基于xxxx的烟幕干扰弹投放策略研究

摘 要

开头段：需要充分概括论文内容，一般两到三句话即可，长度控制在三至五行。

问题一中，解决了什么问题；应用了什么方法；得到了什么结果。

问题二中，解决了什么问题；应用了什么方法；得到了什么结果。

问题三中，解决了什么问题；应用了什么方法；得到了什么结果。

结尾段：可以总结下全文，也可以介绍下你的论文的亮点，也可以对类似的问题进行适当的推广。

关键词：**无人机投弹；运动学建模；粒子群优化算法；遗传算法**

|  |
| --- |
| **注意事项：**  本Word模版的版本编号是0.2版，是以国赛的标准创建的模版，也适用于国内绝大多数的数学建模比赛。模版的使用方法可以查看下面这个视频：  数学建模清风——论文排版教程  <https://www.bilibili.com/video/BV1Ci4y1c7Ld>  未来当发现模版中的问题或者比赛要求有更新时，我会发布更新后的新版本。大家可以在公众号《数学建模学习交流》后台发送“国赛论文模版”获取最新的模版的信息。只要本文档的版本号和公众号后台最新的版本号一致，则说明你下载的是最新版本。  关于具体怎么写好数学建模论文的每一部分，可以看下面这个视频：  <https://www.bilibili.com/video/BV1Na411w7c2/>  **红色字体的文字是上面这个视频中的笔记，在实际论文中不要出现。**  首页三要素: 论文标题 + 摘要 + 关键词  （1）标题：   * 基于所使用的主要模型或者方法作为标题（推荐） * 直接使用赛题所给的题目或者要研究的问题作为标题   （2）摘要：  摘要是数模论文写作中最重要的一部分，因为评阅老师的时间有限，拿到一篇论文后不会完整的从头读到尾，所以评阅老师往往会重点阅读摘要部分，并结合官方的评阅要点来对你的论文进行初步评定。因此，大家一定要好好打磨论文的摘要，摘要一般是其他部分都完成后再来书写，写完后需要反复阅读反复修改。  （3）关键词：  关键词一般放4-6个，可以放论文中使用的主要模型，也可以放论文里面出现次数较多，能体现论文的主要内容的词。 |

# 问题重述

烟幕干扰弹是一种通过在目标空域形成烟幕或气溶胶云团遮蔽，以干扰敌方导弹对我方特定目标进行攻击的武器，其成本低、效率高。使用无人机挂载烟幕干扰弹可以实现对其的定点精确投放与时序起爆控制。为使来袭导弹与保护目标之间形成有效烟幕遮蔽，无人机投放烟幕干扰弹的策略至关重要。

在本题中，单台或多台无人机执行投放任务，要在来袭导弹与真目标之间形成有效遮蔽。保护目标为半径 7 m、高 10 m 的圆柱体，底面中心位于 ；初始位置已知的三枚导弹M1，M2，M3以的速度来袭，初始方向朝向位于坐标系原点的假目标；五架无人机初始位置也已知，每架可投放一枚或多枚干扰弹；烟幕云团以3 m/s 匀速下沉，起爆后20秒内在其中心10米范围内可有效遮蔽目标。无人机可在70–140 m/s 范围内调整速度并匀速直线飞行。

需依次解决以下问题：

**问题1**：在给定条件下使用FY1单弹干扰M1并计算有效遮蔽时长；

**问题2**：优化FY1单弹投放策略以延长遮蔽时间；

**问题3**：利用FY1投放三弹干扰M1的投放策略；

**问题4**：FY1-FY3各投一弹干扰M1的协调投放策略；

**问题5**：利用全部FY1-FY5五架无人机（每架至多投放三弹）同时对M1-M3三枚导弹进行干扰，给出最优投放策略。

# 问题分析

## 问题一

本问题中，利用无人机 FY1 投放 1 枚烟幕干扰弹实施对 M1 的干扰，需要在给定无人机飞行速度、飞行方向、干扰弹投放时间与起爆时间间隔的条件下，计算烟幕干扰弹对M1的有效遮蔽时长。

在本题中，可将圆柱形真目标视为质点，烟幕干扰弹爆炸后生成球形烟幕匀速下沉，导弹在三维空间中匀速直线的过程中，导弹与目标质点的连线经过球形烟幕这一过程的时长即为烟幕干扰弹对M1的有效遮蔽时长。

由题目条件建立导弹与球形烟幕三维运动学模型，进而联立运动学方程，对时间进行离散化遍历，最终得出总的有效遮蔽时长。

## 问题二

本问题中，利用无人机 FY1 投放 1 枚烟幕干扰弹实施对 M1 的干扰，需要在问题1的基础上，优化 FY1 的飞行方向、飞行速度、烟幕干扰弹投放点、烟幕干扰弹起爆点，使得遮蔽时间尽可能长。

本文将FY1飞行方向、飞行速度、飞行过程中烟幕干扰弹投放时间和起爆时间作为决策变量，得到以遮蔽时长为目标的非线性优化模型。首先，考虑无人机匀速直线飞行、干扰弹投出后做平抛运动及球形烟幕匀速下沉，建立无人机、云团和导弹的空间运动学模型，分别描述无人机飞行、干扰弹抛物线下落及爆炸后球状云团匀速下沉过程，以及导弹的直线飞行轨迹。接着，与问题1采用相同的空间几何遮蔽判据，即判断导弹-真目标连线与云团中心的空间距离是否小于云团半径，并通过数值仿真方法统计遮蔽时长。最后，将遮蔽时长作为目标函数，对投放参数进行优化。

