2$时$\frac{dB}{dt}=0$．

救援费可分为两部分；一部分是灭火器材的消耗及消防队员的薪金等，与队员人数及灭火所用的时间均有关，另一部分是运送队员和器材等一次性支出，只与队员人数有关．

2.2 模型假设

需要对烧毁森林的损失费、救援费及火势蔓延程度$\frac{dB}{dt}$的形式作出假设。

1．损失费与森林烧毁面积$B(t_2)$成正比，比例系数c_1，c_1即烧毁单位面积的损失费．

2．从失火到开始救火这段时间($0\le t\le t_1$)内，火势蔓延程度$\frac{dB}{dt}$与时间t成正比，比例系数β称火势蔓延速度。

3．派出消防队员 x 名，开始救火以后($t\ge t_1$)火势蔓延速度降为$\beta-\lambda x$，其中λ可视为每个队员的平均灭火速度．显然应有$\beta<\lambda x$ [1]

4．每个消防队员单位时间的费用为c_2，于是每个队员的救火费用是$c_2(t_2-t_1)$；每个队员的一次性支出是c_3.

2.3 符号说明

符号	符号说明
t 时刻森林烧毁面积	$B(t)$
火势蔓延速度	β
火势蔓延程度	$\frac{dB}{dt}$
烧毁单位面积的损失费	c_1
开始灭火时刻	t_1
消防员人数	x
每个队员的平均灭火速度	λ
每个消防队员单位时间的费用为	c_2
每个队员的一次性支出是	c_3
灭火时刻	t_2
总费用	c

根据假设，容易得出:

$$B(t)=\begin{cases}\int_0^t \beta t\,dt\ (0<t\le t_1)\\ \frac{\beta t_1^2}{2}+\int_0^{t-t1}(\beta t_1+(\beta-\lambda x)t)dt\ (t1<t\le t_2)\end{cases}$$

$$\int_0^{t_1}\beta\ dt+\int_0^{t_2-t_1}(\beta-\lambda x)\ dt=0$$

$$c=c_1B(t_2)+c_2x(t_2-t_1)+c_3$$

2.4 模型建立

2.4.1 一般情况下的仿真

根据当$t=t_2$时$\frac{dB}{dt}=0$，即火焰蔓延速度为 0，可以得出

$$\int_0^{t_1}\beta\ dt+\int_0^{t_2-t_1}(\beta-\lambda x)\ dt=0$$

求解，可得:

$$t_2 = \frac{\lambda x}{\lambda x - \beta}$$

再根据:

$$B(t_2) = \frac{\beta t_1^2}{2} + \beta t_1(t_2 - t_1) + \frac{(\beta - \lambda x)(t_2 - t_1)^2}{2}$$

求出烧毁的总面积。

最后，根据:

$$c = c_1 B(t_2) + c_2 x(t_2 - t_1) + c_3$$

计算出本次森林救火的总成本。

2.4.2 最优派遣求解

目标函数:

$$\min \mathrm{c} = c_1 B(t_2) + c_2 x(t_2 - t_1) + c_3$$

约束:

$$x \geq \frac{\beta}{\lambda}$$

是一个**非线性的整数优化**问题，因此采用遗传算法可快速求解:

- 最大迭代次数: 最多运行 100 代
- 停滞条件: 如果连续 50 代没有改进，算法将停止
- 种群大小: 每代的种群有 50 个个体
- 交叉概率: 0.8
- 精英保留: 每代保留 2 个最优秀的个体直接进入下一代

3 系统仿真过程

3.1 仿真流程

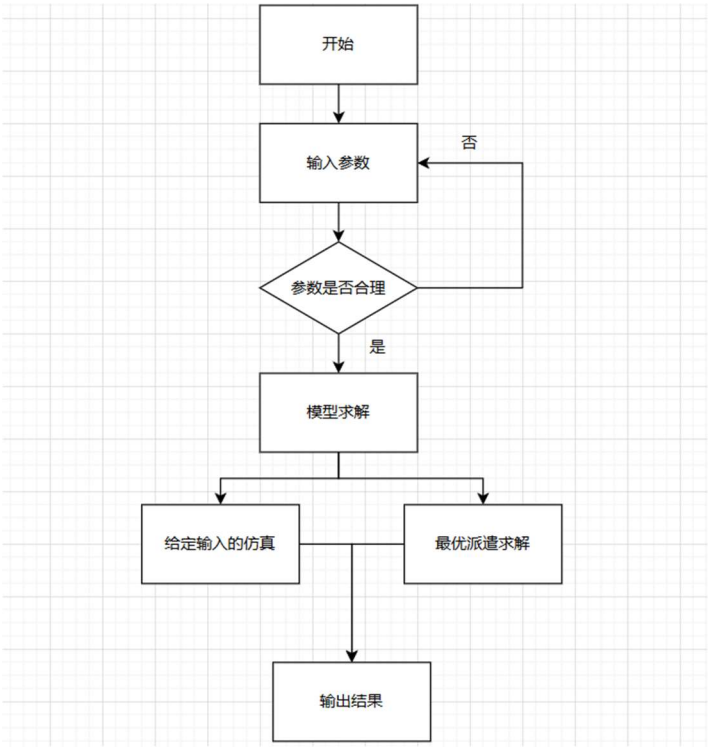
仿真过程的流程主要有:

