

# 烟幕干扰弹多无人机协同投放策略的建模与优化

## 摘要

针对无人机投放烟幕干扰弹遮蔽真目标、抵御多枚空地导弹来袭的军事需求，本文构建统一的运动学模型与几何约束有效遮蔽判断模型，提出分场景的分层优化策略，解决不同规模下的烟幕投放策略优化问题。

首先，通过分析导弹、无人机与烟幕弹的运动特性，建立基于线段-球体相交的遮蔽判断模型，量化“导弹-真目标视线被烟幕云团（半径 10m，有效 20s）遮挡”的几何与时间约束。针对不同场景设计优化算法：1）单机单弹场景（问题 1-2），采用粒子群优化（PSO）算法求解无人机速度、航向、投弹/起爆时序，最大化遮蔽时长；2）单机多弹场景（问题 3），扩展时序约束（两弹间隔 $\geq 1s$ ），通过 PSO 实现多弹时空协同；3）多机单弹场景（问题 4），构建“时间矩阵预计算+贪心分配”框架，引入重叠惩罚项避免遮蔽冗余；4）多机多弹多导弹场景（问题 5），采用“任务分配-参数预优化-协同验证”分层架构，统筹 5 架无人机（每机最多 3 弹）对 3 枚导弹的干扰资源。

程序运行结果表明，模型可显著提升对真目标的有效遮蔽时长：固定参数场景（问题 1）遮蔽时长为 1.39s；单机单弹最优场景（问题 2）遮蔽时长达 4.59s，较固定参数提升 229%；单机三弹协同（问题 3）通过时空互补实现总遮蔽时长 6.5s；三机协同（问题 4）总遮蔽时长增至 11.6s；五机多弹对三枚导弹（问题 5）的总遮蔽时长达 20.26s，实现多目标全时段覆盖。

本文模型具有强通用性与可扩展性，可有效处理单机到多机、单目标到多目标的干扰需求，为战场烟幕干扰弹投放策略制定提供量化理论依据与决策支持。

**关键词：**粒子群优化；分层优化；贪心算法；时间矩阵

# 一、问题背景与重述

## （一）问题背景

烟幕干扰弹是一种通过化学燃烧或爆炸形成烟幕云团，用于遮蔽目标、干扰敌方导弹的低成本高效能防护手段。现代防御体系中，无人机凭借其长续航与高机动性，成为投放干扰弹的重要平台。

本题设定中，真目标为位置、形状已知的圆柱体，假目标位于坐标原点。3枚导弹 M1、M2、M3 以  $300\text{m/s}$  速度直指假目标，5架无人机 FY1 - FY5 位于不同初始位置。我们需要设定无人机的航向与速度（ $70 - 140\text{ m/s}$ ），每机投放两弹需间隔至少 1 秒。干扰弹起爆后形成以  $3\text{m/s}$  匀速下沉的球状云团，其中心  $10\text{m}$  范围内在 20 秒内提供有效遮蔽。

各物体的位置示意图如下所示：

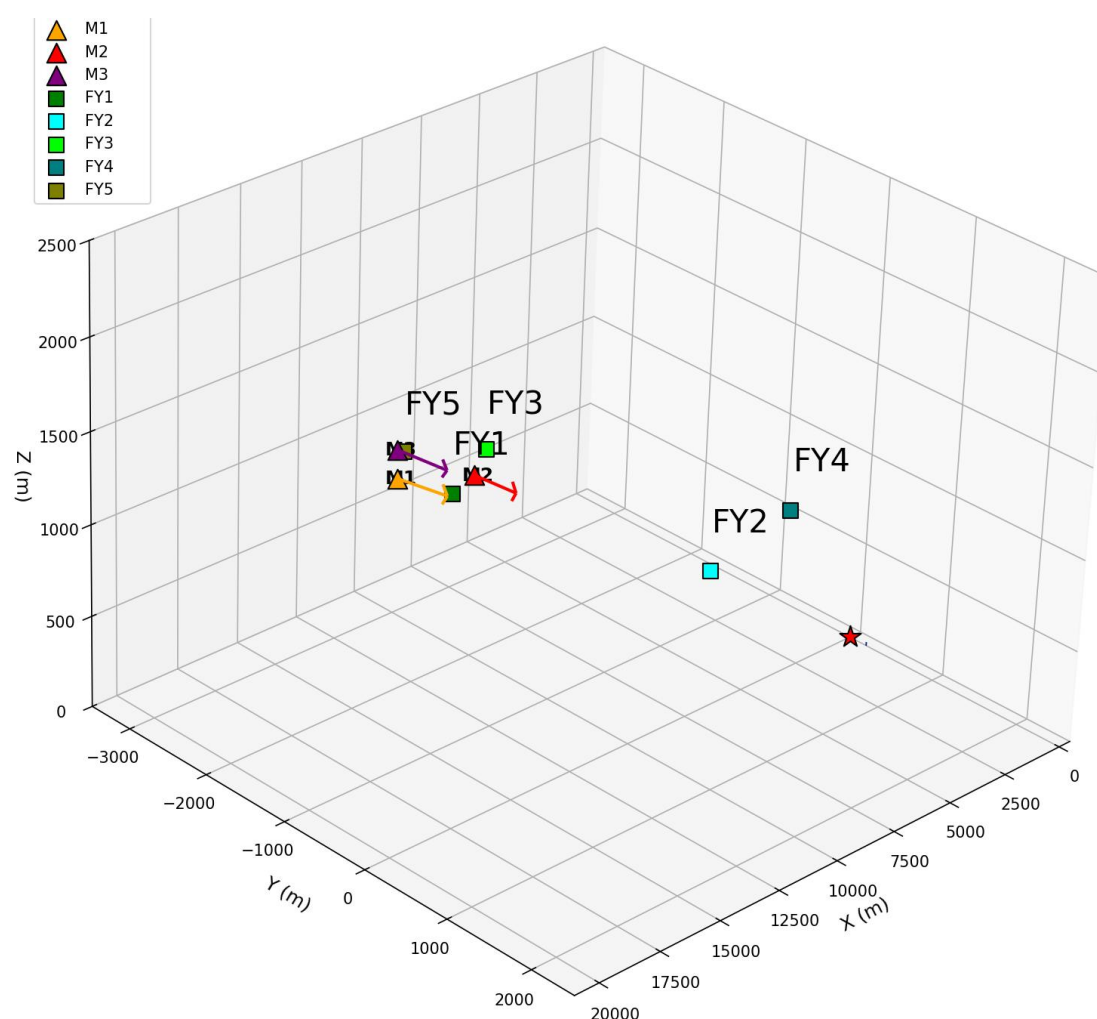


图 1 导弹、无人机与真假目标三维坐标示意图<sup>[1]</sup>

## （二）问题重述

需为以下五种情景设计无人机飞行方向、速度、干扰弹投放点与起爆点，以最大化真目标的有效遮蔽时间：

问题一：FY1 以  $120\text{ m/s}$  向假目标飞行，任务  $1.5\text{s}$  后投弹， $3.6\text{s}$  后起爆，

求对 M1 的有效遮蔽时间。

问题二：仅用 FY1 投 1 枚弹应对 M1，优化其飞行参数、投弹点与起爆点，以最大化遮蔽时间。

问题三：用 FY1 投 3 枚弹应对 M1，设计完整策略。

问题四：用 FY1 - FY3 各投 1 枚弹协同应对 M1，设计多机策略。

问题五：用 5 架无人机（每机最多 3 弹）同时应对 M1 - M3，统筹资源分配与时空协同。

## 二、问题分析

问题一为固定参数下的遮蔽时间计算，需建立导弹、无人机、干扰弹的运动学模型，并引入烟幕云形成、下沉与有效遮蔽的时空约束条件。

问题二属于四变量（速度、航向、投弹点、起爆点）优化问题，需以遮蔽时间为目标函数，综合考虑物理约束，适用智能算法求解。

问题三需协调三枚弹的投放与起爆时序，实现烟幕云团的时空互补，属多阶段决策问题，可结合时序优化或动态规划方法。

问题四扩展为多智能体协同优化，需在满足单机约束基础上，优化三机间的相对运动与弹幕配合，可采用分布式优化或联合决策框架。

问题五为全局资源调度问题，需同时处理目标分配、单机多弹策略与起爆控制，可采用分层优化结合群智能算法求解。

## 三、模型准备

### （一）模型假设

1. 导弹、无人机、烟幕弹、假目标均视为质点；
2. 烟幕弹脱离无人机后仅受重力（忽略空气阻力），做平抛运动；
3. 本文对有效遮蔽的定义：导弹与真目标表面任意一点的连线段均与烟幕云团球体相交，且时间在 20s 有效遮蔽期内。

### （二）符号说明

本文使用的符号、对应含义、对应单位和说明如下表。

表 1 符号说明表

符号	含义	单位	说明
无人机相关			
$D_i(0)$	无人机 $FY_i$ 的初始位置向量	m	$i=1, 2, \dots, 5$
$v_{d_i}$	无人机 $FY_i$ 的飞行速度	m/s	$70 \leq v_{d_i} \leq 140$
$\theta_i$	无人机 $FY_i$ 的飞行方向角（与 x 轴正方向夹角）	°	$0 \leq \theta_i \leq 360$
$\vec{e}_{d_i}$	无人机 $FY_i$ 飞行方向的单位向量	—	$(\cos \theta_i, \sin \theta_i, 0)$
导弹相关			
$R_k(0)$	导弹 $M_k$ 的初始位置向量	m	$k=1, 2, 3$
$\vec{v}_{m_k}$	导弹 $M_k$ 的速度向量	m/s	$\ \vec{v}_{m_k}\  = 300$
$\vec{e}_{m_k}$	导弹 $M_k$ 飞行方向的单位向量	—	指向假目标（原点）
$T_{total,k}$	导弹 $M_k$ 从初始位置到假目标的总时间	s	$T_{total,k} = \ R_k(0)\ /300$
烟幕弹相关			
$t_{drop,ij}$	无人机 $FY_i$ 投放第 $j$ 枚弹的时间	s	$t_{drop,ij} \geq 0$
$t_{delay,ij}$	第 $j$ 枚弹的起爆延迟时间	s	$t_{delay,ij} \geq 0$
$t_{burst,ij}$	第 $j$ 枚弹的起爆时间	s	$t_{burst,ij} = t_{drop,ij} + t_{delay,ij}$
$S_{drop,ij}$	第 $j$ 枚弹的投放点位置向量	m	由无人机位置和投放时间决定
$S_{burst,ij}$	第 $j$ 枚弹的起爆点位置向量	m	由投放点和延迟时间决定
$S_{smoke,ij}(t)$	第 $j$ 枚弹在时刻 $t$ 的烟幕中心位置	m	$t \in [t_{burst,ij}, t_{burst,ij} + 20]$
$v_s$	烟幕云团下沉速度	m/s	$v_s = 3$
$r_{smoke}$	烟幕有效遮蔽半径	m	$r_{smoke} = 10$
真目标相关			
$Q$	真目标表面采样点	m	圆柱表面点，用于遮蔽判断
center	真目标圆柱底面圆心	m	$(0, 200, 0)$
$r_{target}$	真目标圆柱半径	m	$r_{target} = 7$
$h_{target}$	真目标圆柱高度	m	$h_{target} = 10$
其他通用符号			
$g$	重力加速度	m/s <sup>2</sup>	$g = 9.8$
$I(t)$	遮蔽指示函数	—	$I(t) = 1$ 表示有效遮蔽, 否则为 0
$T_{cover}$	总有效遮蔽时间	s	目标函数，需最大化
$\lambda$	重叠时间惩罚系数	—	用于问题四、五的协同优化

注：其他符号在论文相应部分有所说明。

## 四、模型的建立与求解

### （一）统一的基础模型的建立

我们将适用于问题一到问题五的物理模型和有效遮蔽判断模型在下方给出：

#### 1. 坐标系与时间基准

##### （1）坐标系定义

以假目标为坐标原点  $(0, 0, 0)$ ，建立右手空间直角坐标系：

- ① 水平面为  $xy$  平面，水平向右为  $x$  轴正方向，水平向前为  $y$  轴正方向；
- ② 竖直向上为  $z$  轴正方向；
- ③ 真目标为底面圆心  $(0, 200, 0)$ 、半径 7m、高度 10m 的圆柱体。

##### （2）时间基准

定义无人机受领任务时刻为初始时刻  $t=0$ ，所有时间变量均基于此基准：

- ①  $t \geq 0$ ：无人机开始按规划轨迹飞行、导弹开始向假目标运动；
- ② 超过导弹击中假目标的时刻后（ $t > T_{total,k}$ ， $T_{total,k}$  为导弹  $M_k$  总飞行时间），干扰失去意义。

#### 2. 运动学模型的建立

##### （1）无人机运动模型

设无人机编号为  $FY_i (i = 1, 2, \dots, 5)$ ，对应 5 架无人机)，每架无人机仅执行等高度匀速直线飞行。记  $B_{ij}$  为第  $i$  架无人机投放的第  $j$  枚烟幕弹。

##### ① 无人机位置方程

飞行方向单位向量为：

$$\mathbf{e}_{d_i} = (\cos \theta_i, \sin \theta_i, 0) \quad (1)$$

任意时刻  $t \geq 0$ ， $FY_i$  的位置为：

$$\mathbf{D}_i(t) = \mathbf{D}_i(0) + v_{d_i} \cdot \mathbf{e}_{d_i} \cdot t \quad (2)$$

分量形式：

$$\begin{cases} x_{D_i}(t) = x_{i0} + v_{d_i} \cdot \cos \theta_i \cdot t \\ y_{D_i}(t) = y_{i0} + v_{d_i} \cdot \sin \theta_i \cdot t \\ z_{D_i}(t) = z_{i0} \end{cases} \quad (3)$$

##### ② 干扰弹投放点

$FY_i$  投放  $B_{ij}$  的时刻为  $t_{drop,ij}$ ，投放点  $S_{drop,ij}$ （即此时无人机位置）：

$$\mathbf{S}_{drop,ij} = \mathbf{D}_i(t_{drop,ij}) = \mathbf{D}_i(0) + v_{d_i} \cdot \mathbf{e}_{d_i} \cdot t_{drop,ij} \quad (4)$$