参数优化过程中，采用粒子群优化算法（PSO），在参数空间内搜索无人机投放时间、速度和角度的最优组合。每一组参数通过模型仿真得到目标遮蔽时长，PSO根据全局和个体最优动态调整搜索方向，最终获得最大遮蔽时长及其对应参数。

## 问题三

本问题中，需利用无人机FY1投放3枚烟幕干扰弹，实施对M1的干扰，给出最有投放策略。

本题中，FY1飞行方向、飞行速度、飞行过程中三枚烟幕干扰弹分别的投放时间和起爆时间需作为作为决策变量，基于问题1和问题2的运动学分析可以建立无人机、云团和导弹的空间运动学模型，分别描述无人机飞行、干扰弹抛物线下落及爆炸后球状云团匀速下沉过程，以及导弹的直线飞行轨迹。相对于前两个问题，优化问题高度非线性、变量多且目标函数不可导，我们采用了遗传算法（GA）进行全局搜索，将时间速度角度等离散数组作为优化变量，并且进行局部优化最终得到最优的协同投放策略。

## 问题四

## 问题五

# 模型假设

1. 烟幕云团起爆后瞬时形成，且为理想球体，下沉速度为恒定3 m/s，有效遮蔽范围为云团中心10米半径内，持续时间20秒；
2. 每架无人机投放两枚干扰弹的时间间隔至少为1秒；
3. 导弹从初始坐标开始以300 m/s 的速度朝假目标方向匀速直线运动；
4. 烟幕干扰弹在投放后起爆前的下落过程中为自由落体运动，忽略空气阻力；
5. 无人机飞行速度在70–140 m/s 范围内连续可调节，且调节后立即生效；
6. 多个烟幕云团之间互不干扰，独立形成遮蔽效果，即使重叠遮蔽效果仍生效；

# 符号说明

|  |  |  |
| --- | --- | --- |
| **符号** | **说明** | **单位** |
|  | 导弹飞行速度 |  |
|  | 球状烟幕云团下沉速度 |  |
|  | 无人机飞行速度 |  |
|  | 时间变量 |  |
|  | 干扰弹投放时间 |  |
|  | 干扰弹起爆时间 |  |
|  | 重力加速度 |  |
|  | 导弹位置（t时刻） |  |
| (t) | 烟幕干扰弹位置（t时刻） |  |
|  | 球状烟幕中心位置（t时刻） |  |
|  | 无人机位置（t时刻） |  |
|  | 球状烟幕半径 |  |
|  |  |  |
|  |  |  |