(1) 参数初始化步骤，用户输入仿真所需要的参数。包括: 消防员人数x、火势蔓延速度β、每个消防员的灭火速度λ、开始灭火时间t_1、单位面积损失费用c_1、每队员单位时间费用c_2、每队员一次性费用c_3。对参数的合理性进行检验后，即判定: $x > \frac{\beta}{\lambda}$。初

始化对应的参数。

（2） 根据建立的模型进行求解。

（3） 结果输出。包括：灭火前的火灾蔓延面积$B(t_1)$、最终烧毁面积$B(t_2)$、损失费用c_1、救援费用c_2+c_3、总费用c、灭火完成时刻t_2。



3.2 仿真关键

3.2.1 仿真界面的设计

本次仿真通过 appdesigner 工具进行界面的设计，同时设计回调函数，实现仿真参数的输入、仿真结果的可视化输出。

3.2.2 数学模型的仿真和求解

solve 函数求解**灭火时刻**

$$\int_0^{t_1}\beta\ dt+\int_0^{t_2-t_1}(\beta-\lambda x)\ dt=0$$

```
function t2_solution = solve_t2(beta, lambda, x, t1)
    % 解方程：beta * t1 + (beta - lambda * x) * (t2 - t1) = 0
    syms t2;
    equation = beta * t1 + (beta - lambda * x) * (t2 - t1) == 0;
    t2_solution = double(solve(equation, t2));  % 转化为数值解
```

```
end
```

分段函数求解**火焰蔓延程度**

```
% 计算火灾燃烧程度
function burning_degree=calculate_burning_degree(beta,t1,lambda,x,time)
    if time<t1
        burning_degree = beta * time;   % 火灾蔓延程度
    else
        t_ext = time-t1;
        burning_degree = beta * t1 + (beta - lambda * x) * t_ext;
    end
end
```

分段函数求解**火焰燃烧面积**

$$B(t) = \begin{cases} \int_0^t \beta t \, dt \ (0 < t \le t_1) \\ \frac{\beta t_1^2}{2} + \int_0^{t-t1} (\beta t_1 + (\beta - \lambda x)t) dt \ (t1 < t \le t_2) \end{cases}$$

```
% 计算火灾燃烧面积
function burning_area = calculate_burning_area(beta,t1,lambda,x,time)
    if time<t1
        burning_area = 0.5 * beta * time^2;   % 火灾蔓延产生的燃烧面积
    else
        t_ext = time-t1;
        burning_area = 0.5 * beta * t1^2 + beta * t1 * t_ext + 0.5 * (beta - lambda * x) * (t_ext^2);

    end
end
```

遗传算法求解**最优派遣人数**

```
function [x_optimal, min_cost, t2] = solve_optimize_constrained(beta, lambda, t1, c1, c2, c3)
    % 目标函数
    objective_func = @(x) objective(x, t1, beta, lambda, c1, c2, c3);

    % 设置遗传算法选项
    options = optimoptions('ga', 'Display', 'iter', 'MaxGenerations', 100, 'MaxStallGenerations', 50,
'PopulationSize', 50, 'CrossoverFraction', 0.8, 'EliteCount', 2);

    % 设置遗传算法约束：x >= ceil(beta / lambda) + 1，并且 x 必须为整数
    lb = ceil(beta / lambda) ; % 最小值
    ub = 10000; % 假设上限为 100，你可以根据实际情况调整
```

```
    % 求解：使用遗传算法
    [x_optimal, ~] = ga(objective_func, 1, [], [], [], [], lb, ub, [], options); % 1 代表优化的是一个变量 x

    % x 四舍五入
    x_optimal = round(x_optimal);
    min_cost = objective(x_optimal, t1, beta, lambda, c1, c2, c3);

    % 计算 t2
    t2 = (lambda * x_optimal / (lambda * x_optimal - beta)) * t1;

    % 输出最优结果
    disp(['Optimal x: ', num2str(x_optimal)]);
    disp(['Minimum cost: ', num2str(min_cost)]);
end
```

3.2.3 仿真的输入

用户输入仿真所需要的参数。包括：消防员人数x、火势蔓延速度β、每个消防员的灭火速度λ、开始灭火时间t_1、单位面积损失费用c_1、每队员单位时间费用c_2、每队员一次性费用c_3。在 APP 界面中，每个参数对应一个可编辑文本框（uieditfield）和一个标签（uilabel）用于提示用户输入参数的含义。同时，代码支持通过键盘输入各个参数并将值赋给相应的变量。