##### （2）烟幕干扰弹运动模型

干扰弹运动分为投放后-起爆前（平抛阶段）和起爆后-云团下沉（匀速下沉阶段）。

① 阶段 1: 投放后→起爆前（平抛运动， $t \in [t_{drop,ij}, t_{burst,ij}]$ ）

干扰弹脱离无人机后仅受重力，水平方向速度与无人机一致（无空气阻力），竖直方向做自由落体运动：

• 水平速度：

$$v_{bomb,水平} = v_{d_i} \cdot e_{d_i} \quad (5)$$

• 竖直加速度：

$$a_{bomb,竖直} = (0, 0, -g) \quad (6)$$

任意时刻  $t \in [t_{drop,ij}, t_{burst,ij}]$ ， $B_{ij}$  的位置为：

$$S_{bomb,ij}(t) = S_{drop,ij} + v_{bomb,水平} \cdot (t - t_{drop,ij}) + \frac{1}{2} a_{bomb,竖直} \cdot (t - t_{drop,ij})^2 \quad (7)$$

分量形式（代入  $\vec{e}_{d_i}$ ）：

$$\begin{cases} x_{bomb,ij}(t) = x_{drop,ij} + v_{d_i} \cdot \cos \theta_i \cdot (t - t_{drop,ij}) \\ y_{bomb,ij}(t) = y_{drop,ij} + v_{d_i} \cdot \sin \theta_i \cdot (t - t_{drop,ij}) \\ z_{bomb,ij}(t) = z_{drop,ij} - \frac{1}{2} g \cdot (t - t_{drop,ij})^2 \end{cases} \quad (8)$$

起爆点  $S_{burst,ij}$ ：当  $t = t_{burst,ij}$  时， $B_{ij}$  的位置（即云团初始中心）：

$$S_{burst,ij} = S_{bomb,ij}(t_{burst,ij}) = S_{drop,ij} + v_{d_i} \cdot e_{d_i} \cdot t_{delay,ij} - (0, 0, \frac{1}{2} g t_{delay,ij}^2) \quad (9)$$

② 阶段 2: 起爆后→云团下沉（匀速下沉， $t \in [t_{burst,ij}, t_{burst,ij} + 20]$ ）

起爆后瞬时形成球状云团，云团中心仅沿  $z$  轴负方向匀速下沉（无水平运动），有效遮蔽时长为起爆后 20s。任意时刻  $t \in [t_{burst,ij}, t_{burst,ij} + 20]$ ，云团中心

$S_{smoke,ij}(t)$  的位置为：

$$S_{smoke,ij}(t) = S_{burst,ij} - (0, 0, v_s \cdot (t - t_{burst,ij})) \quad (10)$$

分量形式：

$$\begin{cases} x_{B,ij}(t) = x_{burst,ij} \\ y_{B,ij}(t) = y_{burst,ij} \\ z_{B,ij}(t) = z_{burst,ij} - v_s \cdot (t - t_{burst,ij}) \end{cases} \quad (11)$$

(3) 导弹运动模型

设导弹编号为  $M_k$ （ $k = 1, 2, 3$ ，对应 3 枚导弹），所有导弹均直指假目标（原点）匀速飞行，速度大小  $v_{missile} = 300 \text{ m/s}$ 。

① 导弹飞行方向与速度向量

导弹飞行方向为“从初始位置指向原点”，方向单位向量为：

$$\mathbf{e}_{\text{missile},k} = -\frac{\mathbf{R}_k(0)}{\|\mathbf{R}_k(0)\|} \quad (12)$$

（负号表示与 $\mathbf{R}_k(0)$ 反向，即指向原点）

导弹速度向量为：

$$\mathbf{v}_{\text{missile},k} = v_{\text{missile}} \cdot \mathbf{e}_{\text{missile},k} \quad (13)$$

## ② 导弹位置方程

任意时刻 $t \geq 0$ ， $M_k$ 的位置为：

$$\mathbf{R}_k(t) = \mathbf{R}_k(0) + \mathbf{v}_{\text{missile},k} \cdot t \quad (14)$$

分量形式（代入 $\mathbf{e}_{\text{missile},k}$ ）：

$$\begin{cases} x_{R_k}(t) = x_{k0} - v_{\text{missile}} \cdot \frac{x_{k0}}{\|\mathbf{R}_k(0)\|} \cdot t \\ y_{R_k}(t) = y_{k0} - v_{\text{missile}} \cdot \frac{y_{k0}}{\|\mathbf{R}_k(0)\|} \cdot t \\ z_{R_k}(t) = z_{k0} - v_{\text{missile}} \cdot \frac{z_{k0}}{\|\mathbf{R}_k(0)\|} \cdot t \end{cases} \quad (15)$$

其中， $\|\mathbf{R}_k(0)\|$ 为 $M_k$ 初始位置到原点的距离：

$$\|\mathbf{R}_k(0)\| = \sqrt{x_{k0}^2 + y_{k0}^2 + z_{k0}^2} \quad (16)$$

## 3. 统一的有效遮蔽判断模型的建立

有效遮蔽需同时满足时间条件和几何条件，适用于所有问题中“ $S_{ij}$ 对导弹 $M_k$ 的遮蔽”判断。

### (1) 时间条件

仅当判断时刻 $t$ 落在“云团有效时段”内，才可能形成遮蔽：

$t \in [t_{\text{burst},ij}, t_{\text{burst},ij} + 20]$ 且 $t < T_{\text{total},k}$ （ $T_{\text{total},k}$ 后导弹已击中假目标，遮蔽无意义）

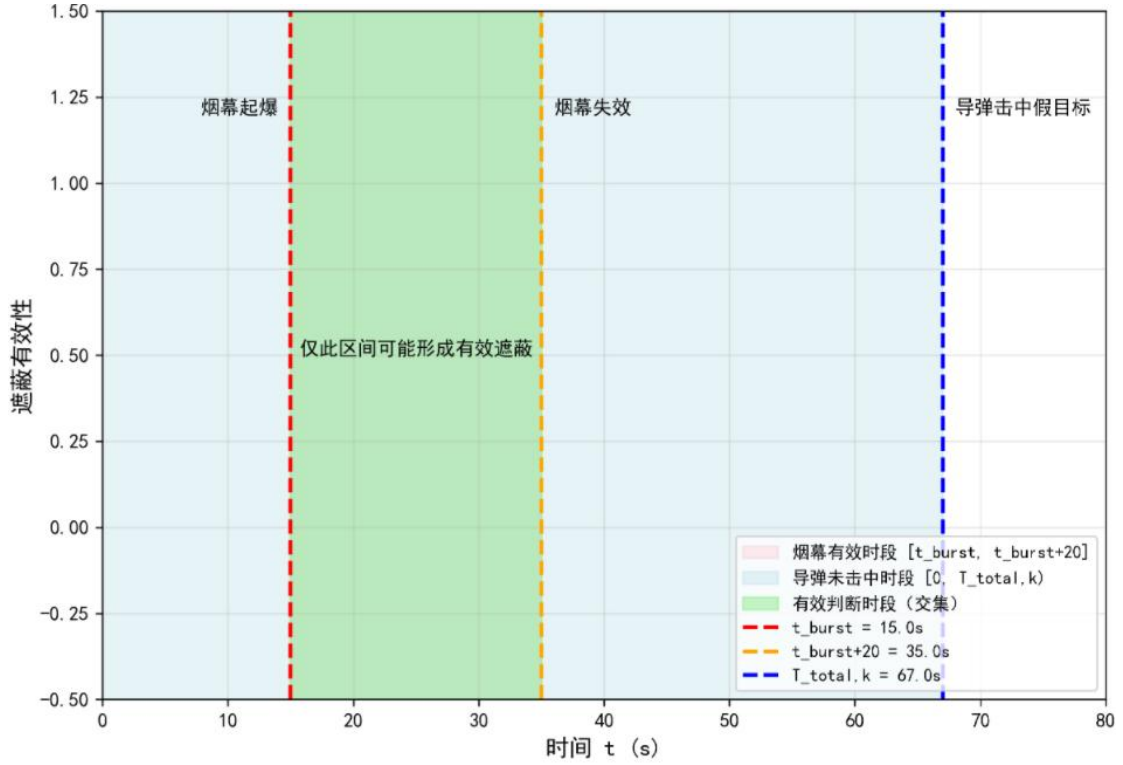


图 2 有效遮蔽时间条件示意图<sup>[2]</sup>

## (2) 几何条件

几何条件定义：导弹 $M_k$ 在时刻 $t$ 的位置 $R_k(t)$ ，到真目标表面任意一点 $Q$ 的连线线段 $R_k(t)Q$ ，与烟幕云团（中心 $S_{smoke,ij}(t)$ ，半径 $r_{smoke} = 10m$ ）相交。

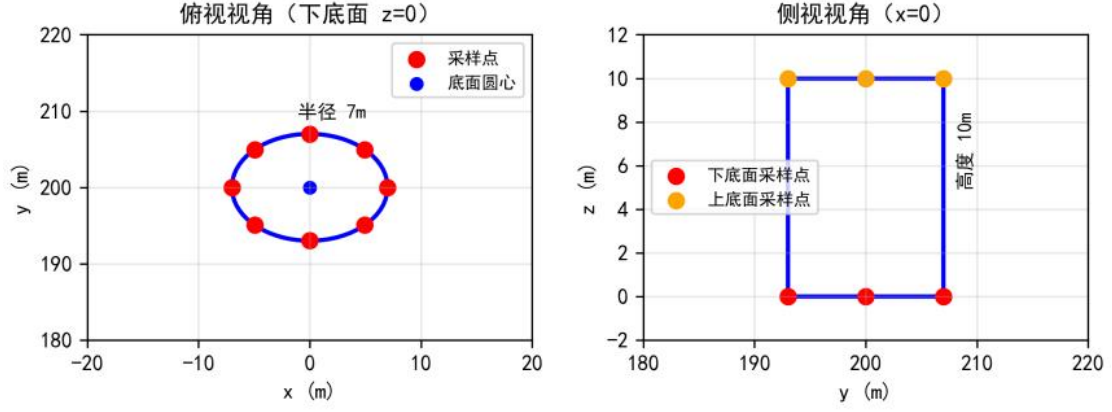
### ① 真目标采样点（统一表征圆柱体）

真目标为圆柱体，需通过采样点 $Q$ 覆盖其表面，从而避免简化为点导致的误差：

- 1) 底面： $z = 0$ （下底面）和 $z = 10$ （上底面），圆心 $(0,200,0)$ ，半径 $7m$ ；
- 2) 采样方式：极坐标采样，任意采样点 $Q$ 的坐标为：

$$Q = (0 + 7 \cos \alpha, 200 + 7 \sin \alpha, z_q) \quad (17)$$

其中， $\alpha \in [0, 2\pi)$ （绕圆柱中心的角度）， $z_q \in \{0, 10\}$ （上下底面），采样数量可设为 $n$ （如 $n = 64$ ，确保覆盖均匀）。



真目标：底面圆心(0, 200, 0)、半径7m、高10m  
采样方式：极坐标覆盖上下底面，避免简化为点的误差

图 3 真目标采样示意图

## ② 线段与球体相交判断（向量法）【5】

设：

- 线段端点：  $A = R_k(t)$ （导弹位置），  $B = Q$ （真目标采样点）；
- 球体： 中心  $C = S_{smoke,ij}(t)$ （云团中心）， 半径  $r = r_{smoke} = 10 \text{ m}$ 。

判断步骤：

- 1) 计算向量：  $\mathbf{AB} = B - A$ ,  $\mathbf{AC} = C - A$ ;
- 2) 计算  $\mathbf{AC}$  在  $\mathbf{AB}$  上的投影参数  $t_{proj}$ （判断交点是否在线段上）：

$$t_{proj} = \frac{\mathbf{AC} \cdot \mathbf{AB}}{\|\mathbf{AB}\|^2} \quad (18)$$

- 若  $t_{proj} < 0$ ： 交点在 A 外侧， 线段 AB 不经过球体；
- 若  $t_{proj} > 1$ ： 交点在 B 外侧， 线段 AB 不经过球体；
- 若  $0 \leq t_{proj} \leq 1$ ： 交点为线段上的点  $P = A + t_{proj} \cdot \mathbf{AB}$ ， 继续判断距离。