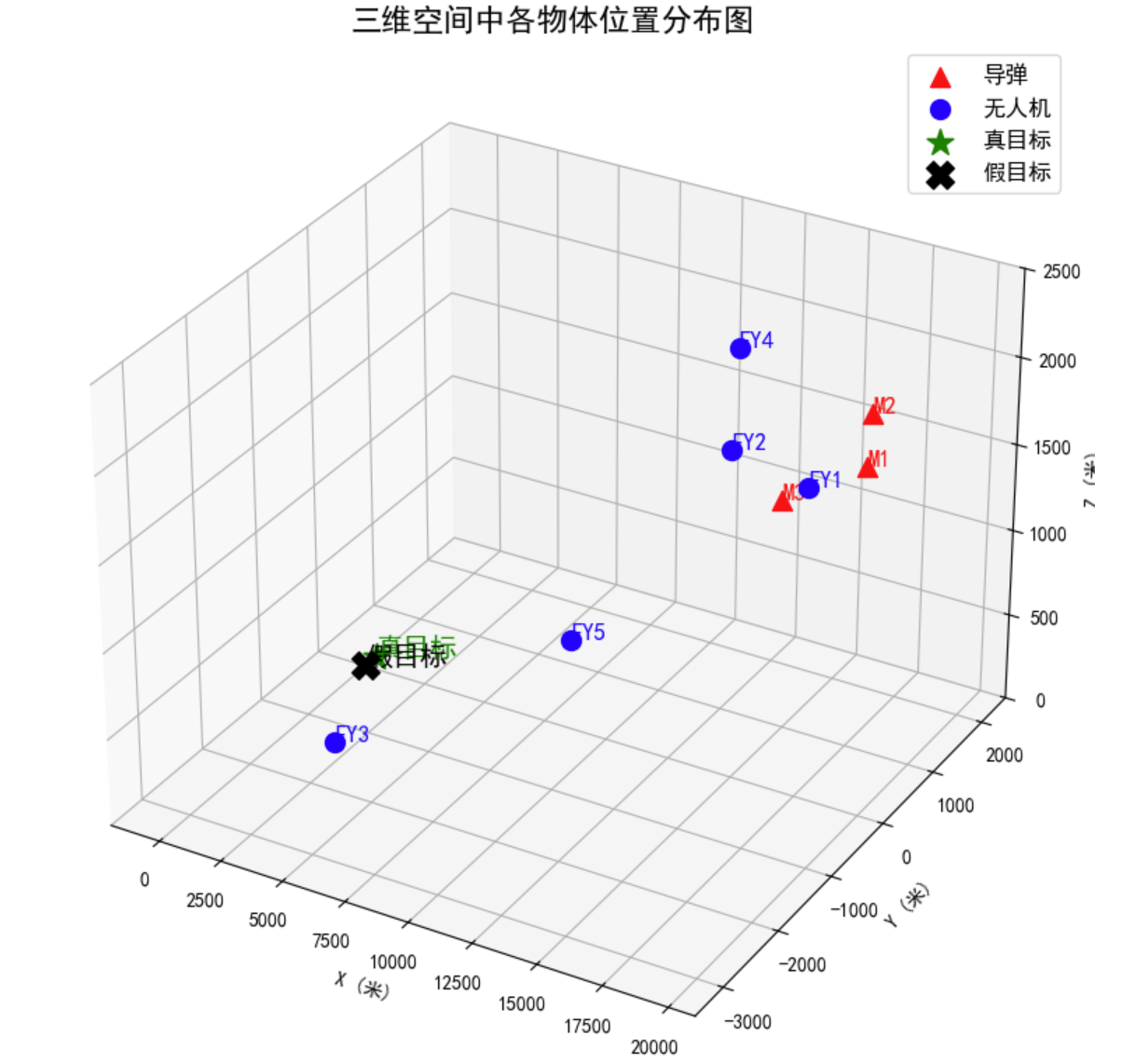
注：其他符号的释义后续会在文章中给出。

# 模型的建立与求解

## 问题一：特定条件下单机单弹干扰的有效遮蔽时长求解

### 各物体初始运动模型的建立

以假目标为原点建立空间坐标系，各物体初始位置如图1。导弹 M1、M2、M3初始分别位于 (20000, 0, 2000)、(19000, 600, 2100)、(18000, −600, 1900)，将沿直线弹道以300m/s飞向原点。五架无人机FY1至FY5分别部署于（17800,0,1800）、（12000,1400,1400）、（6000,-3000,700）、（11000,2000,1800）、（13000,-2000,1300）米的空间位置；而需要遮蔽的真目标位于（0,200,0）处，单位均为米（m）。

图1 各物体初始位置分布图

1. **导弹M1运动分析**

导弹 M1 的运动轨迹采用匀速直线运动模型进行描述。在该模型中，可假设导弹在飞行过程中不受任何外力干扰，因此其运动状态保持恒定，运动轨迹为一条直线；假目标的位置为三维坐标系原点O (0,0,0)，导弹的初始位置为 (20000, 0, 2000)；导弹的飞行速度大小，其飞行方向精确指向假目标原点O。

1. **无人机FY运动分析**

无人机 FY1 的运动为水平匀速直线运动，可假设无人机在飞行过程中保持固定高度，且以恒定速度沿直线飞行，忽略空气动力学效应及自身机动变化等影响因素； 无人机初始位置位于三维坐标系中的点，其初始位置位于𝑋𝑂𝑍平面内；无人机飞行速度大小，飞行方向精确指向假目标所在位置即坐标系原点𝑂。

在问题1中，无人机在受领任务后 投放烟幕弹。烟幕弹投放时刻的位置取决于该时刻无人机的位置。根据匀速直线运动模型，投放时刻位置为：

1. **烟幕干扰弹投放后起爆前运动分析**

此时对烟幕弹进行受力分析，其只受重力的影响，因此在竖直方向存在加速度，同时烟幕弹竖直方向的初速度；在水平方向，由于惯性的影响，烟幕弹保持投放时的水平初速度不变，在不考虑空气阻力的情况下，该水平方向的速度为无人机的飞行速度向量，因此此时烟幕弹脱离无人机后，在重力作用下作平抛运动，其水平位置为：

竖直位置为：

其中为无人机的初始高度，即。

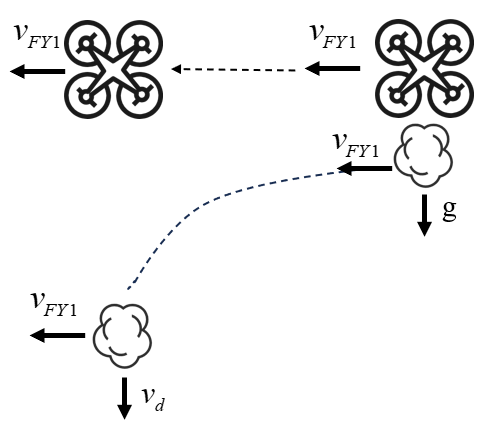
根据以上方程，最终可求解对应的烟幕弹位置。

图2 烟幕起爆位置分析图

1. **起爆后球形烟幕运动分析**

在问题1中，烟幕弹投出后经过后起爆，起爆瞬间形成初始球状云团并扩散达到最大，此时烟幕云团呈的球形，球心与起爆点重合，此时的烟幕起爆位置如图2所示。

烟幕云团形成后，其在水平方向保持静止，X、Y坐标不变；在竖直方向上，球形烟幕以向下运动。

### 有效遮蔽的判定条件

烟幕干扰弹的战术目的是在导弹与真目标之间形成视觉屏障，只要烟幕云团挡住了导弹指向真目标的“视线”，即导弹与真目标的连线穿过烟幕，导弹就无法通过自身的光学手段锁定真目标，从而实现了有效遮蔽。