3.2.4 仿真模式的选择

为实现手动输入消防员人数和自动计算最优派遣人数的功能，我进行了扩展，主要包括以下内容：

1. 增加一个选项按钮组（ButtonGroup），用于选择两种模式：

   手动模式：用户输入所有参数，包括消防员人数 x，系统直接计算并输出结果。

   自动模式：用户输入除 x 外的其他参数，系统根据优化算法自动计算最优的 x 并输出结果。

2. 自动计算最优人数：添加一个算法，通过迭代或最优化方法求解，使总成本最小化，并输出最优消防员人数和相关结果。

3.2.5 仿真的动态过程

森林的燃烧面积和火焰蔓延程度均可通过数学公式计算出来并以函数图的形式动态绘制，随着时间的步进，图像逐渐被绘制展示出来。同时用户界面中有显示森林和火焰大小的图像，随着火焰的蔓延和灭火，图片也随之动态改变，展示真实的仿真效果。

3.2.6 仿真的输出

仿真的输出包括：灭火前的火灾蔓延面积$B(t_1)$、最终烧毁面积$B(t_2)$、损失费用c_1、救援费用c_2+c_3、总费用c、灭火完成时刻t_2。以及绘制的$B-t$曲线和$\frac{dB}{dt}-t$曲线。在最优派遣求解下，会输出不同派遣人员下的总成本曲线以及最优点。

3.3 仿真难点

本仿真的核心算法难点在于利用遗传算法**求解非线性整数规划问题**,具体体现在消防员人数 x 的最优解计算。由于目标函数涉及火焰蔓延程度与灭火效果之间的非线性关系，以及成本计算中的约束条件。

代码的实现难点在于:

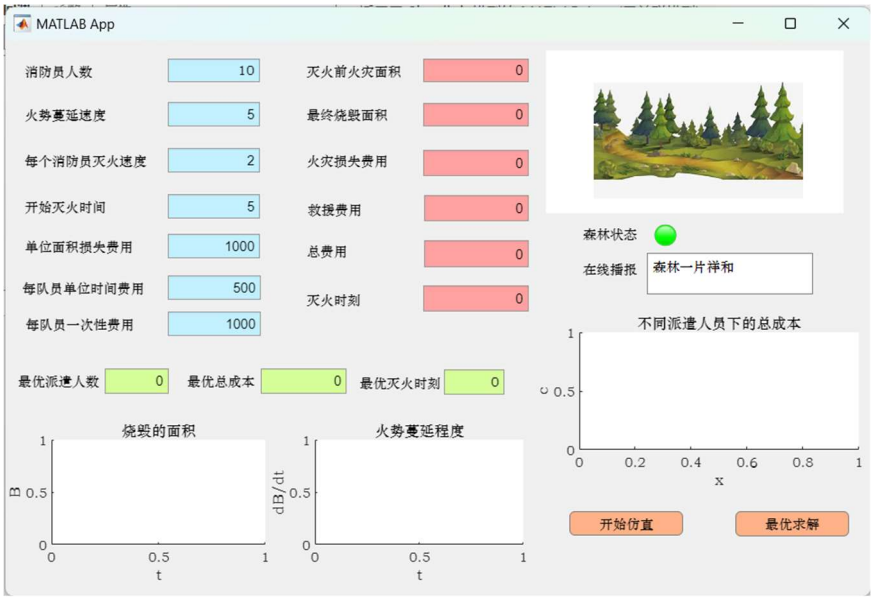
1. 回调函数的设计与参数传递：在 MATLAB APPDESIGNER 中，需要**利用回调函数将用户输入的参数传递给计算函数**，并将计算结果动态返回给输出显示与图像绘制部分。回调函数的正确设计对于功能的完整实现至关重要。

2. 动态仿真部分根据遗传算法得出的最优解，计算火灾蔓延过程，并实时绘制燃烧面积和火焰蔓延程度的变化曲线。同时，动态更新火焰图像，直观展示火势蔓延与扑灭效果。

4 程序运行指南

在工程下的命令行窗口输入 app 名称 **forest_fire**，或者双击打开 **forest_fire.mlapp** 后点击运行进行本程序的运行。

5 仿真模型的亮点



1. 高度集成化：将所有功能集成到一页图形化界面中

为了实现一个高度集成化的用户界面，采用了一个统一的布局，将仿真的输入、控制和输出全部集中在一个界面上。这样的设计可以让用户无需在多个窗口或页面之间切换，从而提高操作的便捷性和效率。界面将包含以下几个部分：