离。

- 3) 计算球心 C 到线段 AB 的距离  $d$ ：

$$d = \frac{\|\mathbf{AC} \times \mathbf{AB}\|}{\|\mathbf{AB}\|} \quad (19)$$

- 4) 若  $d \leq r$ ： 线段 AB 与球体相交， 满足几何条件； 否则不满足。

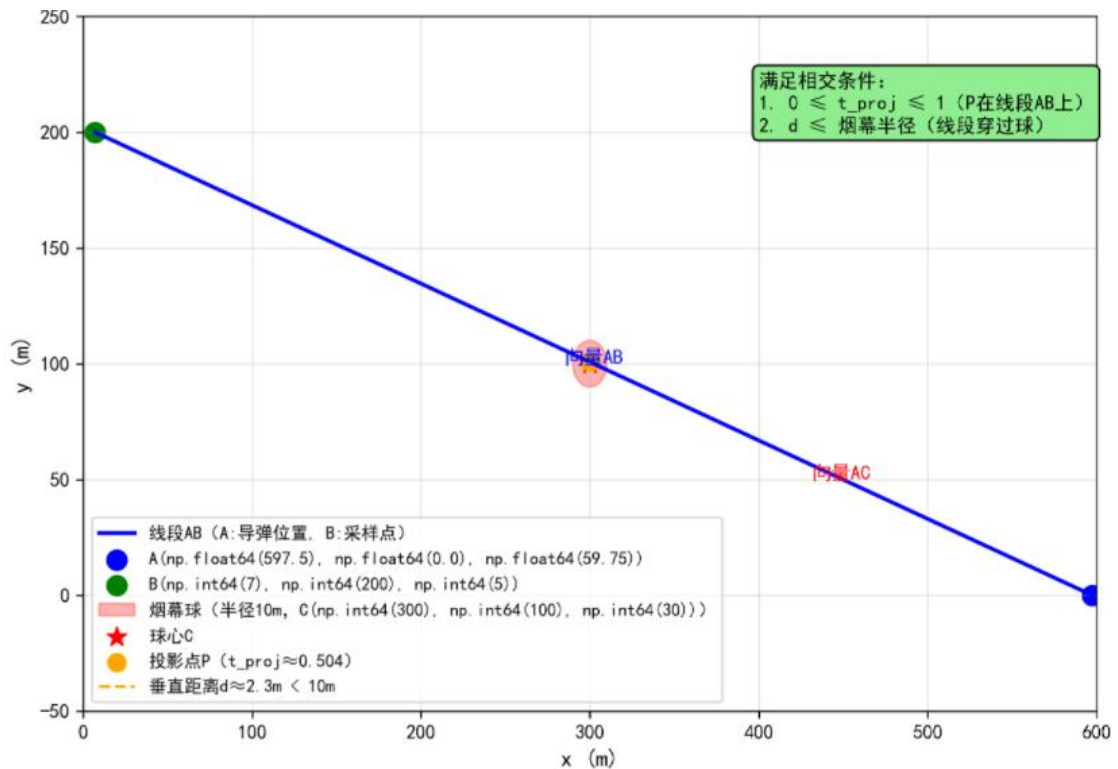


图 4 导弹-采样点线段与烟幕球相交判断示例图<sup>【3】</sup>

### (3) 最终有效遮蔽判定

对任意时刻  $t$ ，若同时满足：

- 时间条件： $t \in [t_{burst,ij}, t_{burst,ij} + 20] \cap [0, T_{total,k})$ ；
- 几何条件：任意取真目标采样点  $Q$ ，都满足线段  $R_k(t)Q$  与云团球体相交。

则判定“ $S_{ij}$ 在时刻 $t$ 对 $M_k$ 形成有效遮蔽”，遮蔽时长为所有满足条件的 $t$ 的总区间长度。

## (二) 问题一：单架无人机单枚干扰弹对 $M_1$ 的遮蔽时长计算

### 1. 模型的建立

#### (1) 确定决策变量

表 2 问题一决策变量

决策变量	物理意义
干扰弹投放时刻	无人机 FY1 的第 1 枚烟幕弹的投放时间
干扰弹投放位置坐标	无人机 FY1 的第 1 枚烟幕弹在 $t_{drop,11}$ 时刻的烟幕中心位置
干扰弹起爆延迟时间	无人机 FY1 的第 1 枚烟幕弹的起爆时间与投放时间之差

#### (2) 进行遮蔽判断

我们通过“真目标采样+向量距离判断”实现判断逻辑，分两步：

① 真目标采样点生成 (Points 函数)

通过采样圆柱表面点保证判断精度，Points(n) 函数逻辑如下（n 为每个底面的采样点数）：

- 取圆柱上下两个底面 ( $z=0$  和  $z=10$ )；
- 每个底面按极坐标采样，采样点坐标由公式 (17) 得到；
- 最终生成  $2n$  个采样点，覆盖圆柱表面关键位置。

由代码 `points = zeros(n*2, 3)` 存储采样点，循环生成上下底面的 ( $x, y, z$ )，确保真目标的空间形态被准确表征。

② 遮蔽判断逻辑 (occlusion 函数)

在烟幕干扰弹的有效遮蔽时段内，为精准判定每一刻是否形成有效遮蔽，采用时间步长  $\Delta t = 0.01 \text{ s}$  对该时段进行遍历（时间步索引  $k = 0, 1, 2, \dots, 2000$ ，覆盖完整有效时段）。对于每个时间步对应的时刻  $t = t_{burst,11} + k \cdot \Delta t$ ，先计算烟幕云团中心、导弹 M1 及真目标采样点的实时位置，再通过三个向量条件完成遮蔽判断。

1) 各物体实时位置计算

a. 烟幕云团中心位置  $S_{smoke,11}(t)$

对任意时刻  $t \in [t_{burst,11}, t_{burst,11} + 20]$ ，将  $S_{burst,11}$ 、 $v_s$ 、 $t - t_{burst,11}$  代入公式 (10)，得到云团中心位置  $S_{smoke,11}(t)$ ：

$$S_{smoke,11}(t) = (17188, 0, 1736.5 - 3 \times (t - 5.1)) \text{ (m)} \quad (20)$$

b. 导弹 M1 实时位置  $R_1(t)$

将  $R_1(0)$ 、 $v_{missile,1}$ 、 $t$  代入公式 (14)，得到任意时刻 M1 位置：

$$R_1(t) = (20000 - 298.5t, 0, 2000 - 29.85t) \text{ (m)} \quad (21)$$

c. 真目标采样点  $P$  坐标

见公式 (17)。

2) 向量定义与遮蔽判断条件

量化判断线段  $R_1(t)P$  与烟幕云团是否相交，定义核心向量如下：

•  $AB = P - R_1(t)$ ：导弹实时位置  $R_1(t)$  指向采样点  $P$  的向量（即线段  $R_1(t)P$  的方向向量）；

•  $AC = S_{11}(t) - R_1(t)$ ：导弹实时位置  $R_1(t)$  指向烟幕云团中心  $S_{smoke,11}(t)$  的向量；

•  $BC = S_{11}(t) - P$ ：采样点  $P$  指向烟幕云团中心  $S_{smoke,11}(t)$  的向量。

基于上述向量，通过以下三个条件判定线段与烟幕云团是否无交点（即无遮蔽），若三个条件均不满足，则判定为有效遮蔽。

表 3 判断是否屏蔽的向量条件

条件符号	代码变量	物理意义	公式（基于向量）
cond1	$d1 > 10$	烟幕云团中心到线段 $R_1(t)P$ 的距离 $> 10$ m，线段与烟幕无交点	公式（19）
cond2	$(d2 > 10) \& (\dot{a}_{abc} < 0)$	烟幕中心到导弹距离 $> 10$ m，且采样点在导弹远离烟幕一侧	$d2 = \ AC\ ;$ $\dot{a}_{abc} = AB \cdot AC$
cond3	$(d3 > 10) \& (\dot{b}_{abc} < 0)$	采样点到烟幕中心距离 $> 10$ m，且导弹在采样点远离烟幕一侧	$d3 = \ BC\ ;$ $\dot{b}_{abc} = BA \cdot BC$

### ③ 遮蔽判定规则与时长统计

#### 1) 有效遮蔽判定

对任意时间步 $t$ ，需同时满足以下两个前提，再结合向量条件判定：

- 时间前提： $t \in [t_{burst,11}, t_{burst,11} + 20]$  且  $t < 67.0$  s（导弹击中假目标前）；
  - 向量条件前提：任意一个真目标采样点 $P$ ，使得 cond1、cond2、cond3 均不满足（即线段 $R_1(t)P$ 与烟幕云团相交）。
- 若上述条件均满足，则判定时刻 $t$ 存在有效遮蔽。

#### 2) 遮蔽时长统计

- 初始化 $t_{start} = -1$ （首次遮蔽时刻，初始为无效值）、 $t_{end} = -1$ （首次非遮蔽时刻，初始为无效值）；
- 遍历所有时间步：
 

若首次判定为遮蔽( $t_{start} = -1$  且满足遮蔽条件)，更新 $t_{start} = t$ ；

若已记录 $t_{start}$ 但首次判定为非遮蔽（不满足遮蔽条件），更新 $t_{end} = t$ ，并终止遍历（后续时段无遮蔽或遮蔽不连续，仅统计首次连续遮蔽时长）；
- 时长计算：
 

若 $t_{start} \neq -1$  且  $t_{end} \neq -1$ ，遮蔽时长 $= t_{end} - t_{start}$ ；

若遍历结束仍为遮蔽（ $t_{end} = -1$ ），遮蔽时长 $= 20 - t_{start}$ ；

若始终无遮蔽( $t_{start} = -1$ )，遮蔽时长 $= 0$ 。

### (3) 算法说明

#### Algorithm 1 单枚干扰弹对 M1 的遮蔽时长计算

**Input:** 固定参数（无人机位置、速度、方向、投放时间、延迟时间）

**Output:** 遮蔽时长

爆炸点  $\leftarrow$  根据运动模型计算

---

导弹位置  $\leftarrow$  根据导弹运动模型计算

总遮蔽时间  $\leftarrow 0$

For 时间从爆炸时刻到爆炸后 20 秒 do

    计算烟幕当前位置（考虑下沉）

    计算导弹当前位置

    If 所有采样点都被遮蔽 then

        总遮蔽时间  $\leftarrow$  总遮蔽时间 + 时间步长

    End If

End For

Return 总遮蔽时间

End

---

## 2. 计算结果

运行 question1.m 代码，最终输出  $t_{\text{ans}} = 1.39$ ，即烟幕干扰弹对 M1 的有效遮蔽时长为 1.39s。

### （三）问题二：单架无人机单枚干扰弹对 M1 的最优投放策略设计

#### 1. 模型的建立

##### （1）确定决策变量

表 4 问题二决策变量

决策变量	约束条件
无人机飞行速度	$70 \leq v \leq 140 m/s$
无人机飞行方向	$0^\circ \leq \theta \leq 360^\circ$
干扰弹投放时间	$t_{drop} \geq 0$ ，且需满足 $t_{drop} + t_{delay} < T_{total}$ （避免导弹已击中目标后起爆）
干扰弹起爆延迟	$t_{delay} \geq 0$ ，且需满足起爆点 $z$ 坐标 $z_{boom} > 0$ （避免云团落地失效）

##### （2）有效遮蔽判断

###### ① 时间条件

$$t \in [t_{burst}, t_{burst} + 20] \cap [0, 67.0);$$

###### ② 几何条件

对任意真目标采样点  $Q$ ，满足上文 3. (2) 所述几何条件时，判定为有效遮蔽，遮蔽时长为满足条件的  $t$  的总区间长度。

##### （3）构建目标函数

以有效遮蔽总时长 $T_{cover}$ 为目标函数，即：

$$\max T_{cover} = \int_{t_{boom}}^{t_{boom}+20} I(t)dt$$

式中， $I(t)$ 为指示函数， $I(t) = 1$  表示时刻  $t$  满足有效遮蔽条件， $I(t) = 0$  表示不满足。

(1) 使用粒子群优化（PSO）算法求解<sup>[4]</sup>

(2) 算法参数

表 5 问题二粒子群优化算法参数表

参数	符号	取值	说明
粒子数量	$N$	500	平衡搜索效率与精度，数量过少易陷入局部最优，过多增加计算量
最大迭代次数	$K$	100	确保算法收敛，避免无意义迭代
惯性权重	$w$	0.9→0.4	线性递减，初期大权重增强全局搜索，后期小权重增强局部收敛
学习因子	$c1, c2$	2, 2	分别控制粒子向自身最优、群体最优学习的程度，取值为 2 时搜索性能较优
位置约束	$lb, ub$	[70, 0, 0, 0], [140, 360, 67, 19.17]	决策变量的上下边界，确保解的物理有效性

(3) 算法说明

Algorithm 2 基于 PSO 的单枚干扰弹最优投放策略搜索

Input: 参数范围约束

Output: 最优参数组合，最大遮蔽时间

While 迭代次数未达上限 do

For each particle do

fit ← 计算遮蔽时间(粒子参数)

If fit > 当前最优 then

更新最优解

End If

End For

更新所有粒子

End While

Return 最优参数和最大遮蔽时间

End

2. 计算结果

无人机速度：83.75 m/s；无人机方向（与 x 轴正方向的夹角）： 4.31 °

投放时间： 1.54 s；延迟爆炸时间： 0.00 s

干扰弹投放点：（17928.58, 9.69, 1800.00）m

干扰弹引爆点：（17928.58, 9.69, 1800.00）m

最长遮盖时间： 4.59 s

#### （四）问题三：单架无人机三枚干扰弹对 M1 的投放策略规划

##### 1. 模型的建立

###### （1）确定决策变量

根据题目，我们需优化 8 个连续决策变量，涵盖无人机飞行参数与 3 枚干扰弹的投放/起爆参数：

表 6 问题三决策变量

决策变量	约束条件
无人机 FY1 飞行速度	$70 \leq v_{d1} \leq 140 \text{ (m/s)}$
无人机 FY1 飞行方向角	$0 \leq \theta_1 \leq 360 \text{ (}^\circ\text{)}$
第 1 枚弹投放时间	$t_{drop,11} \geq 0$ ，且 $t_{drop,11} + t_{delay,11} \leq T_{total,1}$
第 2 枚弹投放时间	$t_{drop,12} \geq$ $t_{drop,11} + 1 \text{ (s)}$ ， 且 $t_{drop,12} + t_{delay,12} \leq T_{total,1}$
第 3 枚弹投放时间	$t_{drop,13} \geq t_{drop,12} + 1 \text{ (s)}$ ，且 $t_{drop,13} + t_{delay,13}$ $\leq T_{total,1}$
第 1 枚弹起爆延迟	$t_{delay,11} \geq 0$ ，且起爆点 z 坐标 $> 0$ (即 $t_{delay,11} \leq 19.17$ )
第 2 枚弹起爆延迟	$t_{delay,12} \geq 0$ ，且起爆点 z 坐标 $> 0$ (即 $t_{delay,12} \leq 19.17$ )
第 3 枚弹起爆延迟	$t_{delay,13} \geq 0$ ，且起爆点 z 坐标 $> 0$ (即 $t_{delay,13} \leq 19.17$ )

###### （2）多弹协同有效遮蔽判断

① 时间条件：  $t \in [t_{burst}, t_{burst} + 20] \cap [0, 67.0]$ ；

② 几何条件：对真目标采样点  $Q$ ，计算导弹位置  $R_1(t)$  到  $Q$  的线段  $R_1(t)Q$ ：

1) 设  $A = R_1(t)$ 、 $B = Q$ 、 $C = B(t)$ ，计算向量  $\mathbf{AB} = B - A$ 、 $\mathbf{AC} = C - A$ ；