设真目标坐标表示为T，导弹坐标为，球状烟幕中心坐标为，则“视线遮挡”的数学判据为线段与烟幕球体相交，且该时刻应当在烟幕干扰弹起爆后20s内方可称为有效遮蔽。

球形烟幕球心到导弹W与真目标T连线直线MT的距离为：

上文有效遮蔽的判定条件可表示为：；

时间限制条件：。

### 模型求解

求解以上模型，对于约束的时间段用均匀时间步长进行离散化遍历，在每一时刻，判断有效遮蔽条件是否成立；统计所有被遮蔽时刻总时长，得到总的有效遮蔽时长。

最终得到有效遮蔽时长 。

## 问题二：单机单弹干扰的最优遮蔽策略

### 决策变量及其约束条件

在题目1的基础上，我们设置决定有效遮蔽时长的决策变量如下：

无人机FY1飞行方向角: 无人机在水平面上的飞行方向与轴正方向的夹角，则。则无人机FY1飞行方向单位向量为；

无人机飞行速度v，；

干扰弹投放时间；

干扰弹起爆时间，；

要想起到遮蔽作用，烟幕干扰弹必须在落地前起爆，即确保烟幕弹在起爆时刻的高度不低于零，约束如下：

### 建立各物体运动学模型

除几个决策变量未确定外，各物体运动状态与题目1中相同，因此可以得到以下物体的运动学方程。

1. 导弹M1运动学方程（与问题1相同）：
2. 无人机FY1运动学方程（投放干扰弹前）：
3. 烟幕干扰弹抛出后平抛运动：
4. 烟幕干扰弹爆炸后球状烟幕匀速下落：
5. 遮蔽条件及其判断公式也与问题1相同

### 模型求解

粒子群优化算法（Particle Swarm Optimization, PSO）是一种基于群体智能的全局优化算法。其基本思想是模拟鸟群、鱼群等生物群体在搜索食物时的协作与信息共享行为。PSO通过群体中多个粒子（即候选解）的协同搜索，动态调整粒子的位置和速度，从而在解空间中寻找最优解。

本题涉及多个参数的联合优化，目标函数复杂且无法解析表达，传统解析方法难以适用；且本体数学模型模型需通过数值仿真计算目标函数值，目标函数不可导；而PSO无需梯度信息，适合此类黑箱优化问题；其次， PSO具有全局搜索能力强、实现简便、收敛速度快等优点，能够有效避免陷入局部最优。因此，我们选用了PSO算法来高效、稳定地求解本题的最优参数组合。

粒子群优化算法的流程如图3，首先初始化随机初始化粒子群的各个粒子在参数空间中的位置和速度，并设置算法参数；其次进行适应度评价，即对每个粒子，根据模型目标函数（本题为负的遮蔽时长）计算其适应度值；然后更新个体与全局最优，对比每个粒子的当前适应度与历史最优适应度，更新个体最优位置，同时更新全局最优位置；并按上述公式更新每个粒子的速度和位置，保证其不超过参数空间边界；最终达到最大迭代次数或收敛条件后终止，输出全局最优解。

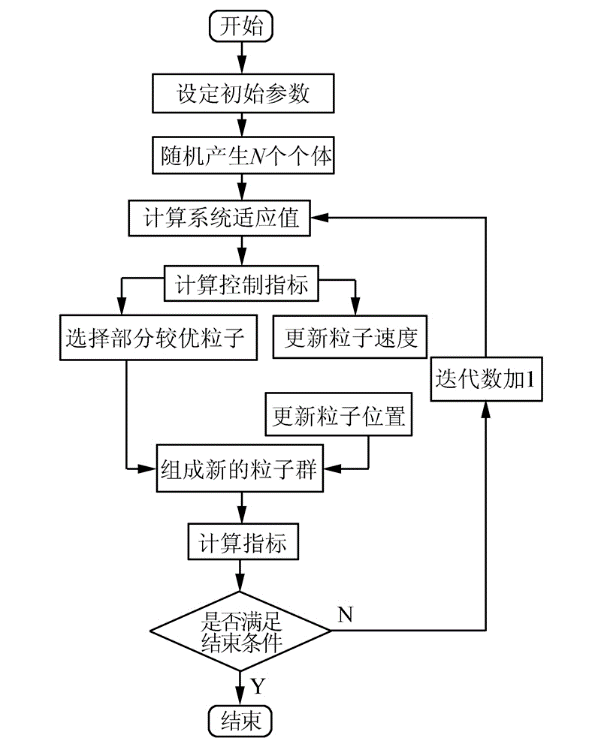


图3 PSO算法流程图

在本题中，粒子的每一组参数对应一次具体的云团投放方案（包括投放时刻、爆炸时刻、无人机飞行速度、角度等）。适应度函数为遮蔽时长的负值，PSO搜索参数空间，最终获得能最大化遮蔽时间的最优投放策略。通过仿真可以得到最优变量组合及对应的最大遮蔽时长。

经过多次尝试和迭代，我们将PSO算法的参数设置如下：

粒子数：400；

最大迭代次数：150；

惯性权重(w)初始为0.9，迭代线性递减至0.4；

学习因子：；

算法的搜索空间为：投放延迟、爆炸延迟、球状云团生效时间、无人机速度、投放角度等参数的离散索引。

通过迭代，得到的最优决策变量如下：

无人机FY1飞行方向角；

无人机飞行速度；

干扰弹投放时间；

干扰弹起爆时间；

对应的最大遮蔽时长为：

## 问题三：单机多弹干扰的最优遮蔽策略

### 模型的建立

基于问题1和问题2，可知三枚干扰弹每枚在投放后按给定水平初速度做平抛运动，引爆后形成球状烟幕云团，云团随时间匀速下沉，通过引爆坐标和下沉速度可以计算云团中心位置。