参数输入区域：用户可以在这里输入或选择仿真所需的所有参数，如火灾模拟的环境条件、消防员的初始人数等。

仿真控制区域：包含启动、暂停和重置仿真的按钮，以及可能的时间控制滑块，允许用户控制仿真的进度。

结果展示区域：动态显示仿真结果，包括图表和数据，用户可以直观地看到仿真的输出。

2. 过程的动态可视化：火灾和灭火过程随着时间的推移将动态显示

仿真设计了一个动态可视化系统，以实时反映火灾的发展和灭火的效果。这将通过以下方式实现：

动态图表：随着仿真的进行，系统将动态更新图表，展示火灾的扩散和消防员的灭火行动。

B-t 曲线和 dB/dt-t 曲线：这些曲线将展示火灾强度随时间的变化以及变化速率，帮助用户理解火灾的动态特性。

森林状态展示图：通过颜色编码或其他视觉提示，展示森林不同区域的火灾影响程度，提供直观的火灾影响范围视图。

3. **最优解的清晰化**：通过遗传算法求得最优派遣消防员人数，并绘制 c-x 曲线

为了清晰展示遗传算法找到的最优解，通过以下的方式来呈现：

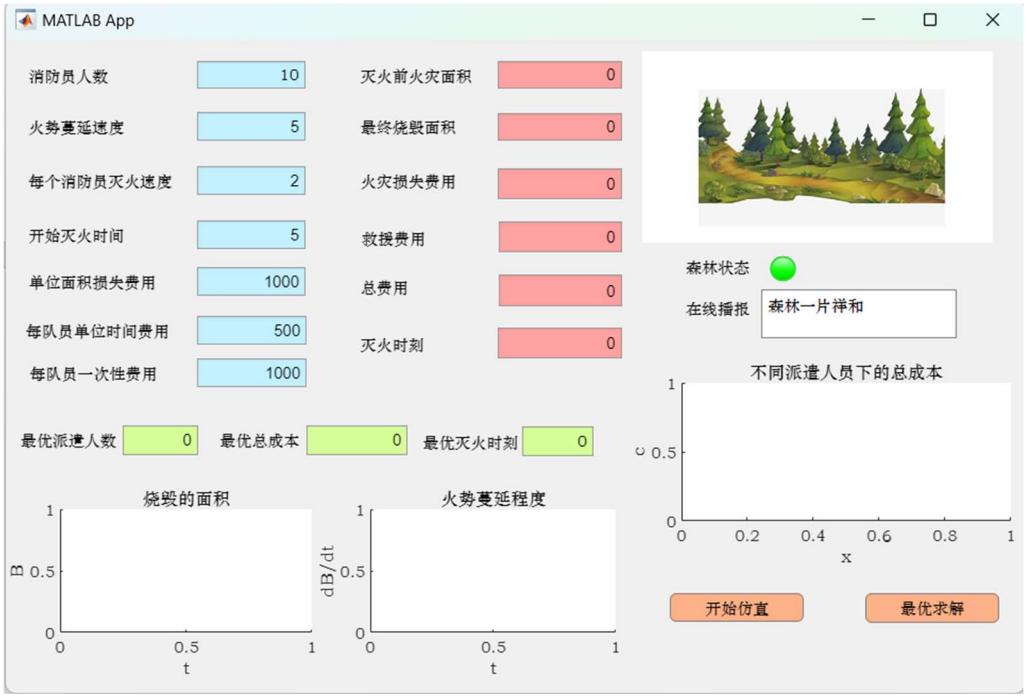
c-x 曲线：这条曲线将展示成本与派遣消防员人数之间的关系，帮助用户理解不同人数配置下的成本效益。

极小值成本突出显示：在 c-x 曲线上，将突出显示代表最优解的极小值点，让用户一眼就能看到成本最低的解决方案。

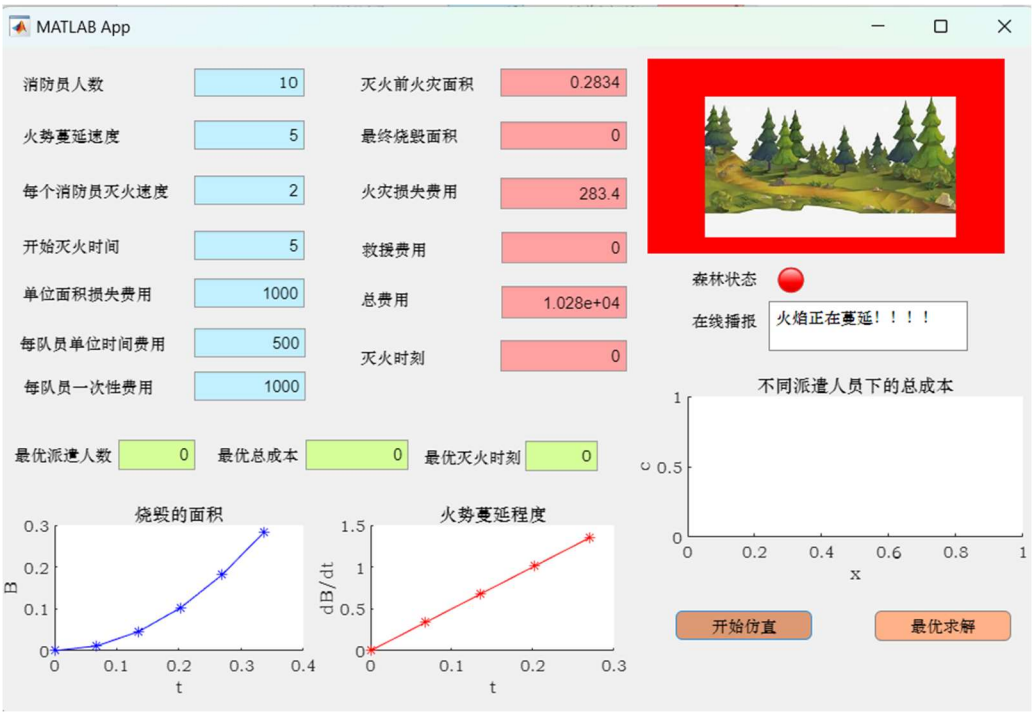
最优解参数展示：在界面上明确标出最优解对应的参数值，如最优的消防员人数，以及对应的最低成本。