2) 将  $\mathbf{AC}$ 、 $\mathbf{AB}$  代入公式(18)，得投影参数  $t_{proj} = \frac{\mathbf{AC} \cdot \mathbf{AB}}{\|\mathbf{AB}\|^2}$ ，判断  $0 \leq t_{proj} \leq 1$   
(交点在线段上)；

3) 将  $\mathbf{AC}$ 、 $\mathbf{AB}$  代入公式(19)，得球心到线段的距离  $d = \frac{\|\mathbf{AC} \times \mathbf{AB}\|}{\|\mathbf{AB}\|}$ ，若  $d \leq$

10m，则满足几何条件。

满足上述两条件时，判定为有效遮蔽，遮蔽时长为满足条件的 $t$ 的总区间长度。

### (3) 目标函数构建

以有效遮蔽总时长 $T_{cover}$ 为目标函数，即：

$$\max T_{cover} = \int_{t_{boom}}^{t_{boom}+20} I(t)dt$$

式中， $I(t)$ 为指示函数， $I(t) = 1$  表示时刻  $t$  满足有效遮蔽条件， $I(t) = 0$

表示不满足。

## 2. 基于粒子群优化（PSO）的求解<sup>[4]</sup>

由于目标函数 $T_{cover}$ 与决策变量呈非线性耦合关系，无法通过解析法求解，因此我们采用粒子群优化算法，通过模拟群体粒子的迭代搜索实现最优解查找。

### (1) 算法参数设置

表 7 问题三粒子群优化算法参数表

参数	取值	说明
粒子数量	50	确保覆盖 8 维决策空间的可行域
最大迭代次数	500	增加迭代次数以避免陷入局部最优，平衡收敛性与计算效率
惯性权重 $w$	0.9→0.4	线性递减，初期大权重增强全局搜索，后期小权重增强局部收敛
学习因子 ( $c_1, c_2$ )	2, 2	分别控制粒子向 “自身最优” “群体最优” 学习的强度，保持搜索稳定性
位置约束 [ $lb, ub$ ]	[70, 0, 0, 0, 0, 0, 0, 0], [140, 360, 67, 67, 67, 19.17, 19.17, 19.17]	对应 8 个决策变量的上下边界，确保物理有效性

### (2) 约束处理机制

在计算粒子适应度（即 $T_{cover}$ ）时，若违反以下约束，直接返回适应度 0，强制粒子搜索可行域：

- 投弹间隔约束：  $t_{drop,12} < t_{drop,11} + 1$  或  $t_{drop,13} < t_{drop,12} + 1$ ;
- 起爆延迟约束：  $t_{delay,1j} > 19.17$  (s) ( $j = 1,2,3$ );
- 起爆时刻约束：  $t_{burst,1j} \geq 67.0$  (s) ( $j = 1,2,3$ );

- 起爆点高度约束:  $z_{burst,1j} \leq 0 (j = 1,2,3)$ 。

### (3) 算法说明

---

#### Algorithm 3 单架无人机三枚干扰弹对 M1 的投放策略

---

Input: 参数范围约束

Output: 最优参数组合, 最大遮蔽时间

```

While 迭代次数未达上限 do
  For each particle do
    If 投弹间隔 < 1s OR 起爆时间 > 67s then
      fit ← 0
    Else
      fit ← 计算三弹遮蔽时间(粒子参数)
    End If
    If fit > 当前最优 then
      更新最优解
    End If
  End For
  更新所有粒子
End While

Return 最优参数和最大遮蔽时间
End

```

---

### 3. 计算结果

运行程序得到最大有效遮蔽时长 6.5s, 其余参数填入 result1.xlsx 作为支撑材料上交。

#### (五) 问题四: 三架无人机单枚干扰弹对 M1 的协同投放

##### 1. 模型的建立

###### (1) 确定决策变量

三架无人机(记为  $FY_i, i = 1,2,3$ )各对应 4 个决策变量, 共 12 个连续决策变量, 需满足物理约束与协同需求。

表 8 问题四决策变量

决策变量	约束条件
无人机 $FY_i$ 飞行速度	$70 \leq v_{d_i} \leq 140 \text{ m/s} (i = 1, 2, 3)$
无人机 $FY_i$ 飞行方向角	$0^\circ \leq \theta_i \leq 360^\circ (i = 1, 2, 3)$
$FY_i$ 投放弹的时刻	$t_{drop,i1} \geq 0$ , 且 $t_{drop,i1} + t_{delay,i1} < T_{total,1}$ ( $T_{total,1} \approx 67.0 \text{ s}$ , 导弹击中假目标前起爆)
$FY_i$ 弹的起爆延迟	$t_{delay,i1} \geq 0$ , 且起爆点 $z$ 坐标 $> 0$ (即 $t_{delay,i1} \leq \sqrt{\frac{2z_{i0}}{g}}$ , $z_{i0}$ 为 $FY_i$ 初始高度)

## (2) 三机协同有效遮蔽判断

协同遮蔽需满足“时间条件+几何条件”的“或逻辑”（任意一架无人机的烟幕弹满足即可），且总时长需计算遮蔽区间的并集（避免重叠时段重复计数）。

### ① 真目标采样点生成

沿用圆柱体采样方法，对真目标进行极坐标采样：径向采样数 $n_r = 8$ （绕圆柱中心的角度 $\alpha \in [0, 2\pi)$ ），高度采样数 $n_h = 11$ （ $z \in [0, 10] \text{ m}$ ），共生成 $8 \times 11 = 88$ 个采样点，覆盖圆柱表面。

### ② 单弹遮蔽判断（时间+几何条件）

对 $FY_i$ 的弹，任意时刻 $t$ 的遮蔽判断与问题三相同。

### ③ 协同遮蔽与总时长计算

- 1) 协同逻辑：时刻 $t$ 判定为“有效遮蔽”，当且仅当存在至少一个 $i \in \{1, 2, 3\}$ ，使得 $FY_i$ 的弹满足“时间条件+几何条件”；
- 2) 总时长计算：采用“区间并集”算法，步骤如下：提取三枚弹的有效遮蔽区间 $I_i = [t_{start,i}, t_{end,i}]$ （ $t_{start,i}$ 为首次满足条件时刻， $t_{end,i}$ 为首次不满足条件时刻）；对区间按起始时间排序，合并重叠或相邻区间；总有效遮蔽时长 $T_{cover} = \sum$ （合并后区间的结束时间－起始时间）。

## (3) 目标函数构建

以“三架无人机协同的总有效遮蔽时长最大化”为目标，即：

$$\max T_{cover} = \int_0^{67.0} I(t) dt$$

其中，指示函数 $I(t) = 1$ （时刻 $t$ 满足协同遮蔽条件）， $I(t) = 0$ （否则）。

## 2. 带重叠惩罚的粒子群优化（PSO）算法求解

由于12个决策变量存在强耦合，且需避免遮蔽时段过度重叠，我们采用带重叠惩罚的PSO算法求解，核心是在适应度函数中引入重叠惩罚项，引导区间错开。

(1) 算法参数设置

表 9 问题四粒子群优化算法参数表

参数	取值	说明
粒子数量	300	确保覆盖 12 维决策空间的可行域
最大迭代次数	300	增加迭代次数以避免陷入局部最优，平衡收敛性与计算效率
惯性权重 $w$	0.9→0.4	线性递减，初期全局搜索，后期局部收敛
学习因子 ( $c_1, c_2$ )	2, 2	控制粒子向自身最优、群体最优学习的强度
位置约束 [ $lb, ub$ ]	[70, 0, 0, 0, 0, 0, 0, 0], [140, 360, 67, 67, 67, 19.17, 19.17, 19.17]	按每架无人机的约束设置上下界
重叠惩罚系数	$\lambda$	惩罚重叠时段，平衡单弹时长与协同互补

(2) 适应度函数（带惩罚）

对粒子的决策变量组合，适应度计算如下：

$$fit = T_{cover} - \lambda \cdot Overlap$$

其中， $Overlap$ 为三枚弹遮蔽区间的总重叠时长（重叠部分仅计一次，多余部分扣除）；若违反约束（如起爆点落地、起爆时刻超 67s，则 $fit = -1000$ （强制粒子搜索可行域）。

(3) 算法说明

Algorithm 4 三架无人机各投一枚弹对 M1 的协同投放
Input: 三架无人机位置
Output: 各无人机最优参数, 总遮蔽时间
For each drone do
While 迭代次数未达上限 do
For each particle do
$fit \leftarrow$ 计算遮蔽时间(粒子参数) - $\lambda \times$ 与已优化无人机遮蔽时间重叠部分
If $fit >$ 当前最优 then
更新最优解
End If
End For
更新所有粒子
End While

更新总遮蔽时间段
End For
总遮蔽时间 $\leftarrow$ 计算时间段并集长度
Return 各无人机参数和总遮蔽时间
End

3. 计算结果

运行程序得到最大有效遮蔽时长 11.6s, 其余参数填入 result2.xlsx 作为支撑材料上交。

（六）问题五：五架无人机多枚干扰弹对三枚导弹的全面干扰

1. 模型的建立

(1) 确定决策变量

表 10 问题五决策变量

决策变量	意义	约束条件
遮蔽时间矩阵	$FY(i)$ 对 $M(j)$ 的最大单机电遮蔽时长	$time\_matrix(i,j) \geq 0$ (遮蔽时长非负)
任务分配变量	二进制变量, $x_{ik}$ = 1 表示 $FY(i)$ 分配给 $M(k)$ , $x_{ik}$ = 0 反之	$\sum_{k=1}^3 x_{ik} = 1$ (每架无人机仅服务 1 枚导) ); $\sum_{i=1}^5 x_{ik} \geq 1$ (每枚导弹至少 1 架 无人机服务)
无人机飞行方向角	$FY_i$ 飞行方向与 x 轴正方向的 逆时针夹角	$0^{\circ} \leq \theta_i \leq 360^{\circ} (i = 1,2,3,4,5)$
无人机飞行速度	$FY(i)$ ( $i$ = 1,2,3,4,5)等高度匀速飞行速	$70 \leq v_{d_i} \leq 140 \text{ m/s}$
投弹时刻	$FY(i)$ 投放第 j 枚弹( $j$ = 1,2,3,4,5)的时刻	$t_{drop,i1} \geq 0$ ; $t_{drop,i(j+1)} \geq t_{drop,ij} + 1 \text{ s}$ (投弹间隔约束)
起爆延迟	$FY(i)$ 第 j 枚弹的起爆延迟(投 放至起爆间隔)	$t_{delay,ij} \geq 0$ ; 起爆点 z 坐标> 0 (即 $t_{delay,ij} \leq \sqrt{\frac{2z_{i0}}{g}}$ , $z_{i0}$ 为 $FY(i)$ 初始高度)

(2) 多机-多弹-多导弹协同遮蔽判断

按“导弹分组”判断遮蔽效果：分配给 $M(k)$ 的所有无人机的弹，仅需干扰

$M(k)$ 对真目标的视线；总遮蔽时长为三枚导弹各自有效遮蔽时长之和（需扣除单枚导弹内多弹的重叠时段）。

对 $M(k)$ ，时刻  $t$  判定为“有效遮蔽”，当且仅当存在至少一枚分配给它的弹满足有效遮蔽判定的“时间条件 + 几何条件”；

$M(k)$ 的总遮蔽时长为其所有有效时刻区间的并集长度（扣除重叠时段）。

### (3) 目标函数构建

$$\max \begin{cases} \text{time\_matrix}(i, j) = \text{objective\_function}(FY_i, M_j) & (\text{预计算单机电效果}) \\ T_{total\_cover} = \sum_{k=1}^3 T_{cover, k} & (\text{总遮蔽时长}) \end{cases} \quad (22)$$

## 2. 基于“时间矩阵+预计算”的分层优化求解

由于变量维度高（41 个）且约束复杂，我们采用“预计算时间矩阵→贪心分配→调用预优化结果”的三层架构：

### (1) 上层：无人机-导弹任务分配（时间矩阵引导）

基于“遮蔽效果优先”，通过计算“无人机-导弹遮蔽时间矩阵”实现资源最优分配，步骤如下：

#### ① 计算遮蔽时间矩阵

对每架无人机 $FY(i)$ 与每枚导弹 $M(j)$ ，独立运行 PSO 优化其单机多弹参数，

得到 $FY(i)$ 对 $M(j)$ 的最大遮蔽时长 $time\_matrix(i, j)$ ，并存储对应的最优参数（飞行速度、方向、投弹/起爆时序）于 $results\_precomp(i, j)$ 。

#### ② 贪心分配（按时间降序）

基于时间矩阵，按“遮蔽时长最大化”原则分配无人机，确保每枚导弹最多 2 架无人机（避免冗余），步骤如下：

##### 1) 生成排序对

将  $time\_matrix(i, j)$  按 值 降 序 排 序 ， 得 到  $sorted\_pairs = [i, j, time\_matrix(i, j)]$ ；

##### 2) 第一轮分配

遍历排序对，优先将无人机分配给能产生最大遮蔽时长的导弹，且每枚导弹最多分配 2 架；

##### 3) 第二轮分配

对未分配的无人机，分配给仍有剩余名额（<2 架）且遮蔽时长最大的导弹；

##### 4) 生成最终分配方案

### (2) 中层：预计算优化结果（提升效率）

在分配前，已通过( $compute\_time\_matrix$ 函数预计算所有 $FY(i)$ ,  $M(j)$ 对的最优参数，分配完成后直接调用对应参数，无需重复运行 PSO。

### (3) 下层：单机多弹参数验证（约束检查）

对调用的预优化结果，再次验证约束条件：

投弹间隔约束：( $t_{drop, i2} < t_{drop, i1} + 1$ )或( $t_{drop, i3} < t_{drop, i2} + 1$ )；

起爆延迟约束： $t_{delay, ij} > \sqrt{\frac{2z_{i0}}{g}}$ ；

起爆时刻约束:  $t_{burst,ij} \geq T_{total,k}$ ;