三各球状烟幕云团，其中对于每个云团的遮蔽判据也与前两问相同，可分别判定某时刻导弹M1“视线”是否被云团遮蔽。

可以设定决策变量及其约束如下：

无人机FY1飞行方向角，；

无人机飞行速度v，；

干扰弹投放时间，，，，且；

干扰弹起爆时间，，，。

球形干扰云团生效区间，，。

所有参数均需在约束区间内离散化，以便于后续算法实现。

对于每一组参数，可以模拟每个时间步下三枚云团与导弹与目标连线的空间关系，统计所有被遮蔽的时间步长之和，得到有效遮蔽总时长。

### 模型的求解

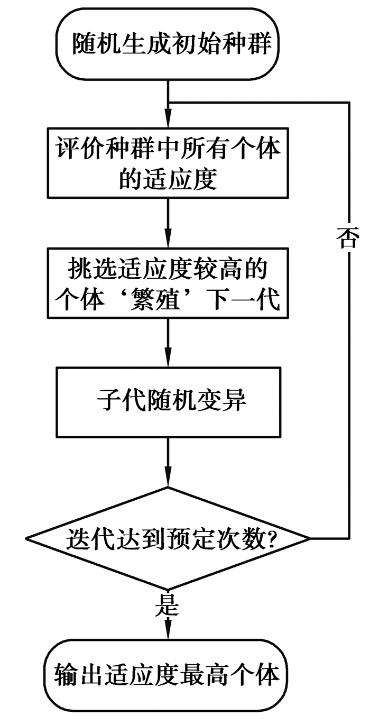
遗传算法（Genetic Algorithm, GA）是一种基于自然选择和遗传机制的群体智能优化方法，其基本思想来源于达尔文生物进化理论。算法通过对一组候选解（称为“个体”或“染色体”）进行迭代优化，在种群之间不断进行选择、交叉和变异操作，从而在解空间中搜索最优解，遗传算法一般流程如图4所示。

图4 GA算法流程图

本题优化变量多、约束复杂、目标函数无解析表达且不可导，适合采用遗传算法这样的群体智能优化方法。遗传算法无需目标函数导数信息，具备全局搜索能力，易于与运动学模型和有效遮蔽的复杂判据结合。

在本题中先对变量编码与初始化，将全部11个决策变量离散为索引编码，个体为长度11的整数数组；让种群一半个体在参数中值附近微扰生成，另一半随机生成，以提高多样性和收敛效率；接着对每个个体调用适应度函数计算遮蔽总时长，比较其总时长（适应度）每代保留前10%适应度的最优个体，并采用锦标赛选择产生父代，提高优秀基因传递概率；然后以0.85的概率执行单点交叉，交换父代基因片段，增强多样性；同时以0.45概率变异，对适应度低个体多基因大幅变异，高适应度个体仅小幅扰动，防止陷入局部最优。

在一轮演化结束后，使用了局部爬山优化的方法，对当前最优个体进行10轮爬山法微调，进一步提升解的精细度和局部最优性。

最终演化达到最大进化代数（250代）后，输出最优参数组及最大遮蔽时长。

得到的最优参数组为：

；v=126.906474m/s;

，，；

，，；

此时最大遮蔽时长为。

### 模型的求解

### 模型的求解

### 模型的求解

|  |  |
| --- | --- |
| 这里插入公式 | () |
| 问题四：多机单弹干扰的最优遮蔽策略模型的求解模型的求解模型的求解模型的求解模型的求解问题三模型的建立与求解模型的求解模型的求解模型的求解模型的求解模型的求解  |  |  | | --- | --- | | 这里插入公式 | () | |  |

# 模型的分析与检验

模型的分析与检验的内容也可以放到模型的建立与求解部分，这里我们单独抽出来进行讲解，因为这部分往往是论文的加分项，很多优秀论文也会单独抽出一节来对这个内容进行讨论。

模型的分析 ：在建模比赛中模型分析主要有两种，一个是灵敏度(性)分析，另一个是误差分析。灵敏度分析是研究与分析一个系统（或模型）的状态或输出变化对系统参数或周围条件变化的敏感程度的方法。其通用的步骤是：控制其他参数不变的情况下，改变模型中某个重要参数的值，然后观察模型的结果的变化情况。误差分析是指分析模型中的误差来源，或者估算模型中存在的误差，一般用于预测问题或者数值计算类问题。

模型的检验：模型检验可以分为两种，一种是使用模型之前应该进行的检验，例如层次分析法中一致性检验，灰色预测中的准指数规律的检验，这部分内容应该放在模型的建立部分；另一种是使用了模型后对模型的结果进行检验，数模中最常见的是稳定性检验，实际上这里的稳定性检验和前面的灵敏度分析非常类似，等会大家看到例子就明白了。

(大家尽量在论文中使用灵敏度分析，视频中有详细的讲解)

# 模型的评价、改进与推广

注：本部分的标题需要根据你的内容进行调整，例如：如果你没有写模型推广的话，就直接把标题写成模型的评价与改进。很多论文也把这部分的内容直接统称为“模型评价”部分，也是可以的。

## 模型的优点

优缺点是必须要写的内容，改进和推广是可选的，但还是建议大家写，实力比较强的建模者可以在这一块充分发挥，这部分对于整个论文的作用在于画龙点睛。

## 模型的缺点

缺点写的个数要比优点少

## 模型的改进

主要是针对模型中缺点有哪些可以改进的地方；

## 模型的推广

将原题的要求进行扩展，进一步讨论模型的实用性和可行性。

# 参考文献

1. 于立君, 陈佳, 刘繁明, 王辉. 改进粒子群算法的PID神经网络解耦控制[J]. 智能系统学报, 2015, 10(05): 699-704.
2. 马创, 王尧, 李林峰. 基于遗传算法与支持向量机的水质预测模型. 重庆大学学报, 2021, 44(7): 108-114.
3. 罗瑞耀,王得霖,罗威,等.烟幕弹应对察打一体无人机的投放策略研究[J].光电技术应用,2022,37(06):90-98.