6 程序运行实例

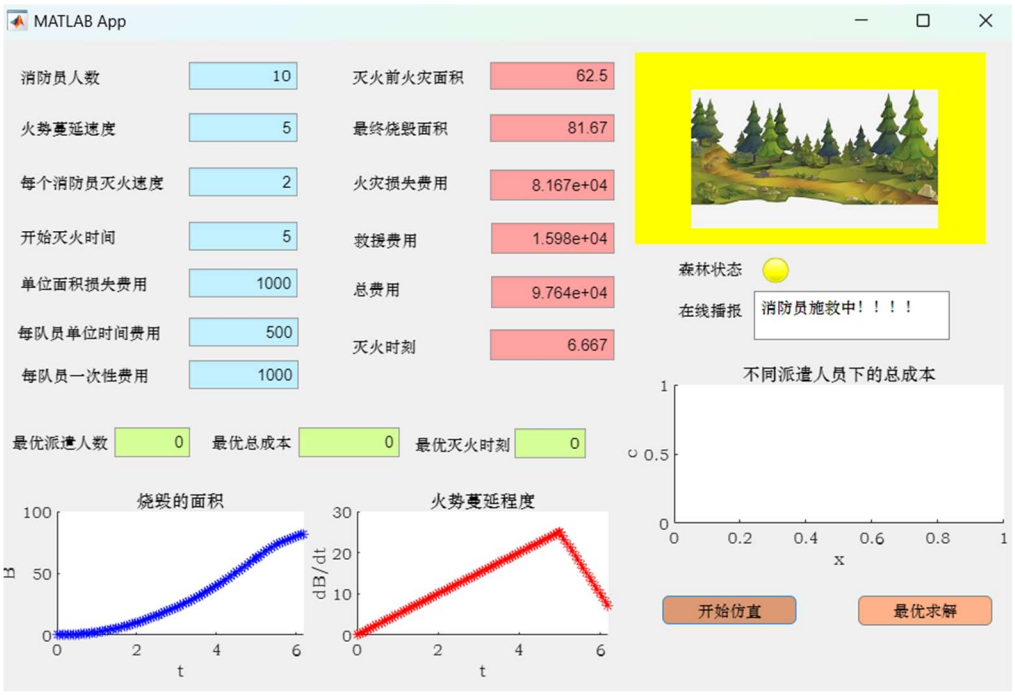
1. 主界面展示



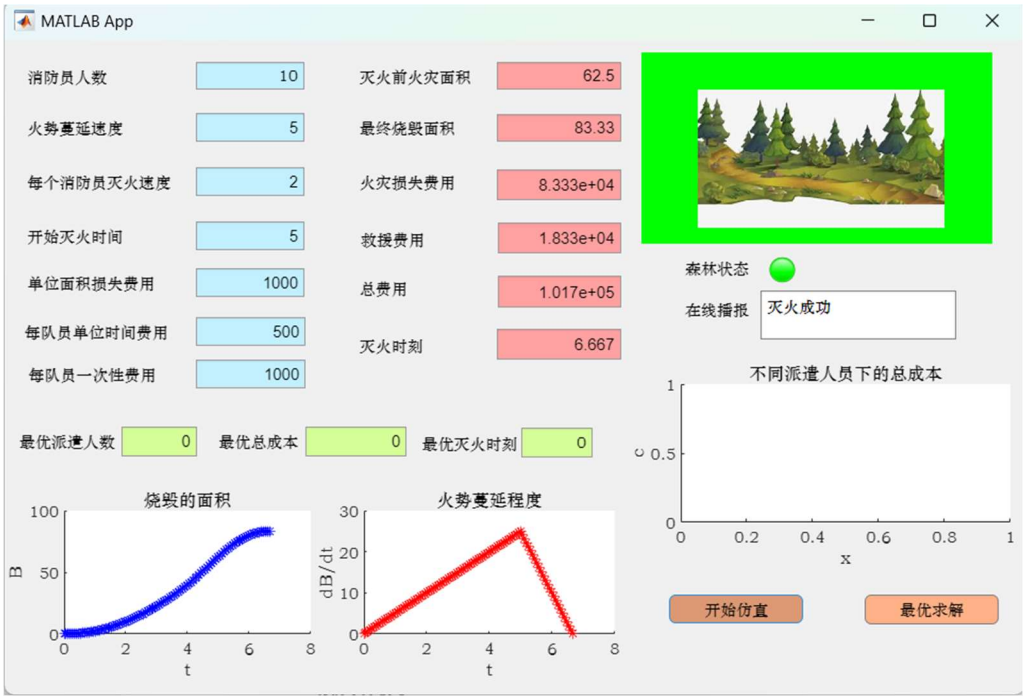
2. 着火过程



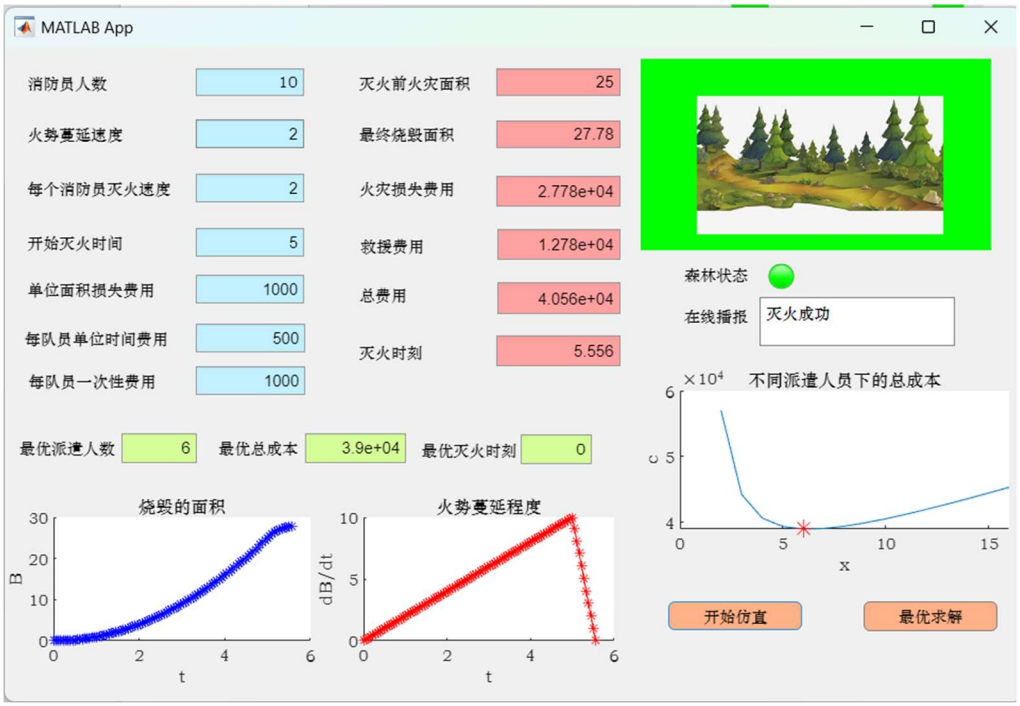
3. 消防员救火过程



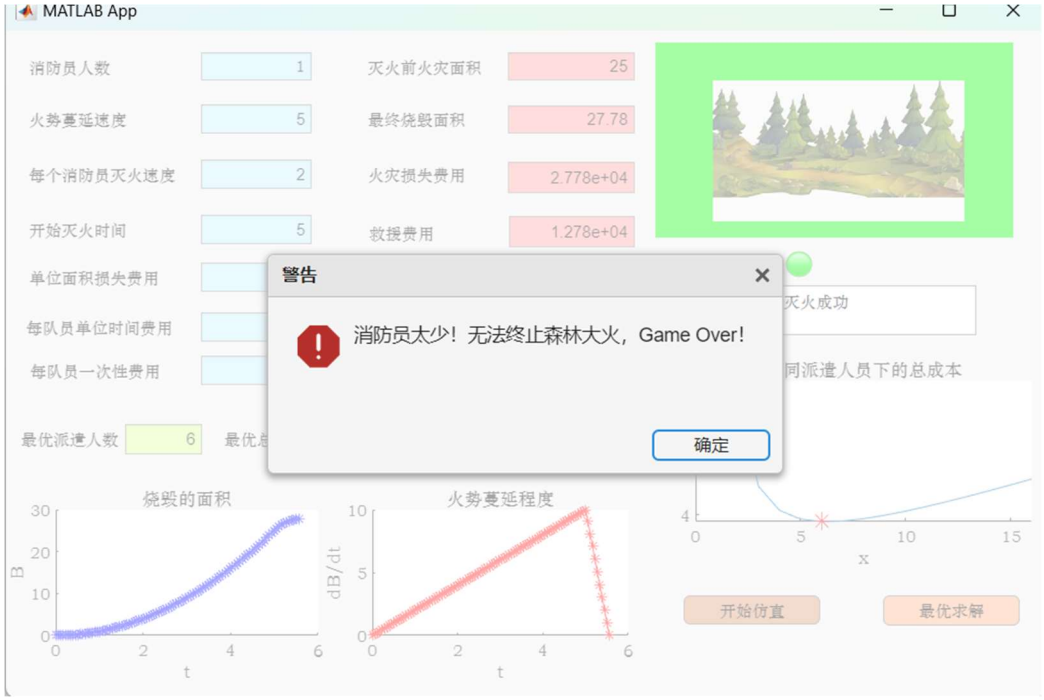
4. 消防员灭火



5. 最优派遣人数求解



6. 条件性约束



7 心得体会

本次实验还让我感受到了从理论到实践的转化过程。课程的目的不仅是让我掌握建模的基本工具，更是希望我能够学会如何用计算工具解决实际生活中遇到的问题。这个实验就是一个很好的案例，从设定问题、建立模型到实现仿真，每一步都需要细致的推敲和调试。在程序开发过程中，我不断碰到问题，比如函数传参失败、图形显示异常或逻辑错误等等，幸运的是，经过大量的查阅资料、自主探索以及尝试不同解决方案，大部分问题都被成功解决。这让我意识到，面对问题时冷静分析和多途径查找答案的能力是非常重要的。