起爆点高度约束:  $S_{burst,ij,z} \leq 0$ 。

#### (4) 算法说明

---

##### Algorithm 5 五架无人机多枚干扰弹对三枚导弹的全面干扰

---

**Input:** 5 架无人机位置, 3 枚导弹位置, 真目标位置

**Output:** 各导弹总遮蔽时间, 各无人机投放策略

```
For 每架无人机 do
    For 每枚导弹 do
        使用 PSO 优化计算该无人机对该导弹的最优遮蔽时间
        记录最优参数和遮蔽时间
    End For
End For

For 每架无人机 do
    选择能使该无人机获得最大遮蔽时间的导弹
    (限制: 每枚导弹最多分配 2 架无人机)
End For

For 每架无人机 do
    从预计算结果中获取对应导弹的最优参数
    确定投放策略(速度, 角度, 3 次投放时间和延迟时间)
End For

For 每枚导弹 do
    收集所有分配给该导弹的烟幕弹信息
    模拟导弹飞行过程, 计算总遮蔽时间
End For

输出各导弹总遮蔽时间
输出各无人机投放策略

End
```

---

### 3. 计算结果

运行程序得到最大有效遮蔽时长 20.26s, 其余参数填入 result3.xlsx 作为支撑材料上交。

## 五、模型的评价

### (一) 模型的优点

## 1. 通用性与模块化

通过统一的基础模型（运动学模型 + 遮蔽判断模型），可适用于单机单弹、单机多弹、多机协同等多种情景，具有良好的扩展性和适应性。

## 2. 协同优化机制

在多机多弹问题中引入重叠惩罚项和分层优化策略，有效避免了遮蔽时间段的冗余，提升了协同干扰效果。

# （二）模型的缺点

## 1. 计算复杂度高

计算复杂度高：尤其是在多机多弹协同优化中，变量维度高、约束复杂，PSO算法需大量迭代，计算时间较长。

## 2. 局部最优风险

PSO 算法对初始种群和参数敏感，可能陷入局部最优，影响全局最优解的获取。

## 3. 真目标采样精度

虽然通过采样点近似圆柱体表面，但采样密度和分布方式仍可能影响遮蔽判断的准确性。

# （三）改进方向

## 1. 优化算法改进

可尝试结合遗传算法、模拟退火等全局优化方法，或采用多策略混合优化算法，以提高收敛性和解的质量。

## 2. 并行计算与加速

利用 GPU 并行计算或分布式计算框架，加速高维优化问题的求解过程。

# 参考文献

- 【1】 DeepSeek, DeepSeek-V3, 深度求索, 2025-09-07
- 【2】 DeepSeek, DeepSeek-V3, 深度求索, 2025-09-07
- 【3】 DeepSeek, DeepSeek-V3, 深度求索, 2025-09-07
- 【4】 J. Kennedy and R. Eberhart, "Particle swarm optimization," in Proceedings of ICNN' 95 - International Conference on Neural Networks, 1995, vol. 4, pp. 1942 - 1948 vol.4. doi: 10.1109/ICNN.1995.488968.
- 【5】 W. Zhou, L. Lin, Y. Hong, Q. Li, X. Shen, and E. E. Kuruoglu, "Beyond Photometric Consistency: Geometry-Based Occlusion-Aware Unsupervised Light Field Disparity Estimation," IEEE Transactions on Neural Networks and Learning Systems, vol. 35, no. 11, pp. 15660 - 15674, 2024, doi: 10.1109/TNNLS.2023.3289056.

# 附录

### 支撑材料文件列表:

1. question1.m
2. question2.m
3. question3.m
4. question4.m
5. question5.m
6. calculate\_shield\_time.m
7. result1.xlsx
8. result2.xlsx
9. result3.xlsx

### 完整代码:

#### 1. question1.m

```
function t_ans = question1()
    g = 9.8;
    x_fyl = 17800 - 1.5 * 120;
    x_boom = x_fyl - 3.6 * 120;
    z_boom = 1800 - 0.5 * g * 3.6 * 3.6;
    x_M1 = 20000 - 5.1 * 300 * 10 / sqrt(101);
    z_M1 = 2000 - 5.1 * 300 / sqrt(101);
    ax_boom = [x_boom 0 z_boom];
    ax_M1 = [x_M1 0 z_M1];
    points = Points(64);
    t_ans = occlusion(ax_boom, ax_M1, points);
end

function points = Points(n)
    center = [0 200 0];
    r = 7;
    h = 10;
    points = zeros(n * 2, 3);
    ind = 1;
    theta = linspace(0, 2 * pi, n);
    z_levels = linspace(center(3), h, 2);
    for j = 1 : 2
        z = z_levels(j);
        for i = 1 : n
            x = center(1) + r * cos(theta(i));
            y = center(2) + r * sin(theta(i));
            points(ind, :) = [x, y, z];
            ind = ind + 1;
        end
    end
end

function t = occlusion(axb, axm, points)
    t_temp = 0.01;
```

```

t_start = -1;
t_end = -1;
axm_norm = norm(axm);
axm_dir = axm / axm_norm;
axm_step = 300 * axm_dir;
for i = 0 : (20 / t_temp)
    step_val = i * t_temp;
    axb_temp = axb - step_val * [0 0 3];
    axm_temp = axm - step_val * axm_step;
    v2 = axm_temp - axb_temp;
    d2 = norm(v2);
    d2_sq = d2 * d2;
    v1 = axm_temp - points;
    v1_dot_v2 = v1 * v2';
    cross_norm_sq = sum(v1.^2, 2) * d2_sq - v1_dot_v2.^2;
    d1 = sqrt(max(cross_norm_sq, 0)) ./ sqrt(sum(v1.^2, 2));
    v3 = axb_temp - points;
    d3 = sqrt(sum(v3.^2, 2));
    v1_dot_v3 = sum(v1 .* v3, 2);
    cond1 = (d1 > 10);
    cond2 = (d2 > 10) & (v1_dot_v2 < 0);
    cond3 = (d3 > 10) & (v1_dot_v3 < 0);
    if_occlusion = ~any(cond1 | cond2 | cond3);
    if if_occlusion
        if t_start == -1
            t_start = step_val;
        end
    else
        if t_start ~= -1 && t_end == -1
            t_end = step_val;
            t = t_end - t_start;
            return;
        end
    end
end
if if_occlusion
    t = 20 - t_start;
elseif t_start == -1
    t = 0;
end
end
end

```

## 2. question2.m

```

function question2
    num_particles = 500;

```

```

max_iter = 100;
dim = 4;
lb = [70, 0, 0, 0];
ub = [140, 360, 67, 19.17];
s_total = norm([20000, 0, 2000]);
T_total = s_total / 300;
positions = repmat(lb, num_particles, 1) +...
    rand(num_particles, dim) .* repmat(ub - lb, num_particles, 1);
velocities = zeros(num_particles, dim);
pbest_positions = positions;
pbest_values = -inf(num_particles, 1);
gbest_value = -inf;
gbest_position = [];
for i = 1:num_particles
    fit = smoke_cover_time(positions(i,:), T_total);
    pbest_values(i) = fit;
    if fit > gbest_value
        gbest_value = fit;
        gbest_position = positions(i,:);
    end
end
w_start = 0.9;
w_end = 0.4;
c1 = 2;
c2 = 2;
for iter = 1:max_iter
    w = w_start - (w_start - w_end) * (iter / max_iter);
    for i = 1:num_particles
        r1 = rand(1, dim);
        r2 = rand(1, dim);
        velocities(i,:) = w * velocities(i,:) + ...
            c1 * r1 .* (pbest_positions(i,:) - positions(i,:)) + ...
            c2 * r2 .* (gbest_position - positions(i,:));
        positions(i,:) = positions(i,:) + velocities(i,:);
        positions(i,:) = min(max(positions(i,:), lb), ub);
        fit = smoke_cover_time(positions(i,:), T_total);
        if fit > pbest_values(i)
            pbest_values(i) = fit;
            pbest_positions(i,:) = positions(i,:);
            if fit > gbest_value
                gbest_value = fit;
                gbest_position = positions(i,:);
            end
        end
    end
end
end

```

```

        end
        fprintf(' 第 %d 次迭代最长遮挡时间为: %.2f s\n', iter,
gbest_value);
    end
    [cover_time,          drop_point,          bomb_point]          =
smoke_cover_time(gbest_position, T_total);
    fprintf('\n 最终结果:\n');
    fprintf(' 无人机速度: %.2f m/s\n', gbest_position(1));
    fprintf(' 无人机方向 (与 x 轴正方向的夹角): %.2f ° \n',
gbest_position(2));
    fprintf(' 投放时间: %.2f s\n', gbest_position(3));
    fprintf(' 延迟爆炸时间: %.2f s\n', gbest_position(4));
    fprintf(' 干扰弹投放点: (%.2f, %.2f, %.2f) m\n', drop_point);
    fprintf(' 干扰弹引爆点: (%.2f, %.2f, %.2f) m\n', bomb_point);
    fprintf(' 最长遮盖时间: %.2f s\n', cover_time);
    disp(gbest_position);
end
function [cover_time, drop_point, bomb_point] = smoke_cover_time(X,
T_total)
    v = X(1);
    theta_deg = X(2);
    t_drop = X(3);
    t_delay = X(4);
    drone_start = [17800, 0, 1800];
    drop_point = drone_start + [v * cosd(theta_deg) * t_drop, ...
                                v * sind(theta_deg) * t_drop, ...
                                0];
    bomb_point = drop_point + [v * cosd(theta_deg) * t_delay, ...
                                v * sind(theta_deg) * t_delay, ...
                                -0.5 * 9.8 * t_delay^2];
    if (t_drop + t_delay) >= T_total || bomb_point(3) < 0
        cover_time = 0;
        return;
    end
    n_circle = 64;
    points = Points(n_circle);
    missile_start = [20000, 0, 2000];
    missile_target = [0, 0, 0];
    t_explode = t_drop + t_delay;
    missile_dir = (missile_target - missile_start) ./...
        norm(missile_target - missile_start);
    missile_pos_explode = missile_start + missile_dir * 300 * t_explode;
    cover_time = occlusion(bomb_point, missile_pos_explode, points);
end

```

```

function points = Points(n)
    center = [0 200 0];
    r = 7;
    h = 10;
    points = zeros(n * 2, 3);
    ind = 1;
    theta = linspace(0, 2 * pi, n);
    z_levels = linspace(center(3), center(3) + h, 2);
    for j = 1 : 2
        z = z_levels(j);
        for i = 1 : n
            x = center(1) + r * cos(theta(i));
            y = center(2) + r * sin(theta(i));
            points(ind, :) = [x, y, z];
            ind = ind + 1;
        end
    end
end

function t = occlusion(axb, axm, points)
    t_temp = 0.01;
    t_start = -1;
    t_end = -1;
    axm_norm = norm(axm);
    axm_dir = axm / axm_norm;
    axm_step = 300 * axm_dir;
    for i = 0 : (20 / t_temp)
        step_val = i * t_temp;
        axb_temp = axb - step_val * [0 0 3];
        axm_temp = axm - step_val * axm_step;
        v2 = axm_temp - axb_temp;
        d2 = norm(v2);
        d2_sq = d2 * d2;
        v1 = axm_temp - points;
        d1_sq = sum(v1.^2, 2);
        v1_dot_v2 = sum(v1 .* v2, 2);
        cross_norm_sq = d1_sq * d2_sq - v1_dot_v2.^2;
        d1 = sqrt(max(cross_norm_sq, 0)) ./ sqrt(d1_sq);
        v3 = axb_temp - points;
        d3 = sqrt(sum(v3.^2, 2));
        v1_dot_v3 = sum(v1 .* v3, 2);
        cond1 = (d1 > 10);
        cond2 = (d2 > 10) & (v1_dot_v2 < 0);
        cond3 = (d3 > 10) & (v1_dot_v3 < 0);
        if_occlusion = ~any(cond1 | cond2 | cond3);
    end
end

```

```

        if if_occlusion
            if t_start == -1
                t_start = step_val;
            end
        else
            if t_start ~= -1 && t_end == -1
                t_end = step_val;
                t = t_end - t_start;
                return;
            end
        end
    end
end
if if_occlusion
    t = 20 - t_start;
elseif t_start == -1
    t = 0;
else
    t = 0;
end
end
end

```

### 3. question3.m

```

function queA333()
    num_particles = 30;
    max_iter = 200;%可以再大一些，有可能还不收敛
    dim = 8;
    lb = [70, 0, 0, 0, 0, 0, 0, 0];
    ub = [140, 360, 67, 67, 67, 19.17, 19.17, 19.17];
    positions = repmat(lb, num_particles, 1) + rand(num_particles,
dim) .* repmat(ub - lb, num_particles, 1);
    velocities = zeros(num_particles, dim);
    pbest_positions = positions;
    pbest_values = zeros(num_particles, 1);
    gbest_value = -inf;
    gbest_position = [];
    for i = 1:num_particles
        fit = objective_function(positions(i, :));
        pbest_values(i) = fit;
        if fit > gbest_value
            gbest_value = fit;
            gbest_position = positions(i, :);
        end
    end
end
w_start = 0.9;

```

```

w_end = 0.4;
c1 = 2;
c2 = 2;
for iter = 1:max_iter
    w = w_start - (w_start - w_end) * (iter / max_iter);
    for i = 1:num_particles
        r1 = rand(1, dim);
        r2 = rand(1, dim);
        velocities(i, :) = w * velocities(i, :) + ...
            c1 * r1 .* (pbest_positions(i, :) - positions(i, :))
+ ...
            c2 * r2 .* (gbest_position - positions(i, :));
        positions(i, :) = positions(i, :) + velocities(i, :);
        positions(i, :) = min(max(positions(i, :), lb), ub);
        fit = objective_function(positions(i, :));
        if fit > pbest_values(i)
            pbest_values(i) = fit;
            pbest_positions(i, :) = positions(i, :);
            if fit > gbest_value
                gbest_value = fit;
                gbest_position = positions(i, :);
            end
        end
    end
    fprintf(' 第%d 次:迭代结果为: %.2f\n', iter, gbest_value);
end
v = gbest_position(1);
theta = gbest_position(2);
t_drop1 = gbest_position(3);
t_drop2 = gbest_position(4);
t_drop3 = gbest_position(5);
t_delay1 = gbest_position(6);
t_delay2 = gbest_position(7);
t_delay3 = gbest_position(8);
drone_start = [17800, 0, 1800];
dir_vector = [cosd(theta), sind(theta), 0];
drop_point1 = drone_start + v * dir_vector * t_drop1;
bomb_point1 = drop_point1 + v * dir_vector * t_delay1 - [0, 0,
0.5*9.8*t_delay1^2];
drop_point2 = drone_start + v * dir_vector * t_drop2;
bomb_point2 = drop_point2 + v * dir_vector * t_delay2 - [0, 0,
0.5*9.8*t_delay2^2];
drop_point3 = drone_start + v * dir_vector * t_drop3;
bomb_point3 = drop_point3 + v * dir_vector * t_delay3 - [0, 0,