网上资源(例如数据库、政府报告)的表述方式为： [编号] 作者，资源标题，网址，访问时间。

附录

|  |
| --- |
| 附录1 |
| 支撑材料的文件列表 |
| 这是最近国赛要求加入的一个部分，大家可以看我讲的论文写作视频。  <https://www.bilibili.com/video/BV1Na411w7c2> |

|  |
| --- |
| 附录2 |
| 问题1 python程序 |
| import numpy as np    *# 时间参数*  t = np.linspace(5.1, 25.1, 2000)  *# 云团有效时间段（单位: 秒），共20秒*  *# 真目标中心（圆柱底面圆心）*  T\_center = np.array([0, 200, 5])  遮蔽时长 = 0  dt = t[1] - t[0]  *# 步长，大约0.01秒*  for t\_val in t:      tau = t\_val - 5.1  *# 起爆后经过的时间*    *# 云团中心坐标*      C = np.array([17188, 0, 1736.496 - 3 \* tau])    *# M1 当前位置*      m1\_x = 20000 - 3000 / np.sqrt(101) \* t\_val      m1\_y = 0      m1\_z = 2000 - 300 / np.sqrt(101) \* t\_val      P\_m1 = np.array([m1\_x, m1\_y, m1\_z])    *# 构造向量*      PT = T\_center - P\_m1      PC = C - P\_m1    *# 叉乘，点到直线距离*      cross = np.cross(PT, PC)      norm\_PT = np.linalg.norm(PT)      d = np.linalg.norm(cross) / norm\_PT if norm\_PT > 0 else np.inf    *# 投影参数，判断最近点是否在线段上*      proj = np.dot(PC, PT) / (norm\_PT \*\* 2 + 1e-7)        if d <= 10 and 0 <= proj <= 1:          遮蔽时长 += dt    print(f"有效遮蔽时长：{遮蔽时长:.1f} 秒") |