这次实验不仅提升了我的 MATLAB 编程能力，还让我在分析问题和解决问题的思维上得到了极大的锻炼。从一开始对图形化界面开发的陌生，到最终能够熟练地设计一个功能齐全的图形化界面，这个过程让我感受到学习和成长的喜悦。同时，我也意识到，MATLAB 这样强大的工具，在建模、仿真和数据可视化上有着无与伦比的优势。

此外，本次实验也让我重新认识了数学建模的意义。它不仅仅是解决单一的数学问题，而是通过数学的语言和工具来描述现实中的复杂问题，从而找到最优解或合理的解决方案。这种抽象问题、建立模型、编程实现的全过程，既是一种技术能力的体现，也是逻辑思维与创造力的融合。

最后，我非常感谢王老师的悉心指导。他的教学内容和实验安排都极具针对性，让我在

学习中不仅仅停留于理论的掌握，还能够通过实践深入理解这些理论背后的实用性和逻辑性。在老师的引导下，我对 MATLAB 和数学建模的信心得到了极大的提升，也更加明白学以致用的重要性。这次实验是我学习历程中的一次重要突破，它让我意识到，不断深入探索未知领域，才是真正成长的关键[1]。

参考文献

[1] 刘光亚, "MATLAB 遗传算法在函数优化问题中的应用," 船电技术, vol. 42, no. 12, pp. 50-53, 2022.

[2] 段卿 and 林文, "基于 MATLAB App Designer 的激光雨滴谱仪数据质量控制软件开发及应用," 海峡科学, no. 04, pp. 113-117, 2024.

[3] 朱宏, "森林救火的优化模型," 吉林师范大学学报(自然科学版), no. 01, pp. 96-97, 2004.

部分代码附录

simulate_fire_process.m

```
function simulate_fire_process(app)
    % 获取输入参数
    x = app.edit_x.Value;
    beta = app.edit_beta.Value;
    lambda = app.edit_lambda.Value;

    % 检查是否消防员数量不足
    if beta > lambda * x
        uialert(app.fig, '消防员太少！无法终止森林大火，Game Over！', '警告');
        return;
    end

    t1 = app.edit_t1.Value;
    c1 = app.edit_c1.Value;
    c2 = app.edit_c2.Value;
    c3 = app.edit_c3.Value;

    % 计算灭火时间
    t2_solution = solve_t2(beta, lambda, x, t1);

    % 动态绘制火灾与灭火过程
    cla(app.ax);
    time = linspace(0, t2_solution, 100);  % 按默认方式生成时间点
    time = unique([time, t1]);   % 强制加入 t1，并确保没有重复的时间点
    time = sort(time);           % 对时间点进行排序

    situation = zeros(size(time));
    firing = zeros(size(time));

    for i = 1:length(time)
        if time(i) <= t1
            % 计算火灾蔓延程度和燃烧面积
```

```
            situation(i) =
calculate_burning_degree(beta,t1,la
mbda,x,time(i));  % 火灾蔓延程度
            firing(i) =
calculate_burning_area(beta,t1,lamb
da,x,time(i));       % 火灾燃烧面积
            area_burned_before =
firing(i);
            % 计算总成本
            [loss_fee, rescue_fee,
total_cost] =
calculate_cost(area_burned_before,
0, c1, c2, c3, x);

            % 更新输出结果

app.output_area_burned_before.Value
= area_burned_before;

app.output_loss_fee.Value =
loss_fee;

app.output_total_cost.Value =
total_cost;
            app.text.Value = "火焰正
在蔓延！！！！";
        else
            t_ext = time(i) - t1;
            % 计算火灾蔓延程度和燃烧面
积
            situation(i) =
calculate_burning_degree(beta,t1,la
mbda,x,time(i));
            firing(i) =
calculate_burning_area(beta,t1,lamb
da,x,time(i));

            area_burned_total =
firing(i);
            % 计算总成本
            [loss_fee, rescue_fee,
total_cost] =
calculate_cost(area_burned_total,
t_ext, c1, c2, c3, x);
            % 更新输出结果

app.output_total_area_burned.Value
= area_burned_total;

app.output_loss_fee.Value =
loss_fee;

app.output_rescue_fee.Value =
rescue_fee;

app.output_total_cost.Value =
total_cost;
            app.text.Value = "消防员
施救中！！！！";
        end

        % 绘制火灾与灭火过程的曲线
        plot(app.ax, time(1:i),
firing(1:i), 'b*-');
        drawnow;

        % 更新火焰燃烧程度曲线
        plot(app.ax_burning_degree,
time(1:i), situation(1:i), 'r*-');
        drawnow;

        % 动态调整图像尺寸（例如根据
fire_degree 进行调整）
        scale_factor =
max(situation(i) /
max(situation),1e-6);  % 根据火灾蔓延
程度计算缩放因子

        if (scale_factor<0.2)  % 红色
            app.Lamp.Color = [0, 1,
0];  % 变为绿色
            app.UIAxes.Color = [0,
1, 0];
        elseif
(scale_factor<0.8 )  % 绿色
            app.Lamp.Color = [1, 1,
0];  % 变为黄色
```

```
            app.UIAxes.Color = [1,
1, 0];
        else
            app.Lamp.Color = [1, 0,
0];  % 变为红色
            app.UIAxes.Color = [1,
0, 0];
        end

    end
    app.text.Value = "灭火成功";
    % 更新灭火完成时间
    app.output_t2.Value =
t2_solution;
end

function t2_solution =
solve_t2(beta, lambda, x, t1)
    % 解方程：beta * t1 + (beta -
lambda * x) * (t2 - t1) = 0
    syms t2;
    equation = beta * t1 + (beta -
lambda * x) * (t2 - t1) == 0;
    t2_solution =
double(solve(equation, t2));  % 转化
为数值解
end

% 计算火灾总成本
function [loss_fee, rescue_fee,
total_cost] =
calculate_cost(area_burned, t_ext,
c1, c2, c3, x)
    loss_fee = area_burned * c1;
    rescue_fee = c2 * x * t_ext + c3
* x;
    total_cost = loss_fee +
rescue_fee;
end

% 计算火灾燃烧面积
function burning_area =
calculate_burning_area(beta,t1,lamb
da,x,time)
    if time<t1
        burning_area = 0.5 * beta *
time^2;  % 火灾蔓延产生的燃烧面积
    else
        t_ext = time-t1;
        burning_area = 0.5 * beta *
t1^2 + beta * t1 * t_ext + 0.5 *
(beta - lambda * x) * (t_ext^2);

    end
end

% 计算火灾燃烧程度
function burning_degree =
calculate_burning_degree(beta,t1,la
mbda,x,time)
    if time<t1
        burning_degree = beta *
time;  % 火灾蔓延程度
    else
        t_ext = time-t1;
        burning_degree = beta * t1 +
(beta - lambda * x) * t_ext;
    end
end
```