```

```

0.5*9.8*t_delay3^2];
    fprintf(' 结果为:\n');
    fprintf(' 无人机速度: %.2f m/s\n', v);
    fprintf(' 无人机角度: %.2f deg\n', theta);
    fprintf(' 第一枚投弹时间: %.2f s\n', t_drop1);
    fprintf(' 第二枚投弹时间: %.2f s\n', t_drop2);
    fprintf(' 第三枚投弹时间: %.2f s\n', t_drop3);
    fprintf(' 第一枚延迟时间: %.2f s\n', t_delay1);
    fprintf(' 第二枚延迟时间: %.2f s\n', t_delay2);
    fprintf(' 第三枚延迟时间: %.2f s\n', t_delay3);
    fprintf(' 第一枚投放点: (%.2f, %.2f, %.2f)\n', drop_point1);
    fprintf(' 第一枚爆炸点: (%.2f, %.2f, %.2f)\n', bomb_point1);
    fprintf(' 第二枚投放点: (%.2f, %.2f, %.2f)\n', drop_point2);
    fprintf(' 第二枚爆炸点: (%.2f, %.2f, %.2f)\n', bomb_point2);
    fprintf(' 第三枚投放点: (%.2f, %.2f, %.2f)\n', drop_point3);
    fprintf(' 第三枚爆炸点: (%.2f, %.2f, %.2f)\n', bomb_point3);
    fprintf(' 总遮蔽时长: %.2f s\n', gbest_value);
end

function total_time = objective_function(X)
    v = X(1);
    theta = X(2);
    t_drop1 = X(3);
    t_drop2 = X(4);
    t_drop3 = X(5);
    t_delay1 = X(6);
    t_delay2 = X(7);
    t_delay3 = X(8);
    if t_drop2 < t_drop1 + 1 || t_drop3 < t_drop2 + 1
        total_time = 0;
        return;
    end
    if t_delay1 > 19.17 || t_drop1 + t_delay1 > 67 || ...
        t_delay2 > 19.17 || t_drop2 + t_delay2 > 67 || ...
        t_delay3 > 19.17 || t_drop3 + t_delay3 > 67
        total_time = 0;
        return;
    end
    drone_start = [17800, 0, 1800];
    dir_vector = [cosd(theta), sind(theta), 0];
    bomb_point1 = drone_start + v * dir_vector * (t_drop1 + t_delay1) -
[0, 0, 0.5*9.8*t_delay1^2];
    bomb_point2 = drone_start + v * dir_vector * (t_drop2 + t_delay2) -
[0, 0, 0.5*9.8*t_delay2^2];

```

```

    bomb_point3 = drone_start + v * dir_vector * (t_drop3 + t_delay3) -
[0, 0, 0.5*9.8*t_delay3^2];
    t_explode1 = t_drop1 + t_delay1;
    t_explode2 = t_drop2 + t_delay2;
    t_explode3 = t_drop3 + t_delay3;
    missile_start = [20000, 0, 2000];
    missile_speed = 300;
    missile_direction = -missile_start / norm(missile_start);
    missile_speed_vector = missile_speed * missile_direction;
    points = Points(16, 11);
    t_start = min([t_explode1, t_explode2, t_explode3]);
    t_end = max([t_explode1+20, t_explode2+20, t_explode3+20]);
    dt = 0.1;
    total_time = 0;
    for t = t_start:dt:t_end
        missile_pos = missile_start + missile_speed_vector * t;
        occluded = false(size(points, 1), 1);
        if t >= t_explode1 && t <= t_explode1+20
            bomb_pos1 = bomb_point1 - [0, 0, 3*(t - t_explode1)];
            for j = 1:size(points, 1)
                if ~occluded(j)
                    if is_occluded(missile_pos, points(j, :),
bomb_pos1)
                        occluded(j) = true;
                    end
                end
            end
        end
        if t >= t_explode2 && t <= t_explode2+20
            bomb_pos2 = bomb_point2 - [0, 0, 3*(t - t_explode2)];
            for j = 1:size(points, 1)
                if ~occluded(j)
                    if is_occluded(missile_pos, points(j, :),
bomb_pos2)
                        occluded(j) = true;
                    end
                end
            end
        end
        if t >= t_explode3 && t <= t_explode3+20
            bomb_pos3 = bomb_point3 - [0, 0, 3*(t - t_explode3)];
            for j = 1:size(points, 1)
                if ~occluded(j)
                    if is_occluded(missile_pos, points(j, :),

```

```

bomb_pos3)
            occluded(j) = true;
        end
    end
end
end
if all(occluded)
    total_time = total_time + dt;
end
end
end

function flag = is_occluded(A, B, C)
    AB = B - A;
    AC = C - A;
    d_AB = norm(AB);
    if d_AB < 1e-5
        flag = false;
        return;
    end
    t = dot(AC, AB) / (d_AB^2);
    if t < 0
        dist = norm(AC);
    elseif t > 1
        dist = norm(C - B);
    else
        P = A + t * AB;
        dist = norm(C - P);
    end
    flag = (dist <= 10) && (t >= 0) && (t <= 1);
end

function points = Points(n, nh)
    center = [0 200 0];
    r = 7;
    h = 10;
    points = zeros(n * nh, 3);
    ind = 1;
    theta = linspace(0, 2 * pi, n);
    z_levels = linspace(center(3), h, nh);
    for j = 1 : nh
        z = z_levels(j);
        for i = 1 : n
            x = center(1) + r * cos(theta(i));

```

```

        y = center(2) + r * sin(theta(i));
        points(ind, :) = [x, y, z];
        ind = ind + 1;
    end
end
end

```

#### 4. question4.m

```

function question4()
    missile_start = [20000, 0, 2000];
    missile_target = [0, 0, 0];
    missile_speed = 300;
    s_total = norm(missile_start - missile_target);
    T_total = s_total / missile_speed;
    drone_starts = [17800, 0, 1800;
                    12000, 1400, 1400;
                    6000, -3000, 700];
    lb_common = [70, 0, 0, 0];
    ub_common = [140, 360, 60, 40];
    results = cell(3, 1);
    intervals = [];
    num_particles = 300;
    max_iter = 300;
    lambda = 0.7;
    for drone_idx = 1:3
        fprintf('\n===== 优化无人机 FY%d =====\n', drone_idx);
        max_t_delay = sqrt(2 * drone_starts(drone_idx, 3) / 9.8);
        lb = lb_common;
        ub = ub_common;
        ub(4) = min(ub(4), max_t_delay);
        ub(3) = min(ub(3), T_total - ub(4));
        [best_params, best_interval] = pso_drone_optimization(...
            num_particles, max_iter, lb, ub, T_total, ...
            drone_starts(drone_idx, :), intervals, lambda);
        results{drone_idx} = best_params;
        intervals = [intervals; best_interval];
        fprintf('FY%d 最优参数: 速度=%.2f m/s, 角度=%.2f°, 投放时间\n', ...
            drone_idx, best_params(1), best_params(2),
            best_params(3), best_params(4));
        fprintf('遮蔽时间段: [%.2f, %.2f] s, 持续 %.2f s\n\n', ...
            best_interval(1), best_interval(2),
            diff(best_interval));
    end
    total_cover_time = compute_union_cover_time(intervals);

```

```

fprintf('\n===== 最终投放策略 =====\n');
for drone_idx = 1:3
    params = results{drone_idx};
    drone_start = drone_starts(drone_idx, :);
    drop_point = drone_start +
[params(1)*cosd(params(2))*params(3), ...
params(1)*sind(params(2))*params(3), ...
0];
    bomb_point = drop_point +
[params(1)*cosd(params(2))*params(4), ...
params(1)*sind(params(2))*params(4), ...
-0.5*9.8*params(4)^2];
    fprintf(' 无人机 FY%d:\n', drone_idx);
    fprintf('    速度: %.2f m/s, 方向角: %.2f°, 投放时间: %.2f s, 延
迟时间: %.2f s\n', ...
        params(1), params(2), params(3), params(4));
    fprintf('    投放点: (%.2f, %.2f, %.2f) m\n', drop_point);
    fprintf('    引爆点: (%.2f, %.2f, %.2f) m\n', bomb_point);
    fprintf('    遮蔽时段: [%.2f, %.2f] s (%.2f s)\n', ...
        intervals(drone_idx, 1), intervals(drone_idx, 2),
diff(intervals(drone_idx, :)));
end
fprintf('\n 总有效遮蔽时间: %.2f 秒\n', total_cover_time);
end

function [best_params, best_interval] = pso_drone_optimization(...
    num_particles, max_iter, lb, ub, T_total, drone_start,
fixed_intervals, lambda)
    dim = 4;
    positions = repmat(lb, num_particles, 1) + ...
        rand(num_particles, dim) .* repmat(ub - lb,
num_particles, 1);
    velocities = zeros(num_particles, dim);
    pbest_positions = positions;
    pbest_values = -inf(num_particles, 1);
    gbest_value = -inf;
    gbest_position = positions(1, :);
    gbest_interval = [0, 0];
    for i = 1:num_particles
        if isempty(fixed_intervals)
            fit = smoke_cover_time(positions(i, :), T_total,
drone_start);

```

```

        else
            fit = smoke_cover_time_with_penalty(...
                positions(i, :), T_total, drone_start, fixed_intervals,
lambda);
        end
        pbest_values(i) = fit;
        if fit > gbest_value
            gbest_value = fit;
            gbest_position = positions(i, :);
            [cover_time, ~, ~, ts, te] =
smoke_cover_time(positions(i, :), T_total, drone_start);
            if cover_time > 0
                t_explode = positions(i, 3) + positions(i, 4);
                gbest_interval = [t_explode + ts, t_explode + te];
            end
        end
    end
end
w_start = 0.9;
w_end = 0.4;
c1 = 2.0;
c2 = 2.0;
for iter = 1:max_iter
    w = w_start - (w_start - w_end) * (iter / max_iter);
    for i = 1:num_particles
        r1 = rand(1, dim);
        r2 = rand(1, dim);
        velocities(i, :) = w * velocities(i, :) + ...
            c1 * r1 .* (pbest_positions(i, :) - positions(i, :))
+ ...
            c2 * r2 .* (gbest_position - positions(i, :));
        positions(i, :) = positions(i, :) + velocities(i, :);
        positions(i, :) = min(max(positions(i, :), lb), ub);
        if isempty(fixed_intervals)
            fit = smoke_cover_time(positions(i, :), T_total,
drone_start);
        else
            fit = smoke_cover_time_with_penalty(...
                positions(i, :), T_total, drone_start,
fixed_intervals, lambda);
        end
        if fit > pbest_values(i)
            pbest_values(i) = fit;
            pbest_positions(i, :) = positions(i, :);
            if fit > gbest_value

```

```

        gbest_value = fit;
        gbest_position = positions(i, :);
        [cover_time, ~, ~, ts, te] =
smoke_cover_time(positions(i, :), T_total, drone_start);
        if cover_time > 0
            t_explode = positions(i, 3) + positions(i, 4);
            gbest_interval = [t_explode + ts, t_explode +
te];
        end
    end
end
end
    fprintf('迭代 %d, 最佳适应度: %.4f\n', iter, gbest_value);
end
best_params = gbest_position;
best_interval = gbest_interval;
end

function [cover_time, drop_point, bomb_point, t_start_rel, t_end_rel]
= ...
    smoke_cover_time(X, T_total, drone_start)
v = X(1);
theta_deg = X(2);
t_drop = X(3);
t_delay = X(4);
drop_point = drone_start + [v * cosd(theta_deg) * t_drop, ...
    v * sind(theta_deg) * t_drop, ...
    0];
bomb_point = drop_point + [v * cosd(theta_deg) * t_delay, ...
    v * sind(theta_deg) * t_delay, ...
    -0.5 * 9.8 * t_delay^2];
t_explode = t_drop + t_delay;
if t_explode >= T_total || bomb_point(3) < 0
    cover_time = 0;
    t_start_rel = 0;
    t_end_rel = 0;
    return;
end
n_r = 8;
n_h = 11;
points = generate_target_points(n_r, n_h);
missile_dir = (missile_target - missile_start) / norm(missile_target
- missile_start);
missile_pos_explode = missile_start + missile_dir * 300 * t_explode;

```

```

        [cover_time, t_start_rel, t_end_rel] = occlusion(...
            bomb_point, missile_pos_explode, points, missile_dir);
end

function fit = smoke_cover_time_with_penalty(...
    X, T_total, drone_start, fixed_intervals, lambda)
    [cover_time, ~, ~, t_start_rel, t_end_rel] = ...
        smoke_cover_time(X, T_total, drone_start);
    if cover_time <= 0
        fit = -1000;
        return;
    end
    t_explode = X(3) + X(4);
    S_i = t_explode + t_start_rel;
    E_i = t_explode + t_end_rel;
    overlap = 0;
    for k = 1:size(fixed_intervals, 1)
        I_k = fixed_intervals(k, :);
        overlap = overlap + max(0, min(E_i, I_k(2)) - max(S_i, I_k(1)));
    end
    fit = cover_time - lambda * overlap;
end

function points = generate_target_points(n_r, n_h)
    center = [0, 200, 0];
    radius = 7;
    height = 10;
    points = zeros(n_r * n_h, 3);
    angles = linspace(0, 2*pi, n_r);
    z_levels = linspace(center(3), center(3) + height, n_h);
    idx = 1;
    for j = 1:n_h
        z = z_levels(j);
        for i = 1:n_r
            x = center(1) + radius * cos(angles(i));
            y = center(2) + radius * sin(angles(i));
            points(idx, :) = [x, y, z];
            idx = idx + 1;
        end
    end
end

function [cover_time, t_start_rel, t_end_rel] = ...
    occlusion(bomb_point, missile_pos_explode, points,