|  |
| --- |
| 附录3 |
| 问题2 python程序 |
| import numpy as np  *# 位置参数*  x0 = 17800  y0 = 0  z0 = 1800  *# 速度参数*  v0 = 70  *# 角度参数*  angle = 0  *# 时间参数*  t0 = 6.0  t1 = 8.0  *# 投放时间间隔*  tq = np.linspace(0, 5, 100)  *#爆炸延时*  tp = np.linspace(5, 8, 100)  *# 云团有效时间段（单位: 秒），共20秒*  t = np.linspace(8, 28, 200)  *# 速度范围*  v = np.linspace(70, 140, 140)  *# 角度范围*  agl = np.linspace(0, 90, 180)  *# 真目标中心（圆柱底面圆心）*  T\_center = np.array([0, 200, 5])  *# 注意z取0，题意中底面圆心*  *# # 预计算 M1 的位置*  *# m1\_x = 20000 - 3000 / np.sqrt(101) \* t*  *# m1\_y = np.zeros\_like(t)*  *# m1\_z = 2000 - 300 / np.sqrt(101) \* t*  *# P\_m1 = np.vstack((m1\_x, m1\_y, m1\_z)).T*  *#*  *# # 预计算 PT*  *# PT = T\_center - P\_m1*  *# 时间步长*  dt = t[1] - t[0]  *# 步长，大约0.01秒*  *# 定义目标函数*  def objective\_function(params, dt):  *# 解码参数索引*      t\_val0\_idx = int(np.round(params[0]))      t\_val1\_idx = int(np.round(params[1]))      t\_val\_idx = int(np.round(params[2]))      v\_0\_idx = int(np.round(params[3]))      a\_idx = int(np.round(params[4]))  *# 获取实际值*      t\_val0 = tq[t\_val0\_idx]      t\_val1 = tp[t\_val1\_idx]      t\_val = t[t\_val\_idx]      v\_0 = v[v\_0\_idx]      a = agl[a\_idx]  *# x0 = 17800 - v0 \* np.cos(angle) \* t0*  *# y0 = v0 \* np.sin(angle) \* t0*  *# z0 = 1800*  *#*  *# x1 = x0 - v0 \* np.cos(angle) \* (t1 - t0)*  *# y1 = y0 + v0 \* np.sin(angle) \* (t1 - t0)*  *# z1 = 1800 - 0.5 \* 9.8 \* (t1 - t0) \* (t1 - t0)*  *# tau = t\_val - t1*  *#*  *# C = np.array([x1, y1, z1 - 3 \* tau])  # 云团中心坐标*  *# 初始化遮蔽时长*      shielding\_time = 0      for current\_t in np.arange(t\_val1,t\_val, dt):          x0 = 17800 - v\_0 \* np.cos(a) \* t\_val0          y0 = v\_0 \* np.sin(a) \* t\_val0          z0 = 1800          x1 = x0 - v\_0 \* np.cos(a) \* (t\_val1 - t\_val0)          y1 = y0 + v\_0 \* np.sin(a) \* (t\_val1 - t\_val0)          z1 = 1800 - 0.5 \* 9.8 \* (t\_val1 - t\_val0) \* (t\_val1 - t\_val0)          tau = t\_val - t\_val1          C = np.array([x1, y1, z1 - 3 \* tau])  *# 云团中心坐标*  *# 计算 M1 实时位置*          m1\_x = 20000 - 3000 / np.sqrt(101) \* current\_t          m1\_z = 2000 - 300 / np.sqrt(101) \* current\_t          P\_m1 = np.array([m1\_x, 0, m1\_z])  *# 构造向量*          PT = T\_center - P\_m1          PC = C - P\_m1  *# 几何计算*          cross = np.cross(PT, PC)          norm\_PT = np.linalg.norm(PT)          d = np.linalg.norm(cross) / (norm\_PT + 1e-7)  *# 添加极小量避免除零*  *# 投影计算*          proj = np.dot(PC, PT) / (norm\_PT \*\* 2 + 1e-7)  *# 遮蔽条件判断*          if d <= 10 and 0 <= proj <= 1:              shielding\_time += dt      return -shielding\_time  *# 注意返回负值*  *# 粒子群优化算法*  class PSO:      def \_\_init\_\_(self, objective\_function, bounds, num\_particles, max\_iter, dt):          self.objective\_function = objective\_function          self.bounds = bounds          self.num\_particles = num\_particles          self.max\_iter = max\_iter          self.dt = dt          self.particles = []          self.velocities = []          self.pbest\_positions = []          self.pbest\_scores = []          self.gbest\_position = None          self.gbest\_score = float('inf')          self.w = 0.8  *# 惯性权重*          self.c1 = 2.0  *# 个体学习因子*          self.c2 = 1.6  *# 群体学习因子*  *# 初始化粒子*          for \_ in range(num\_particles):              position = np.random.uniform(bounds[0], bounds[1], size=len(bounds[0]))              velocity = np.random.uniform(-0.01, 0.01, size=len(bounds[0]))              self.particles.append(position)              self.velocities.append(velocity)              score = objective\_function(position, dt)              self.pbest\_positions.append(position)              self.pbest\_scores.append(score)              if score < self.gbest\_score:                  self.gbest\_score = score                  self.gbest\_position = position      def optimize(self):          for iteration in range(self.max\_iter):  *# 动态调整惯性权重*              self.w = 0.9 - (0.9 - 0.4) \* (iteration / self.max\_iter)              for i in range(self.num\_particles):  *# 更新速度*                  r1, r2 = np.random.rand(), np.random.rand()                  self.velocities[i] = (self.w \* self.velocities[i] +                                        self.c1 \* r1 \* (self.pbest\_positions[i] - self.particles[i]) +                                        self.c2 \* r2 \* (self.gbest\_position - self.particles[i]))  *# 更新位置*                  self.particles[i] += self.velocities[i]  *# 限制位置在边界内*                  self.particles[i] = np.clip(self.particles[i], self.bounds[0], self.bounds[1])  *# 计算新位置的得分*                  score = self.objective\_function(self.particles[i], self.dt)  *# 更新个体最优解*                  if score < self.pbest\_scores[i]:                      self.pbest\_positions[i] = self.particles[i]                      self.pbest\_scores[i] = score  *# 更新全局最优解*                      if score < self.gbest\_score:                          self.gbest\_score = score                          self.gbest\_position = self.particles[i]              print(f"Iteration {iteration+1}/{self.max\_iter}, Best Score: {-self.gbest\_score}")          return self.gbest\_position, self.gbest\_score  *# 定义搜索空间的边界*  bounds = (np.array([0, 0, 0, 0, 0]), np.array([len(tq)-1,len(tp)-1, len(t)-1, len(v)-1, len(agl)-1]))  *# 初始化 PSO*  pso = PSO(objective\_function, bounds, num\_particles=400, max\_iter=150, dt=dt)  *# 执行优化*  best\_position, best\_score = pso.optimize()  *# 解码最佳位置*  best\_t\_val0 = tq[int(np.clip(best\_position[0], 0, len(tq) - 1))]  best\_t\_val1 = tp[int(np.clip(best\_position[1], 0, len(tp) - 1))]  best\_t\_val = t[int(np.clip(best\_position[2], 0, len(t) - 1))]  best\_v\_0 = v[int(np.clip(best\_position[3], 0, len(v) - 1))]  best\_a = agl[int(np.clip(best\_position[4], 0, len(agl) - 1))]  print(f"Best Parameters: t\_val0={best\_t\_val0},t\_val1={best\_t\_val1}, t\_val={best\_t\_val}, v\_0={best\_v\_0}, a={best\_a}")  print(f"Best Score: {best\_score}") |