calculate_min_cost.m

```
function calculate_min_cost(app)
% 从编辑字段获取参数值
    beta = app.edit_beta.Value;
    lambda = app.edit_lambda.Value;
    t1 = app.edit_t1.Value;
    c1 = app.edit_c1.Value;
```

```
    c2 = app.edit_c2.Value;
    c3 = app.edit_c3.Value;

    % 调用优化求解函数
    [x_optimal, min_cost, t2] = solve_optimize_constrained(beta, lambda,
t1, c1, c2, c3);

    % 更新输出结果
    app.output_min_cost.Value = min_cost;
    app.output_optimal_x.Value = x_optimal;
    app.output_t2.Value = t2;

    % 在最小点附近取 20 个点绘制曲线
    delta = round(x_optimal/10); % 确定取点的范围
    x_values = max(x_optimal - 10*max(delta,1), ceil(beta / lambda))  : 1 :
x_optimal + 10 * max(delta,1);

    % 计算每个点的目标值
    Z = arrayfun(@(x) objective(x, t1, beta, lambda, c1, c2, c3),
x_values);
    cla(app.ax_optimal);
    % 画图
    plot(app.ax_optimal, x_values, Z);

    % 标记最小点
    hold(app.ax_optimal, 'on');
    plot(app.ax_optimal, x_optimal, min_cost, 'r*', 'MarkerSize', 10);
    hold(app.ax_optimal, 'off');
end

function cost = objective(x, t1, beta, lambda, c1, c2, c3)
    % 目标函数，x 是优化变量，t1 是输入
    t2 = (lambda * x / (lambda * x - beta)) * t1;
    t_ext = t2 - t1;
    burned_area = ((beta * t1^2) / 2 + beta * t1 * t_ext + 0.5 * (beta -
lambda * x) * (t_ext^2));

    % 计算成本
    cost = c1 * burned_area + c2 * x * t_ext + c3*x;
end

function [x_optimal, min_cost, t2] = solve_optimize_constrained(beta,
lambda, t1, c1, c2, c3)
    % 目标函数
```

```
    objective_func = @(x) objective(x, t1, beta, lambda, c1, c2, c3);

    % 设置遗传算法选项
    options = optimoptions('ga', 'Display', 'iter', 'MaxGenerations', 100,
'MaxStallGenerations', 50, 'PopulationSize', 50, 'CrossoverFraction', 0.8,
'EliteCount', 2);

    % 设置遗传算法约束：x >= ceil(beta / lambda) + 1，并且 x 必须为整数
    lb = ceil(beta / lambda); % 最小值
    ub = 10000; % 假设上限为 100，你可以根据实际情况调整

    % 求解：使用遗传算法
    [x_optimal, ~] = ga(objective_func, 1, [], [], [], [], lb, ub, [],
options); % 1 代表我们优化的是一个变量 x

    % x 四舍五入
    x_optimal = round(x_optimal);
    min_cost = objective(x_optimal, t1, beta, lambda, c1, c2, c3);

    % 计算 t2
    t2 = (lambda * x_optimal / (lambda * x_optimal - beta)) * t1;

    % 输出最优结果
    disp(['Optimal x: ', num2str(x_optimal)]);
    disp(['Minimum cost: ', num2str(min_cost)]);
end
```