```

```

missile_dir)
    t_temp = 0.1;
    t_start_rel = -1;
    found_start = false;
    missile_speed_vec = 300 * missile_dir;
    for step = 0:200
        t = step * t_temp;
        if t > 20
            break;
        end
        smoke_pos = bomb_point - [0, 0, 3*t];
        if smoke_pos(3) < 0
            if found_start
                t_end_rel = t;
                cover_time = t_end_rel - t_start_rel;
                return;
            else
                t_start_rel = 0;
                t_end_rel = 0;
                cover_time = 0;
                return;
            end
        end
        missile_pos = missile_pos_explode + missile_speed_vec * t;
        is_occluded = true;
        for j = 1:size(points, 1)
            target_point = points(j, :);

            if ~is_point_occluded(missile_pos, target_point,
smoke_pos)
                is_occluded = false;
                break;
            end
        end
        if is_occluded
            if ~found_start
                t_start_rel = t;
                found_start = true;
            end
        else
            if found_start
                t_end_rel = t;
                cover_time = t_end_rel - t_start_rel;
                return;
            end
        end
    end
end

```

```

        end
    end
end
if found_start
    t_end_rel = 20;
    cover_time = 20 - t_start_rel;
else
    t_start_rel = 0;
    t_end_rel = 0;
    cover_time = 0;
end
end
end

function flag = is_point_occluded(missile_pos, target_point, smoke_pos)
    MT = target_point - missile_pos;
    dist_MT = norm(MT);
    if dist_MT < 1e-6
        flag = false;
        return;
    end
    MS = smoke_pos - missile_pos;
    t = dot(MS, MT) / (dist_MT^2);
    if t < 0
        dist = norm(MS);
    elseif t > 1
        dist = norm(smoke_pos - target_point);
    else
        projection = missile_pos + t * MT;
        dist = norm(smoke_pos - projection);
    end
    flag = (dist <= 10) && (t >= 0) && (t <= 1);
end

function total_cover = compute_union_cover_time(intervals)
    if isempty(intervals)
        total_cover = 0;
        return;
    end
    sorted_intervals = sortrows(intervals, 1);
    total_cover = 0;
    current_start = sorted_intervals(1, 1);
    current_end = sorted_intervals(1, 2);
    for i = 2:size(sorted_intervals, 1)
        if sorted_intervals(i, 1) <= current_end

```

```

        current_end = max(current_end, sorted_intervals(i, 2));
    else
        total_cover = total_cover + (current_end - current_start);
        current_start = sorted_intervals(i, 1);
        current_end = sorted_intervals(i, 2);
    end
end
total_cover = total_cover + (current_end - current_start);
end

function pos = missile_start()
    pos = [20000, 0, 2000];
end

function pos = missile_target()
    pos = [0, 0, 0];
end

```

### 5. question5.m

```

function question5()
    drones = struct('name', {'FY1', 'FY2', 'FY3', 'FY4', 'FY5'}, ...
        'pos', {[17800, 0, 1800], [12000, 1400, 1400], ...
            [6000, -3000, 700], [11000, 2000, 1800], ...
            [13000, -2000, 1300]});
    missiles = struct('name', {'M1', 'M2', 'M3'}, ...
        'pos', {[20000, 0, 2000], [19000, 600, 2100], ...
            [18000, -600, 1900]});

    true_target = [0, 200, 0];
    num_particles = 600;
    max_iter = 200;
    fprintf('开始计算无人机-导弹时间矩阵...\n');
    [time_matrix, results_precomp] = compute_time_matrix(drones,
missiles, true_target, num_particles, max_iter);
    assignments = assign_drones_to_max_time(drones, missiles,
time_matrix);
    results = cell(1, numel(drones));
    for i = 1:numel(drones)
        drone_name = drones(i).name;
        missile_idx = assignments(i).missile_idx;
        missile_name = missiles(missile_idx).name;
        gbest_position=results_precomp(i,missile_idx).gbest_position;
        gbest_value = results_precomp(i, missile_idx).gbest_value;
        v = gbest_position(1);
        theta = gbest_position(2);
        t_drop = gbest_position(3:5);
    end
end

```

```

t_delay = gbest_position(6:8);

    results{i} = struct(...
        'drone_name', drone_name, ...
        'missile_name', missile_name, ...
        'v', v, ...
        'theta', theta, ...
        't_drop', t_drop, ...
        't_delay', t_delay, ...
        'individual_time', gbest_value);
end
missile_data = struct();
for m = 1:numel(missiles)
    missile_name = missiles(m).name;
    missile_pos = missiles(m).pos;
    smoke_data = [];

    for r = 1:numel(results)
        if strcmp(results{r}.missile_name, missile_name)
            drone_name = results{r}.drone_name;
            v = results{r}.v;
            theta = results{r}.theta;
            t_drop = results{r}.t_drop;
            t_delay = results{r}.t_delay;

            drone_idx = find(strcmp({drones.name}, drone_name));
            drone_pos = drones(drone_idx).pos;

            for k = 1:3
                dir_vector = [v * cosd(theta), v * sind(theta), 0];
                drop_point = drone_pos + dir_vector * t_drop(k);
                bomb_point = drop_point + dir_vector * t_delay(k) -
[0, 0, 0.5*9.8*t_delay(k)^2];
                t_explode = t_drop(k) + t_delay(k);

                smoke_data(end+1).bomb_point = bomb_point;
                smoke_data(end).t_explode = t_explode;
            end
        end
    end

    % 计算该导弹的总遮蔽时间
    missile_data(m).name = missile_name;

```

```

        missile_data(m).total_time = calculate_missile_time(...
            missile_pos, true_target, smoke_data);
    end

    % 输出最终结果
    fprintf('\n===== 最终结果 =====\n');
    total_occlusion_time = 0;
    for m = 1:numel(missile_data)
        fprintf('导弹 %s 总遮蔽时间: %.2f 秒\n', ...
            missile_data(m).name, missile_data(m).total_time);
        total_occlusion_time = total_occlusion_time +
missile_data(m).total_time;
    end
    fprintf('\n三枚导弹总遮蔽时间: %.2f 秒\n', total_occlusion_time);

    % 输出各无人机策略
    fprintf('\n===== 无人机投放策略 =====\n');
    for i = 1:numel(results)
        r = results{i};
        fprintf('\n 无人机: %s (目标: %s)\n', r.drone_name,
r.missile_name);
        fprintf('速度: %.2f m/s, 角度: %.2f° \n', r.v, r.theta);
        for k = 1:3
            fprintf('烟幕弹 %d: 投放时间 =%.2fs, 延迟时间
=%.2fs\n', ...
                k, r.t_drop(k), r.t_delay(k));
        end
    end
end

% 计算无人机-导弹时间矩阵（每个无人机对每个导弹优化）
function [time_matrix, results_precomp] = compute_time_matrix(drones,
missiles, true_target, num_particles, max_iter)
    nDrones = numel(drones);
    nMissiles = numel(missiles);
    time_matrix = zeros(nDrones, nMissiles);
    results_precomp = struct('gbest_position', cell(nDrones, nMissiles),
'gbest_value', []);

    for i = 1:nDrones
        for j = 1:nMissiles
            fprintf('优化无人机 %s 对导弹 %s ... \n', drones(i).name,
missiles(j).name);
            [gbest_position, gbest_value] = optimize_drone(...

```

```

        drones(i).pos,        missiles(j).pos,        true_target,
num_particles, max_iter);

        time_matrix(i, j) = gbest_value;
        results_precomp(i, j).gbest_position = gbest_position;
        results_precomp(i, j).gbest_value = gbest_value;
    end
end

% 显示时间矩阵
fprintf('\n 无人机-导弹遮蔽时间矩阵:\n');
drone_names = {drones.name};
missile_names = {missiles.name};
fprintf('        \t');
fprintf('%s\t', missile_names{:});
fprintf('\n');
for i = 1:nDrones
    fprintf('%s\t', drone_names{i});
    for j = 1:nMissiles
        fprintf('%.2f\t', time_matrix(i, j));
    end
    fprintf('\n');
end
end

% 分配无人机到能获得最大遮蔽时间的导弹
function assignments = assign_drones_to_max_time(drones, missiles,
time_matrix)
    nDrones = numel(drones);
    nMissiles = numel(missiles);
    assignments = struct('drone_name',    {},    'missile_name',    {},
'missile_idx', {});

    % 每个导弹最多分配两个无人机
    missile_count = zeros(1, nMissiles);

    % 贪心分配：按时间降序分配
    sorted_pairs = [];
    for i = 1:nDrones
        for j = 1:nMissiles
            sorted_pairs = [sorted_pairs; i, j, time_matrix(i, j)];
        end
    end
    sorted_pairs = sortrows(sorted_pairs, -3); % 按第三列（时间）降序

```

```

% 初始化分配结构
for i = 1:nDrones
    assignments(i).drone_name = drones(i).name;
    assignments(i).missile_name = '';
    assignments(i).missile_idx = 0;
end

% 第一轮分配：分配最佳匹配
for k = 1:size(sorted_pairs, 1)
    i = sorted_pairs(k, 1); % 无人机索引
    j = sorted_pairs(k, 2); % 导弹索引
    if assignments(i).missile_idx == 0 && missile_count(j) < 2
        missile_count(j) = missile_count(j) + 1;
        assignments(i).missile_name = missiles(j).name;
        assignments(i).missile_idx = j;
    end
end

% 第二轮分配：处理任何未分配的无人机
for i = 1:nDrones
    if assignments(i).missile_idx == 0
        % 选择还有空间的导弹
        available_missiles = find(missile_count < 2);
        if ~isempty(available_missiles)
            % 选择时间最长的可用导弹
            best_time = -1;
            best_j = 0;
            for j = available_missiles
                if time_matrix(i, j) > best_time
                    best_time = time_matrix(i, j);
                    best_j = j;
                end
            end
            if best_j > 0
                missile_count(best_j) = missile_count(best_j) + 1;
                assignments(i).missile_name =
missiles(best_j).name;
                assignments(i).missile_idx = best_j;
            end
        end
    end
end
end

```

```

% 确保所有无人机都已分配
for i = 1:nDrones
    if assignments(i).missile_idx == 0
        % 如果还有未分配的无人机，分配到第一个导弹
        assignments(i).missile_name = missiles(1).name;
        assignments(i).missile_idx = 1;
        missile_count(1) = missile_count(1) + 1;
    end
end

% 显示分配结果
fprintf('\n 无人机分配结果:\n');
for i = 1: numel(assignments)
    fprintf('%s    ->    %s\n',    assignments(i).drone_name,
assignments(i).missile_name);
end
end

% 优化单架无人机的烟幕弹策略（保持不变）
function [gbest_position, gbest_value] = optimize_drone(drone_start,
missile_start, true_target, num_particles, max_iter)
    dim = 8; % 优化变量维度
    lb = [70, 0, 0, 0, 0, 0, 0, 0]; % 下限
    ub = [140, 360, 67, 67, 67, 19.17, 19.17, 19.17]; % 上限

    % 初始化粒子群
    positions = repmat(lb, num_particles, 1) + rand(num_particles,
dim) .* repmat(ub - lb, num_particles, 1);
    velocities = zeros(num_particles, dim);
    pbest_positions = positions;
    pbest_values = zeros(num_particles, 1);
    gbest_value = -inf;
    gbest_position = [];

    % 初始评估
    for i = 1:num_particles
        fit = objective_function(positions(i, :), drone_start,
missile_start, true_target);
        pbest_values(i) = fit;
        if fit > gbest_value
            gbest_value = fit;
            gbest_position = positions(i, :);
        end
    end
end

```

```

% PSO 主循环
w_start = 0.9;
w_end = 0.4;
c1 = 2;
c2 = 2;

for iter = 1:max_iter
    w = w_start - (w_start - w_end) * (iter / max_iter);
    for i = 1:num_particles
        % 更新速度和位置
        r1 = rand(1, dim);
        r2 = rand(1, dim);
        velocities(i, :) = w * velocities(i, :) + ...
            c1 * r1 .* (pbest_positions(i, :) - positions(i, :))
+ ...
            c2 * r2 .* (gbest_position - positions(i, :));

        positions(i, :) = positions(i, :) + velocities(i, :);
        positions(i, :) = min(max(positions(i, :), lb), ub);

        % 评估新位置
        fit = objective_function(positions(i, :), drone_start,
missile_start, true_target);

        % 更新个体和全局最优
        if fit < pbest_values(i)
            pbest_values(i) = fit;
            pbest_positions(i, :) = positions(i, :);
            if fit < gbest_value
                gbest_value = fit;
                gbest_position = positions(i, :);
            end
        end
    end
    fprintf('迭代 %d: 当前最优=%.2f\n', iter, gbest_value);
end
end

% 目标函数：计算单架无人机的遮蔽时间（保持不变）
function total_time = objective_function(X, drone_start, missile_start,
true_target)
    % 解析参数
    v = X(1);

```

```

theta = X(2);
t_drop = X(3:5);
t_delay = X(6:8);

% 约束检查
if t_drop(2) < t_drop(1) + 1 || t_drop(3) < t_drop(2) + 1
    total_time = 0;
    return;
end
if any(t_delay > 19.17) || any(t_drop + t_delay > 67)
    total_time = 0;
    return;
end

% 计算烟幕弹爆炸点和爆炸时间
bomb_points = zeros(3, 3);
t_explodes = zeros(1, 3);
dir_vector = [v * cosd(theta), v * sind(theta), 0];

for k = 1:3
    drop_point = drone_start + dir_vector * t_drop(k);
    bomb_points(k, :) = drop_point + dir_vector * t_delay(k) - [0,
0, 0.5*9.8*t_delay(k)^2];
    t_explodes(k) = t_drop(k) + t_delay(k);
end

% 转换为烟幕数据结构
smoke_data = struct('bomb_point', {}, 't_explode', {});
for k = 1:3
    smoke_data(k).bomb_point = bomb_points(k, :);
    smoke_data(k).t_explode = t_explodes(k);
end

% 计算遮蔽时间
total_time = calculate_missile_time(missile_start, true_target,
smoke_data);
end

% 辅助函数（保持不变）
function points = generate_target_points(center)
    n = 6; % 圆周方向点数
    nh = 4; % 高度方向层数
    r = 7; % 半径
    h = 10; % 高度