|  |
| --- |
| **附录4** |
| 问题3 python代码 |
| import numpy as np  import random  *# ===== 参数表 =====*  x0, y0, z0 = 17800, 0, 1800  tq1 = np.linspace(0, 5, 50)  tp1 = np.linspace(5, 8, 100)  tq2 = np.linspace(6, 11, 50)  tp2 = np.linspace(11, 14, 100)  tq3 = np.linspace(12, 17, 50)  tp3 = np.linspace(17, 20, 100)  t1  = np.linspace(8, 28, 200)  t2  = np.linspace(14, 34, 200)  t3  = np.linspace(20, 40, 200)  v   = np.linspace(70, 140, 140)  agl = np.linspace(0, np.pi/2, 180)  T\_center = np.array([0, 200, 5])  M1\_init = np.array([20000, 0, 2000])  RADIUS = 10  dt = 0.1  T\_max = 80  *# 固定视线方向*  PT = T\_center - M1\_init  norm\_PT = np.linalg.norm(PT)  *# ===== 适应度函数 =====*  def fitness(params):      tq1\_val = tq1[params[0]]      tp1\_val = tp1[params[1]]      tq2\_val = tq2[params[3]]      tp2\_val = tp2[params[4]]      tq3\_val = tq3[params[6]]      tp3\_val = tp3[params[7]]      v0  = v[params[9]]      ang = agl[params[10]]      UAV\_v = np.array([-v0\*np.cos(ang), v0\*np.sin(ang), 0])      smokes = [(tq1\_val, tp1\_val), (tq2\_val, tp2\_val), (tq3\_val, tp3\_val)]      times = np.arange(0, T\_max, dt)      covered = np.zeros\_like(times, dtype=bool)      for tq\_val, tp\_val in smokes:          texplode = tq\_val + tp\_val          for idx, current\_t in enumerate(times):              if not (texplode <= current\_t <= texplode + 20):                  continue              launch\_pos = np.array([x0, y0, z0]) + UAV\_v \* tq\_val              explode\_pos = launch\_pos + UAV\_v \* tp\_val              explode\_pos[2] -= 0.5 \* 9.8 \* (tp\_val\*\*2)              pos\_current = explode\_pos.copy()              pos\_current[2] -= 3.0 \* (current\_t - texplode)              PC = pos\_current - M1\_init              d = np.linalg.norm(np.cross(PT, PC)) / (norm\_PT + 1e-9)              proj = np.dot(PC, PT) / (norm\_PT\*\*2 + 1e-9)              if d <= RADIUS and 0 <= proj <= 1:                  covered[idx] = True      return max(np.sum(covered) \* dt, 1e-6)  *# 防全零*  *# ===== GA 参数 =====*  pop\_size = 200  gens = 250  mutation\_rate = 0.45  mutation\_genes = 3  elite\_fraction = 0.1  *# 精英比例*  bounds = [      len(tq1)-1, len(tp1)-1, len(t1)-1,      len(tq2)-1, len(tp2)-1, len(t2)-1,      len(tq3)-1, len(tp3)-1, len(t3)-1,      len(v)-1,   len(agl)-1  ]  *# 初始化（50%中值附近，50%随机）*  pop = []  for i in range(pop\_size):      if i < pop\_size//2:          ind = [bounds[j]//2 + random.randint(-5,5) for j in range(len(bounds))]          ind = [min(max(vv,0), bounds[j]) for j,vv in enumerate(ind)]      else:          ind = [random.randint(0, ub) for ub in bounds]      pop.append(np.array(ind, dtype=int))  best = None  best\_fit = -1  *# ===== 遗传搜索 =====*  for g in range(gens):      fits = [fitness(ind) for ind in pop]  *# 更新全局最优*      for ind, fit in zip(pop, fits):          if fit > best\_fit:              best\_fit = fit              best = ind.copy()      if (g+1) % 20 == 0:          print(f"Gen {g+1}: Best shielding = {best\_fit:.4f}s")  *# 精英保留*      elite\_count = max(1, int(elite\_fraction \* pop\_size))      elite\_idx = np.argsort(fits)[-elite\_count:]      new\_pop = [pop[i].copy() for i in elite\_idx]  *# 锦标赛选择+交叉*      while len(new\_pop) < pop\_size:          i1, i2 = random.randint(0, pop\_size-1), random.randint(0, pop\_size-1)          p1 = pop[i1] if fits[i1] > fits[i2] else pop[i2]          i3, i4 = random.randint(0, pop\_size-1), random.randint(0, pop\_size-1)          p2 = pop[i3] if fits[i3] > fits[i4] else pop[i4]          c1, c2 = p1.copy(), p2.copy()          if random.random() < 0.85:  *# 高交叉率*              point = random.randint(1, len(bounds)-2)              c1[:point], c2[:point] = p2[:point].copy(), p1[:point].copy()          new\_pop.append(c1)          if len(new\_pop) < pop\_size:              new\_pop.append(c2)  *# 变异（适应度低的变动大，适应度高的微调）*      for i in range(elite\_count, pop\_size):          if random.random() < mutation\_rate:              changes = mutation\_genes if i > pop\_size//2 else 1              for \_ in range(changes):                  m\_idx = random.randint(0, len(bounds)-1)                  new\_pop[i][m\_idx] = random.randint(0, bounds[m\_idx])      pop = new\_pop  *# 最后再做局部爬山优化*  def hill\_climb(ind, steps=3):      global best\_fit      current = ind.copy()      cur\_fit = fitness(current)      for \_ in range(steps):          for j in range(len(bounds)):              new = current.copy()              new[j] = max(0, min(bounds[j], new[j] + random.choice([-1, 1])))              new\_fit = fitness(new)              if new\_fit > cur\_fit:                  current, cur\_fit = new, new\_fit      return current, cur\_fit  best, best\_fit = hill\_climb(best, steps=10)  *# ===== 输出结果 =====*  print("\n=== 最优方案 ===")  print(f"最大视线存在遮蔽时长（秒）：{best\_fit:.4f}")  print("tq1 =", tq1[best[0]], "tp1 =", tp1[best[1]])  print("tq2 =", tq2[best[3]], "tp2 =", tp2[best[4]])  print("tq3 =", tq3[best[6]], "tp3 =", tp3[best[7]])  print("速度 =", v[best[9]], "m/s")  print("角度 =", np.degrees(agl[best[10]]), "°") |

|  |
| --- |
| **附录5** |
| 介绍：该代码是某某语言编写的，作用是什么 |
|  |

|  |
| --- |
| **附录6** |
| 介绍：该代码是某某语言编写的，作用是什么 |
|  |

除了支撑材料的文件列表和源程序代码外，附录中还可以包括下面内容：

* 某一问题的详细证明或求解过程；
* 自己在网上找到的数据；
* 比较大的流程图；
* 较繁杂的图表或计算结果