```

```

points = zeros(n * nh, 3);
idx = 1;
theta = linspace(0, 2*pi, n);
z_levels = linspace(center(3), center(3) + h, nh);

for j = 1:nh
    z = z_levels(j);
    for i = 1:n
        x = center(1) + r * cos(theta(i));
        y = center(2) + r * sin(theta(i));
        points(idx, :) = [x, y, z];
        idx = idx + 1;
    end
end
end

% 辅助函数（保持不变）
function flag = is_occluded(A, B, C)
    AB = B - A;
    AC = C - A;
    d_AB = norm(AB);
    if d_AB < 1e-5
        flag = false;
        return;
    end
    t = dot(AC, AB) / (d_AB^2);
    if t < 0
        dist = norm(AC);
    elseif t > 1
        dist = norm(C - B);
    else
        P = A + t * AB;
        dist = norm(C - P);
    end
    flag = (dist <= 10) && (t >= 0) && (t <= 1);
end

% 计算单枚导弹的遮蔽时间（保持不变）
function total_time = calculate_missile_time(missile_start, true_target,
smoke_data)
    % 导弹参数
    missile_speed = 300;
    T0 = [0, 0, 0]; % 假目标位置

```

```

missile_dir = (T0 - missile_start) / norm(T0 - missile_start);
missile_vel = missile_speed * missile_dir;

% 生成真目标取样点
points = generate_target_points(true_target);

% 确定时间范围
if isempty(smoke_data)
    total_time = 0;
    return;
end

t_explodes = [smoke_data.t_explode];
t_start = min(t_explodes);
t_end = max(t_explodes) + 20;
dt = 0.1;
total_time = 0;

% 时间步进模拟
for t = t_start:dt:t_end
    missile_pos = missile_start + missile_vel * t;
    occluded = false(size(points, 1), 1);

    % 检查每个烟幕弹
    for s = 1:numel(smoke_data)
        if t >= smoke_data(s).t_explode && t <=
smoke_data(s).t_explode + 20
            % 计算烟幕弹当前位置（考虑下降）
            fall_dist = 3 * (t - smoke_data(s).t_explode);
            smoke_pos = smoke_data(s).bomb_point - [0, 0,
fall_dist];

            % 检查每个取样点是否被遮蔽
            for p = 1:size(points, 1)
                if ~occluded(p)
                    if is_occluded(missile_pos, points(p, :),
smoke_pos)
                        occluded(p) = true;
                    end
                end
            end
        end
    end
end
end
end

```

```

        % 如果所有点都被遮蔽，累加时间
        if all(occluded)
            total_time = total_time + dt;
        end
    end
end
end

```

## 6. calculate\_shield\_time.m

```

function bomb_effects = calculate_total_shield_time3(num_drones,
drone_params)
    % 计算多个无人机协同投放干扰弹对真目标的总遮蔽时间，并输出每个干扰
    弹的详细信息
    %
    % 输入参数:
    %   num_drones: 无人机数量 (整数)
    %   drone_params: 无人机参数结构体数组 (1×num_drones)
    %       每个结构体包含以下字段:
    %       start_pos: 无人机起始位置 [x, y, z] (米)
    %       speed: 无人机飞行速度 (米/秒) (70-140 m/s)
    %       direction: 无人机飞行方向 (角度, 0-360 度, 0=正东, 90=正北)
    %       first_drop_time: 第一枚干扰弹投放时间 (秒)
    %       interval1: 第一枚与第二枚干扰弹的投放时间间隔 (秒) (至少 1
    秒)
    %       interval2: 第二枚与第三枚干扰弹的投放时间间隔 (秒) (至少 1
    秒)
    %       delay_times: 三枚干扰弹的延迟起爆时间 [t1, t2, t3] (秒)
    %       num_bombs: 投放干扰弹数量 (1-3)
    %
    % 输出:
    %   bomb_effects: 结构体数组，包含每个干扰弹的详细信息
    %       每个结构体包含:
    %       drone_id: 无人机编号
    %       bomb_id: 干扰弹序号
    %       drop_point: 干扰弹投放位置 [x, y, z]
    %       explode_point: 干扰弹爆炸位置 [x, y, z]
    %       effects: 导弹干扰效果矩阵 (n_missiles × 2)
    %           第一列: 导弹编号 (1, 2, 3)
    %           第二列: 对该导弹的遮蔽时长 (秒)

    % 固定参数
    target_center = [0, 200, 0]; % 真目标中心位置 [x, y, z] (m)
    target_radius = 7; % 真目标圆柱半径 (m)
    target_height = 10; % 真目标圆柱高度 (m)
    g = 9.8; % 重力加速度 (m/s^2)
    missile_speed = 300; % 导弹速度 (m/s)

```

```

T0 = [0, 0, 0]; % 假目标位置 (m)
sink_speed = 3; % 烟幕下沉速度 (m/s)
max_smoke_time = 20; % 烟幕最大持续时间 (s)
dt = 0.01; % 时间步长 (s)

% 导弹初始位置 (三枚导弹)
missile_starts = [20000, 0, 2000; % M1
                  19000, 600, 2100; % M2
                  18000, -600, 1900]; % M3

% 计算每枚导弹的飞行参数
n_missiles = size(missile_starts, 1);
missile_dirs = zeros(n_missiles, 3);
missile_times = zeros(n_missiles, 1);

for i = 1:n_missiles
    vec_to_target = T0 - missile_starts(i, :);
    dist_to_target = norm(vec_to_target);
    missile_dirs(i, :) = vec_to_target / dist_to_target;
    missile_times(i) = dist_to_target / missile_speed;
end

% 检查输入参数
if num_drones ~= length(drone_params)
    error('无人机数量与参数结构体数量不一致');
end

% 生成真目标采样点
n_r = 16; % 半径方向采样点数
n_h = 11; % 高度方向采样点数
points = generate_target_points(target_center, target_radius,
target_height, n_r, n_h);

% 初始化所有干扰弹信息
all_bombs = struct('drone_id', {}, 'bomb_id', {}, 'drop_point',
{}, ...
                   'explode_point', {}, 'start_time', {},
'end_time', {});
bomb_count = 0;

% 处理每个无人机
for drone_id = 1:num_drones
    drone = drone_params(drone_id);

```

```

% 检查无人机参数
if drone.speed < 70 || drone.speed > 140
    error('无人机速度必须在 70-140 m/s 范围内');
end
if drone.num_bombs < 1 || drone.num_bombs > 3
    error('每个无人机投放干扰弹数量应为 1-3 枚');
end
if drone.intervall1 < 1 || drone.interval2 < 1
    error('投放间隔必须至少 1 秒');
end
if length(drone.delay_times) ~= 3
    error('延迟时间数组必须包含三个元素 [t1, t2, t3]');
end

% 计算无人机速度向量
drone_vel = [drone.speed * cosd(drone.direction), ...
             drone.speed * sind(drone.direction), ...
             0];

% 计算每个干扰弹的投放时间
drop_times = zeros(1, 3);
drop_times(1) = drone.first_drop_time;

if drone.num_bombs >= 2
    drop_times(2) = drop_times(1) + drone.intervall1;
end

if drone.num_bombs >= 3
    drop_times(3) = drop_times(2) + drone.interval2;
end

% 处理每个干扰弹
for bomb_id = 1:drone.num_bombs
    bomb_count = bomb_count + 1;

    % 计算投放位置
    drop_point = drone.start_pos + drop_times(bomb_id) *
drone_vel;

    % 获取对应的延迟时间
    delay_time = drone.delay_times(bomb_id);

    % 计算爆炸位置（考虑抛物线运动）
    explode_point = drop_point + drone_vel * delay_time;

```

```

        explode_point(3) = explode_point(3) - 0.5 * g * delay_time^2;

        % 计算爆炸时间和结束时间
        explode_time = drop_times(bomb_id) + delay_time;
        end_time = explode_time + max_smoke_time;

        % 存储干扰弹信息
        all_bombs(bomb_count).drone_id = drone_id;
        all_bombs(bomb_count).bomb_id = bomb_id;
        all_bombs(bomb_count).drop_point = drop_point;
        all_bombs(bomb_count).explode_point = explode_point;
        all_bombs(bomb_count).start_time = explode_time;
        all_bombs(bomb_count).end_time = end_time;
    end
end

% 初始化输出结构
bomb_effects = struct('drone_id', {}, 'bomb_id', {}, 'drop_point',
    {}, ...
        'explode_point', {}, 'effects', {});

% 如果没有干扰弹，返回空结构
if isempty(all_bombs)
    return;
end

% 计算每个干扰弹对每枚导弹的遮蔽效果
for bomb_idx = 1:length(all_bombs)
    bomb = all_bombs(bomb_idx);

    % 初始化效果矩阵 [导弹编号, 遮蔽时间]
    effects = zeros(n_missiles, 2);
    effects(:, 1) = (1:n_missiles)';

    % 对每枚导弹计算遮蔽时间
    for missile_id = 1:n_missiles
        total_time = missile_times(missile_id);
        start_t = bomb.start_time;
        end_t = min(bomb.end_time, total_time);

        % 跳过无效干扰弹
        if bomb.explode_point(3) < 0 || start_t >= total_time
            effects(missile_id, 2) = 0;
            continue;
        end
    end
end

```

```

end

% 初始化遮蔽时间变量
shield_time = 0;
shield_start = -1;

% 时间扫描（干扰弹生效期间）
for t = start_t:dt:end_t
    % 计算导弹当前位置
    missile_pos = missile_starts(missile_id, :) + ...
        missile_speed * t *
missile_dirs(missile_id, :);

    % 计算烟幕当前位置（考虑下沉）
    smoke_time = t - bomb.start_time;
    smoke_pos = bomb.explode_point - [0, 0, sink_speed *
smoke_time];

    % 检查烟幕是否有效
    if smoke_pos(3) < 0
        continue; % 烟幕已沉入地下
    end

    % 检查遮蔽条件
    all_points_covered = true;

    % 条件 1: 烟雾中心在导弹 10m 内
    if norm(missile_pos - smoke_pos) < 10
        % 如果烟雾中心在导弹 10m 内，直接认为遮蔽有效
        if shield_start < 0
            shield_start = t;
        end
        continue;
    end

    % 条件 2: 检查所有采样点是否被遮蔽
    for p_idx = 1:size(points, 1)
        point = points(p_idx, :);

        % 检查导弹与采样点的连线是否被烟幕遮蔽
        if ~is_point_occluded(missile_pos, point,
smoke_pos)
            all_points_covered = false;
            break;

```

```

        end
    end

    % 更新遮蔽状态
    if all_points_covered
        if shield_start < 0
            shield_start = t;
        end
    else
        if shield_start >= 0
            shield_time = shield_time + (t - shield_start);
            shield_start = -1;
        end
    end
end

% 处理最后一次遮蔽
if shield_start >= 0
    shield_time = shield_time + (end_t - shield_start);
end

effects(missile_id, 2) = shield_time;
end

% 存储结果
bomb_effects(bomb_idx).drone_id = bomb.drone_id;
bomb_effects(bomb_idx).bomb_id = bomb.bomb_id;
bomb_effects(bomb_idx).drop_point = bomb.drop_point;
bomb_effects(bomb_idx).explode_point = bomb.explode_point;
bomb_effects(bomb_idx).effects = effects;
end
end

function flag = is_point_occluded(missile_pos, target_point, smoke_pos)
    % 判断烟幕是否遮蔽导弹到目标点的视线
    %
    % 输入:
    %   missile_pos: 导弹当前位置 [x, y, z]
    %   target_point: 目标采样点位置 [x, y, z]
    %   smoke_pos: 烟幕中心位置 [x, y, z]
    %
    % 输出:
    %   flag: 是否遮蔽 (true/false)

```

```

% 导弹到目标点的向量
MT = target_point - missile_pos;
dist_MT = norm(MT);

% 处理零向量情况
if dist_MT < 1e-5
    flag = false;
    return;
end

% 导弹到烟幕的向量
MS = smoke_pos - missile_pos;

% 计算投影参数 t
t = dot(MS, MT) / (dist_MT^2);

% 计算烟幕到导弹-目标连线的最短距离
if t < 0
    dist = norm(MS); % 最近点为导弹位置
elseif t > 1
    dist = norm(smoke_pos - target_point); % 最近点为目标位置
else
    projection = missile_pos + t * MT; % 投影点
    dist = norm(smoke_pos - projection);
end

% 检查遮蔽条件:
% 1. 距离小于 10m
% 2. 烟雾位于导弹和目标点之间 ( $0 \leq t \leq 1$ )
flag = (dist < 10) && (t >= 0) && (t <= 1);
end

function points = generate_target_points(center, radius, height, n_r,
n_h)
% 生成真目标表面采样点
%
% 输入:
%   center: 目标中心 [x, y, z]
%   radius: 圆柱半径
%   height: 圆柱高度
%   n_r: 圆周方向采样点数
%   n_h: 高度方向采样点数
%
% 输出:

```

```
%   points: 采样点坐标矩阵 ( $n\_points \times 3$ )

points = zeros(n_r * n_h, 3);
theta = linspace(0, 2*pi, n_r);
z_levels = linspace(center(3), center(3) + height, n_h);

idx = 1;
for j = 1:n_h
    z = z_levels(j);
    for i = 1:n_r
        x = center(1) + radius * cos(theta(i));
        y = center(2) + radius * sin(theta(i));
        points(idx, :) = [x, y, z];
        idx = idx + 1;
    end
end
end
```